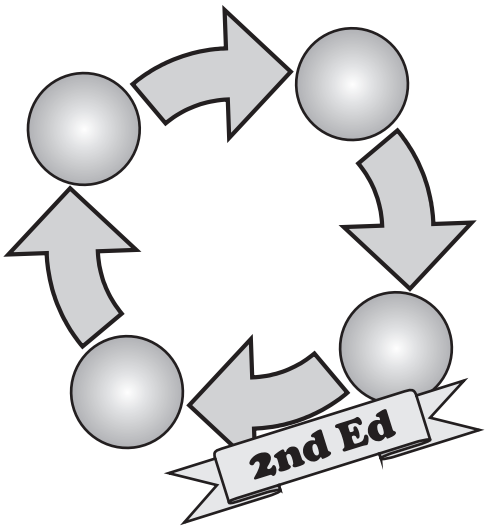
Global Optimization Algorithms – Theory and Application –



Thomas Weise
Version: 2009-06-26

Newest Version: http://www.it-weise.de/

Preface

This e-book is devoted to global optimization algorithms, which are methods to find optimal solutions for given problems. It especially focuses on Evolutionary Computation by discussing evolutionary algorithms, genetic algorithms, Genetic Programming, Learning Classifier Systems, Evolution Strategy, Differential Evolution, Particle Swarm Optimization, and Ant Colony Optimization. It also elaborates on other metaheuristics like Simulated Annealing, Extremal Optimization, Tabu Search, and Random Optimization. The book is no book in the conventional sense: Because of frequent updates and changes, it is not really intended for sequential reading but more as some sort of material collection, encyclopedia, or reference work where you can look up stuff, find the correct context, and are provided with fundamentals.
With this book, two major audience groups are addressed:

1. It can help students since we try to describe the algorithms in an understandable, consistent way and, maybe even more important, includes much of the background knowledge needed to understand them. Thus, you can find summaries on stochastic theory and theoretical computer science in Part IV on page 455. Additionally, application examples are provided which give an idea how problems can be tackled with the different techniques and what results can be expected.
2. Fellow researchers and PhD students may find the application examples helpful too. For them, in-depth discussions on the single methodologies are included that are supported with a large set of useful literature references.

If this book contains something you want to cite or reference in your work, please use the *citation suggestion* provided in Chapter D on page 591.

In order to maximize the utility of this electronic book, it contains automatic, clickable links. They are shaded with dark gray so the book is still b/w printable. You can click on

1. entries in the table of contents,
2. citation references like [916],
3. page references like "95",
4. references such as "see Figure 2.1 on page 96" to sections, figures, tables, and listings, and
5. URLs and links like "`http://www.lania.mx/~ccoello/EMOO/` [accessed 2007-10-25]".[1]

The following scenario is now for example possible: A student reads the text and finds a passage that she wants to investigate in-depth. She clicks on a citation in that seems interesting and the corresponding reference is shown. To some of the references which are online

[1] URLs are usually annotated with the date we have accessed them, like `http://www.lania.mx/~ccoello/EMOO/` [accessed 2007-10-25]. We can neither guarantee that their content remains unchanged, nor that these sites stay available. We also assume no responsibility for anything we linked to.

available, links are provided in the reference text. By clicking on such a link, the Adobe Reader®[2] will open another window and load the regarding document (or a browser window of a site that links to the document). After reading it, the student may use the "backwards" button in the navigation utility to go back to the text initially read in the e-book.

The contents of this book are divided into four parts. In the first part, different optimization technologies will be introduced and their features are described. Often, small examples will be given in order to ease understanding. In the second part starting at page 315, we elaborate on different application examples in detail. With the Sigoa framework, one possible implementation of optimization algorithms in Java, is discussed and we show how some of solutions of the previous problem instances can be realized in Part III on page 439. Finally, in the last part following at page 455, the background knowledge is provided for the rest of the book. Optimization is closely related to stochastic, and hence, an introduction into this subject can be found here. Other important background information concerns theoretical computer science and clustering algorithms.

However, this book is currently worked on. It is still in a very preliminary phase where major parts are still missing or under construction. Other sections or texts are incomplete (tagged with **TODO**). There may as well be errors in the contents or issues may be stated ambiguously (I do not have proof-readers). Additionally, the sequence of the content is not very good. Because of frequent updates, small sections may grow and become chapters, be moved to another place, merged with other sections, and so on. Thus, this book will change often. I choose to update, correct, and improve this book continuously instead of providing a new version each half year or so because I think this way it has a higher utility because it provides more information earlier. By doing so, I also risk confusing you with strange grammar and structure, so if you find something fishy, please let me know so I can correct and improve it right away.

The updates and improvements will result in new versions of the book, which will regularly appear on the website `http://www.it-weise.de/`. The direct download link to the newest version of this book is `http://www.it-weise.de/projects/book.pdf`. The LaTeX source code of this book including all graphics and the bibliography is available at `http://www.it-weise.de/projects/bookSource.zip`. The source may not always be the one of the most current version of the book. Compiling it requires multiple runs of BibTeX because of the nifty way the references are incorporated.

I would be very happy if you provide feedback, report errors or missing things that you have found, criticize something, or have any additional ideas or suggestions. Do not hesitate to contact me via my email address `tweise@gmx.de`.
Matter of fact, a large number of people helped me to improve this book over time. I have enumerated the most important contributors in Chapter C – Thank you guys, I really appreciate your help!

[2] The Adobe Reader® is available for download at `http://www.adobe.com/products/reader/` [accessed 2007-08-13].

At many places in this book we refer to Wikipedia [2219] which is a great source of knowledge. Wikipedia [2219] contains articles and definitions for many of the aspects discussed in this book. Like this book, it is updated and improved frequently. Therefore, including the links adds greatly to the book's utility, in my opinion.

Important Notice

Be aware that this version of this book marks a point of transition from the first edition to the second one. Major fractions of the text of the first edition have not yet been revised and are, thus, not included in this document. However, I believe that this version corrects many shortcomings as well as inconsistencies from the first edition plus is better structured.

Contents

Part I

Global Optimization

1

Introduction

One of the most fundamental principles in our world is the search for an optimal state. It begins in the microcosm where atoms in physics try to form bonds[1] in order to minimize the energy of their electrons [1625]. When molecules form solid bodies during the process of freezing, they try to assume energy-optimal crystal structures. These processes, of course, are not driven by any higher intention but purely result from the laws of physics.

The same goes for the biological principle of *survival of the fittest* [1940] which, together with the biological evolution [485], leads to better adaptation of the species to their environment. Here, a local optimum is a well-adapted species that dominates all other animals in its surroundings. Homo sapiens have reached this level, sharing it with ants, bacteria, flies, cockroaches, and all sorts of other creepy creatures.

As long as humankind exists, we strive for perfection in many areas. We want to reach a maximum degree of happiness with the least amount of effort. In our economy, profit and sales must be maximized and costs should be as low as possible. Therefore, optimization is one of the oldest of sciences which even extends into daily life [1519].

If something is important, general, and abstract enough, there is always a mathematical discipline dealing with it. Global optimization[2] is the branch of applied mathematics and numerical analysis that focuses on, well, optimization. The goal of global optimization is to find the best possible elements $\mathbf{x}^\star$ from a set $\mathbb{X}$ according to a set of criteria $F = \{f_1, f_2, .., f_n\}$. These criteria are expressed as mathematical functions[3], the so-called objective functions.

Definition 1.1 (Objective Function). An objective function $f : \mathbb{X} \mapsto Y$ with $Y \subseteq \mathbb{R}$ is a mathematical function which is subject to optimization.

The codomain Y of an objective function as well as its range must be a subset of the real numbers ($Y \subseteq \mathbb{R}$). The domain $\mathbb{X}$ of f is called problem space and can represent any type of elements like numbers, lists, construction plans, and so on. It is chosen according to the problem to be solved with the optimization process. Objective functions are not necessarily mere mathematical expressions, but can be complex algorithms that, for example, involve multiple simulations. Global optimization comprises all techniques that can be used to find the best elements $\mathbf{x}^\star$ in $\mathbb{X}$ with respect to such criteria $f \in F$.

In the remaining text of this introduction, we will first provide a rough classification of the different optimization techniques which we will investigate in the further course of this book (Section 1.1). In Section 1.2, we will outline how these *best* elements which we are after can be defined. We will use Section 1.3 to shed some more light onto the meaning and inter-relation of the symbols already mentioned (f, F, x, $\mathbf{x}^\star$, $\mathbb{X}$, Y, ...) and outline

[1] http://en.wikipedia.org/wiki/Chemical_bond [accessed 2007-07-12]

[2] http://en.wikipedia.org/wiki/Global_optimization [accessed 2007-07-03]

[3] The concept of mathematical functions is outlined in set theory in Definition 27.27 on page 462.

the general structure of optimization processes. If optimization was a simple thing to do, there wouldn't be a whole branch of mathematics with lots of cunning people dealing with it. In Section 1.4 we will introduce the major problems that can be encountered during optimization. We will discuss Formae as a general way of describing properties of possible solutions in Section 1.5. In this book, we will provide additional hints that point to useful literature, web links, conferences, and so on for all algorithms which we discuss. The first of these information records, dealing with global optimization in general, can be found in Section 1.6.

In the chapters to follow these introductory sections, different approaches to optimization are discussed, examples for the applications are given, and the mathematical foundation and background information is provided.

1.1 A Classification of Optimization Algorithms

In this book, we will only be able to discuss a small fraction of the wide variety of global optimization techniques [1614]. Before digging any deeper into the matter, I will attempt to provide a classification of these algorithms as overview and discuss some basic use cases.

1.1.1 Classification According to Method of Operation

Figure 1.1 sketches a rough taxonomy of global optimization methods. Generally, optimization algorithms can be divided in two basic classes: deterministic and probabilistic algorithms. Deterministic algorithms (see also Definition 30.11 on page 550) are most often used if a clear relation between the characteristics of the possible solutions and their utility for a given problem exists. Then, the search space can efficiently be explored using for example a divide and conquer scheme[4]. If the relation between a solution candidate and its "fitness" are not so obvious or too complicated, or the dimensionality of the search space is very high, it becomes harder to solve a problem deterministically. Trying it would possible result in exhaustive enumeration of the search space, which is not feasible even for relatively small problems.

Then, probabilistic algorithms[5] come into play. The initial work in this area which now has become one of most important research fields in optimization was started about 55 years ago (see [1743, 750, 219], and [287]). An especially relevant family of probabilistic algorithms are the Monte Carlo[6]-based approaches. They trade in guaranteed correctness of the solution for a shorter runtime. This does not mean that the results obtained using them are incorrect – they may just not be the global optima. On the other hand, a solution a little bit inferior to the best possible one is better than one which needs 10^{100} years to be found...

Heuristics used in global optimization are functions that help decide which one of a set of possible solutions is to be examined next. On one hand, deterministic algorithms usually employ heuristics in order to define the processing order of the solution candidates. An example for such a strategy is informed searche, as discussed in Section 17.4 on page 295. Probabilistic methods, on the other hand, may only consider those elements of the search space in further computations that have been selected by the heuristic.

Definition 1.2 (Heuristic). A heuristic[7] [1407, 1711, 1626] is a part of an optimization algorithm that uses the information currently gathered by the algorithm to help to decide which solution candidate should be tested next or how the next individual can be produced. Heuristics are usually problem class dependent.

[4] http://en.wikipedia.org/wiki/Divide_and_conquer_algorithm [accessed 2007-07-09]

[5] The common properties of probabilistic algorithms are specified in Definition 30.18 on page 552.

[6] See Definition 30.20 on page 552 for a in-depth discussion of the Monte Carlo-type probabilistic algorithms

[7] http://en.wikipedia.org/wiki/Heuristic_%28computer_science%29 [accessed 2007-07-03]

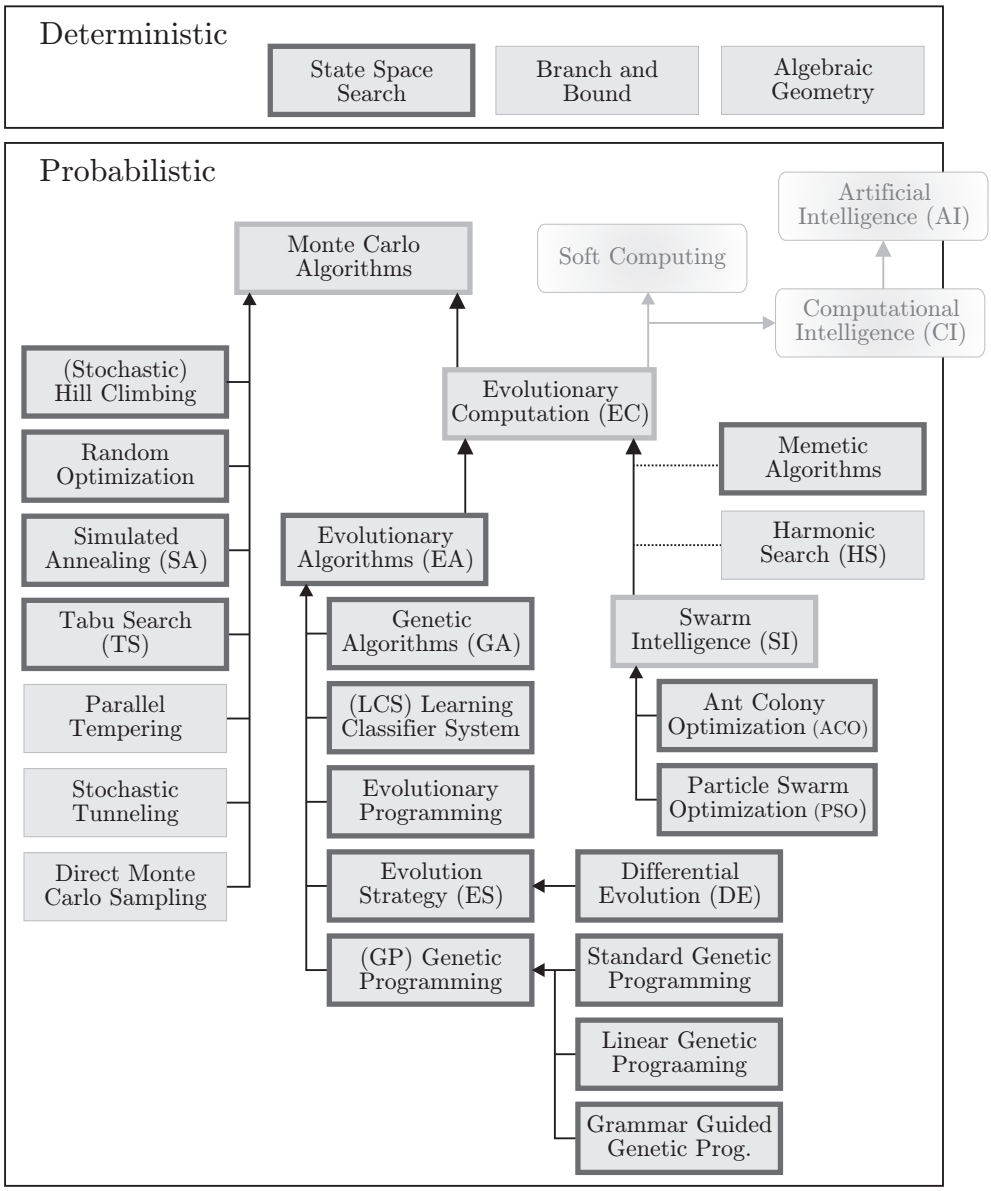


Figure 1.1: The taxonomy of global optimization algorithms.

Definition 1.3 (Metaheuristic). A metaheuristic[8] is a method for solving very general classes of problems. It combines objective functions or heuristics in an abstract and hopefully efficient way, usually without utilizing deeper insight into their structure, i. e., by treating them as black-box-procedures [813, 832, 233].

This combination is often performed stochastically by utilizing statistics obtained from samples from the search space or based on a model of some natural phenomenon or physical process. Simulated annealing, for example, decides which solution candidate to be evaluated next according to the Boltzmann probability factor of atom configurations of solidifying metal melts. Evolutionary algorithms copy the behavior of natural evolution and treat solution candidates as individuals that compete in a virtual environment. Unified

[8] http://en.wikipedia.org/wiki/Metaheuristic [accessed 2007-07-03]

models of metaheuristic optimization procedures have been proposed by Vaessens et al. [2087, 2088], Rayward-Smith [1710], Osman [1588], and Taillard et al. [1996].

An important class of probabilistic Monte Carlo metaheuristics is Evolutionary Computation[9]. It encompasses all algorithms that are based on a set of multiple solution candidates (called population) which are iteratively refined. This field of optimization is also a class of Soft Computing[10] as well as a part of the artificial intelligence[11] area. Some of its most important members are evolutionary algorithms and Swarm Intelligence, which will be discussed in-depth in this book. Besides these nature-inspired and evolutionary approaches, there exist also methods that copy physical processes like the before-mentioned Simulated Annealing, Parallel Tempering, and Raindrop Method, as well as techniques without direct real-world role model like Tabu Search and Random Optimization. As a preview of what can be found in this book, we have marked the techniques that will be discussed with a thicker border in Figure 1.1.

1.1.2 Classification According to Properties

The taxonomy just introduced classifies the optimization methods according to their algorithmic structure and underlying principles, in other words, from the viewpoint of theory. A software engineer or a user who wants to solve a problem with such an approach is however more interested in its "interfacing features" such as speed and precision.

Speed and precision are conflicting objectives, at least in terms of probabilistic algorithms. A general rule of thumb is that you can gain improvements in accuracy of optimization only by investing more time. Scientists in the area of global optimization try to push this Pareto frontier[12] further by inventing new approaches and enhancing or tweaking existing ones.

Optimization Speed

When it comes to time constraints and hence, the required speed of the optimization algorithm, we can distinguish two main types of optimization use cases.

Definition 1.4 (Online Optimization). Online optimization problems are tasks that need to be solved quickly in a time span between ten milliseconds to a few minutes. In order to find a solution in this short time, optimality is normally traded in for speed gains.

Examples for online optimization are robot localization, load balancing, services composition for business processes (see for example Section 22.2.1 on page 384), or updating a factory's machine job schedule after new orders came in. From the examples, it becomes clear that online optimization tasks are often carried out repetitively – new orders will, for instance, continuously arrive in a production facility and need to be scheduled to machines in a way that minimizes the waiting time of all jobs.

Definition 1.5 (Offline Optimization). In offline optimization problems, time is not so important and a user is willing to wait maybe even days if she can get an optimal or close-to-optimal result.

Such problems regard for example design optimization, data mining (see for instance Section 22.1 on page 373), or creating long-term schedules for transportation crews. These optimization processes will usually be carried out only once in a long time.

Before doing anything else, one must be sure about to which of these two classes the problem to be solved belongs.

[9] http://en.wikipedia.org/wiki/Evolutionary_computation [accessed 2007-09-17]
[10] http://en.wikipedia.org/wiki/Soft_computing [accessed 2007-09-17]
[11] http://en.wikipedia.org/wiki/Artificial_intelligence [accessed 2007-09-17]
[12] Pareto frontiers will be discussed in Section 1.2.2 on page 31.

TODO

Number of Criteria

Optimization algorithms can be divided in such which try to find the best values of single objective functions f and such that optimize sets F of target functions. This distinction between single-objective optimization and multi-objective optimization is discussed in depth in Section 1.2.2.

1.2 What is an optimum?

We have already said that global optimization is about finding the best possible solutions for given problems. Thus, it cannot be a bad idea to start out by discussing what it is that makes a solution *optimal*[13].

1.2.1 Single Objective Functions

In the case of optimizing a single criterion f, an optimum is either its maximum or minimum, depending on what we are looking for. If we own a manufacturing plant and have to assign incoming orders to machines, we will do this in a way that *miniminzes* the time needed to complete them. On the other hand, we will arrange the purchase of raw material, the employment of staff, and the placing of commercials in a way that *maximizes* our profit. In global optimization, it is a convention that optimization problems are most often defined as *minimizations* and if a criterion f is subject to maximization, we simply minimize its negation $(-f)$.

Figure 1.2 illustrates such a function f defined over a two-dimensional space $\mathbb{X} = (X_1, X_2)$. As outlined in this graphic, we distinguish between local and global optima. A global optimum is an optimum of the whole domain $\mathbb{X}$ while a local optimum is an optimum of only a subset of $\mathbb{X}$.

Definition 1.6 (Local Maximum). A (local) maximum $\hat{x}_l \in \mathbb{X}$ of one (objective) function $f : \mathbb{X} \mapsto \mathbb{R}$ is an input element with $f(\hat{x}_l) \geq f(x)$ for all x neighboring $\hat{x}_l$.

If $\mathbb{X} \subseteq \mathbb{R}^n$, we can write:

$$\forall \hat{x}_l \; \exists \varepsilon > 0 : f(\hat{x}_l) \geq f(x) \;\; \forall x \in \mathbb{X}, |x - \hat{x}_l| < \varepsilon \tag{1.1}$$

Definition 1.7 (Local Minimum). A (local) minimum $\check{x}_l \in \mathbb{X}$ of one (objective) function $f : \mathbb{X} \mapsto \mathbb{R}$ is an input element with $f(\check{x}_l) \leq f(x)$ for all x neighboring $\check{x}_l$.

If $\mathbb{X} \subseteq \mathbb{R}$, we can write:

$$\forall \check{x}_l \; \exists \varepsilon > 0 : f(\check{x}_l) \leq f(x) \;\; \forall x \in \mathbb{X}, |x - \check{x}_l| < \varepsilon \tag{1.2}$$

Definition 1.8 (Local Optimum). A (local) optimum $x_l^\star \in \mathbb{X}$ of one (objective) function $f : \mathbb{X} \mapsto \mathbb{R}$ is either a local maximum or a local minimum.

Definition 1.9 (Global Maximum). A global maximum $\hat{\mathbf{x}} \in x$ of one (objective) function $f : \mathbb{X} \mapsto \mathbb{R}$ is an input element with $f(\hat{\mathbf{x}}) \geq f(x) \;\; \forall x \in \mathbb{X}$.

Definition 1.10 (Global Minimum). A global minimum $\check{\mathbf{x}} \in \mathbb{X}$ of one (objective) function $f : \mathbb{X} \mapsto \mathbb{R}$ is an input element with $f(\check{\mathbf{x}}) \leq f(x) \;\; \forall x \in \mathbb{X}$.

[13] http://en.wikipedia.org/wiki/Maxima_and_minima [accessed 2007-07-03]

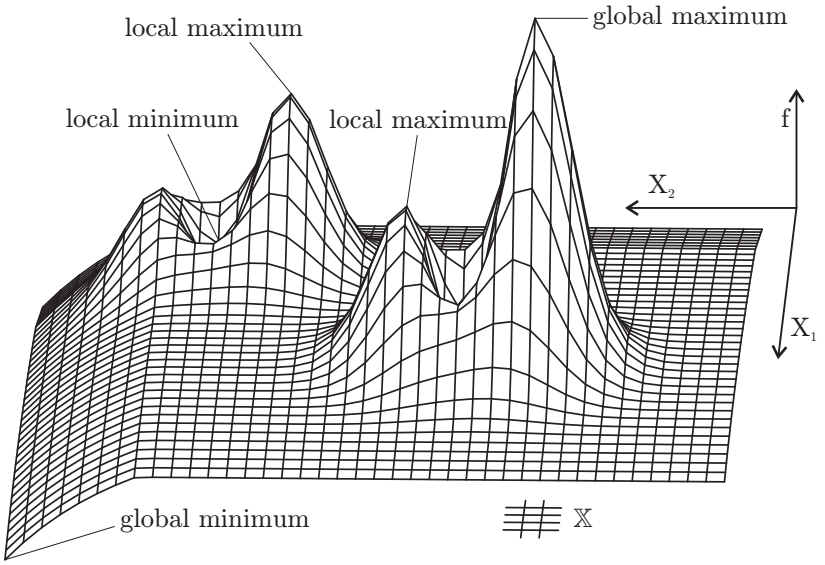


Figure 1.2: Global and local optima of a two-dimensional function.

Definition 1.11 (Global Optimum). A global optimum $\mathbf{x}^\star \in \mathbb{X}$ of one (objective) function $f : \mathbb{X} \mapsto \mathbb{R}$ is either a global maximum or a global minimum.

Even a one-dimensional function $f : \mathbb{X} = \mathbb{R} \mapsto \mathbb{R}$ may have more than one global maximum, multiple global minima, or even both in its domain $\mathbb{X}$. Take the cosine function for example: It has global maxima $\hat{\mathbf{x}}_\mathbf{i}$ at $\hat{\mathbf{x}}_\mathbf{i} = 2i\pi$ and global minima $\check{\mathbf{x}}_\mathbf{i}$ at $\check{\mathbf{x}}_\mathbf{i} = (2i+1)\pi$ for all $i \in \mathbb{Z}$. The correct solution of such an optimization problem would then be a set $\mathbf{X}^\star$ of all optimal inputs in $\mathbb{X}$ rather than a single maximum or minimum. Furthermore, the exact meaning of *optimal* is problem dependent. In single-objective optimization, it either means minimum or maximum. In multi-objective optimization, there exist a variety of approaches to define optima which we will discuss in-depth in Section 1.2.2.

Definition 1.12 (Optimal Set). The optimal set $\mathbf{X}^\star$ is the set that contains all optimal elements.

There are normally multiple, often even infinite many optimal solutions. Since the memory of our computers is limited, we can find only a finite (sub-)set of them. We thus distinguish between the global optimal set $\mathbf{X}^\star$ and the set $X^\star$ of (seemingly optimal) elements which an optimizer returns. The tasks of global optimization algorithms are

1. to find solutions that are as good as possible and
2. that are also widely different from each other [534].

The second goal becomes obvious if we assume that we have an objective function $f : \mathbb{R} \mapsto \mathbb{R}$ which is optimal for all $x \in [0, 10] \Leftrightarrow x \in \mathbf{X}^\star$. This interval contains uncountable many solutions, and an optimization algorithm may yield $X_1^\star = \{0, 0.1, 0.11, 0.05, 0.01\}$ or $X_2^\star = \{0, 2.5, 5, 7.5, 10\}$ as result. Both sets only represent a small subset of the possible solutions. The second result ($X_2^\star$), however, gives us a broader view on the optimal set. Even good optimization algorithms do not necessarily find the real global optima but may only be able to approximate them. In other words, $X_3^\star = \{-0.3, 5, 7.5, 11\}$ is also a possible result of the optimization process, although containing two sub-optimal elements.

In Chapter 19 on page 307, we will introduce different algorithms and approaches that can be used to maintain an optimal set or to select the optimal elements from a given set during an optimization process.

1.2.2 Multiple Objective Functions

Global optimization techniques are not just used for finding the maxima or minima of single functions f. In many real-world design or decision making problems, they are rather applied to sets F consisting of $n = |F|$ objective functions f_i, each representing one criterion to be optimized [537, 360, 716].

$$F = \{f_i : \mathbb{X} \mapsto Y_i \ : \ 0 < i \leq n, Y_i \subseteq \mathbb{R}\} \tag{1.3}$$

Algorithms designed to optimize such sets of objective functions are usually named with the prefix *multi-objective*, like multi-objective evolutionary algorithms which are discussed in Definition 2.2 on page 96.

Examples

Factory Example

Multi-objective optimization often means to compromise conflicting goals. If we go back to our factory example, we can specify the following objectives that all are subject to optimization:

1. Minimize the time between an incoming order and the shipment of the corresponding product.
2. Maximize profit.
3. Minimize costs for advertising, personal, raw materials etc..
4. Maximize product quality.
5. Minimize negative impact on environment.

The last two objectives seem to contradict clearly the cost minimization. Between the personal costs and the time needed for production and the product quality there should also be some kind of (contradictive) relation. The exact mutual influences between objectives can apparently become complicated and are not always obvious.

Artificial Ant Example

Another example for such a situation is the Artificial Ant problem[14] where the goal is to find the most efficient controller for a simulated ant. The efficiency of an ant should not only be measured by the amount of food it is able to pile. For every food item, the ant needs to walk to some point. The more food it piles, the longer the distance it needs to walk. If its behavior is driven by a clever program, it may walk along a shorter route which would not be discovered by an ant with a clumsy controller. Thus, the distance it has to cover to find the food or the time it needs to do so may also be considered in the optimization process. If two control programs produce the same results and one is smaller (i. e., contains fewer instructions) than the other, the smaller one should be preferred. Like in the factory example, the optimization goals conflict with each other.

From these both examples, we can gain another insight: To find the global optimum could mean to maximize one function $f_i \in F$ and to minimize another one $f_j \in F$, $(i \neq j)$. Hence, it makes no sense to talk about a *global maximum* or a *global minimum* in terms of multi-objective optimization. We will thus retreat to the notation of the set of optimal elements $\mathbf{x}^\star \in X^\star \subseteq \mathbb{X}$.

Since compromises for conflicting criteria can be defined in many ways, there exist multiple approaches to define what an optimum is. These different definitions, in turn, lead to different sets $\mathbf{X}^\star$.

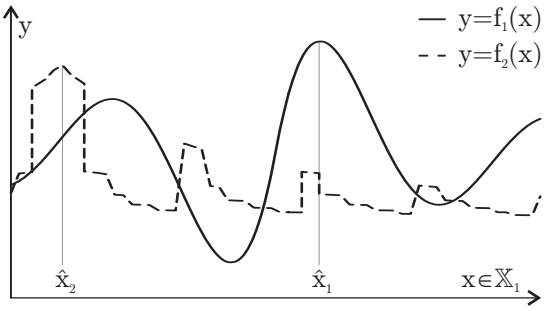


Figure 1.3: Two functions f_1 and f_2 with different maxima $\hat{\mathbf{x}}_1$ and $\hat{\mathbf{x}}_2$.

Graphical Example 1

We will discuss some of these approaches in the following by using two graphical examples for illustration purposes. In the first example pictured in Figure 1.3, we want to maximize two independent objective functions $F_1 = \{f_1, f_2\}$. Both objective functions have the real numbers $\mathbb{R}$ as problem space $\mathbb{X}_1$. The maximum (and thus, the optimum) of f_1 is $\hat{\mathbf{x}}_1$ and the largest value of f_2 is at $\hat{\mathbf{x}}_2$. In Figure 1.3, we can easily see that f_1 and f_2 are partly conflicting: Their maxima are at different locations and there even exist areas where f_1 rises while f_2 falls and vice versa.

Graphical Example 2

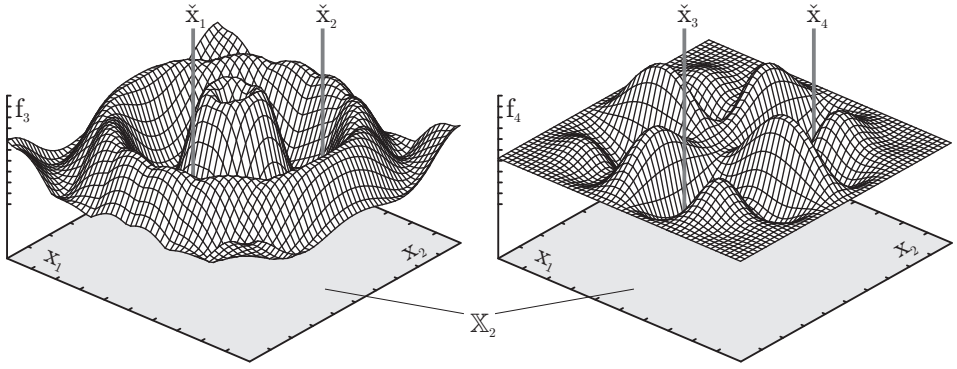


Figure 1.4: Two functions f_3 and f_4 with different minima $\check{\mathbf{x}}_1$, $\check{\mathbf{x}}_2$, $\check{\mathbf{x}}_3$, and $\check{\mathbf{x}}_4$.

The objective functions f_1 and f_2 in the first example are mappings of a one-dimensional problem space $\mathbb{X}_1$ to the real numbers that are to be maximized. In the second example sketched in Figure 1.4, we instead minimize two functions f_3 and f_4 that map a two-dimensional problem space $\mathbb{X}_2 \subset \mathbb{R}^2$ to the real numbers $\mathbb{R}$. Both functions have two global minima; the lowest values of f_3 are $\check{\mathbf{x}}_1$ and $\check{\mathbf{x}}_2$ whereas f_4 gets minimal at $\check{\mathbf{x}}_3$ and $\check{\mathbf{x}}_4$. It should be noted that $\check{\mathbf{x}}_1 \neq \check{\mathbf{x}}_2 \neq \check{\mathbf{x}}_3 \neq \check{\mathbf{x}}_4$.

[14] See Section 21.3.1 on page 354 for more details.

Weighted Sums (Linear Aggregation)

The simplest method to define what is optimal is computing a weighted sum $g(x)$ of all the functions $f_i(x) \in F$.[15] Each objective f_i is multiplied with a weight w_i representing its importance. Using signed weights also allows us to minimize one objective and to maximize another. We can, for instance, apply a weight $w_a = 1$ to an objective function f_a and the weight $wb = -1$ to the criterion f_b. By minimizing $g(x)$, we then actually minimize the first and maximize the second objective function. If we instead maximize $g(x)$, the effect would be converse and f_b would be minimized and f_a would be maximized. Either way, multi-objective problems are reduced to single-objective ones by this method.

$$g(x) = \sum_{i=1}^{n} w_i f_i(x) = \sum_{\forall f_i \in F} w_i f_i(x) \tag{1.4}$$

$$\mathbf{x}^\star \in \mathbf{X}^\star \Leftrightarrow g(\mathbf{x}^\star) \geq g(x) \ \forall x \in \mathbb{X} \tag{1.5}$$

Graphical Example 1

Figure 1.5 demonstrates optimization with the weighted sum approach for the example given in Section 1.2.2. The weights are both set to $1 = w_1 = w_2$. If we maximize $g_1(2)$, we will thus also maximize the functions f_1 and f_2. This leads to a single optimum $\mathbf{x}^\star = \hat{\mathbf{x}}$.

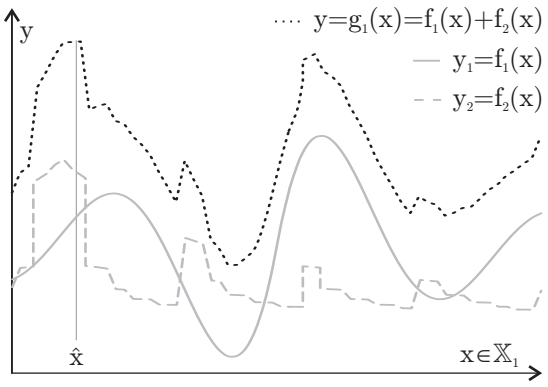


Figure 1.5: Optimization using the weighted sum approach (first example).

Graphical Example 2

The sum of the two-dimensional functions f_3 and f_4 from the second graphical example given in Section 1.2.2 is sketched in Figure 1.6. Again we set the weights w_3 and w_4 to 1. The sum g_2 however is subject to minimization. The graph of g_2 has two especially deep valleys. At the bottoms of these valleys, the two global minima $\check{\mathbf{x}}_5$ and $\check{\mathbf{x}}_6$ can be found.

Problems with Weighted Sums

The drawback of this approach is that it cannot handle functions that rise or fall with different speed[16] properly. In Figure 1.7, we have sketched the sum $g(x)$ of the two objective functions $f_1(x) = -x^2$ and $f_2(x) = e^{x-2}$. When minimizing or maximizing this sum, we

[15] This approach applies a *linear aggregation function* for fitness assignment and is therefore also often referred to as *linear aggregating*.

[16] See Section 30.1.3 on page 550
or http://en.wikipedia.org/wiki/Asymptotic_notation [accessed 2007-07-03] for related information.

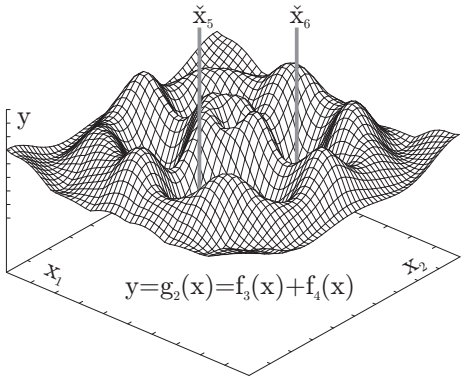


Figure 1.6: Optimization using the weighted sum approach (second example).

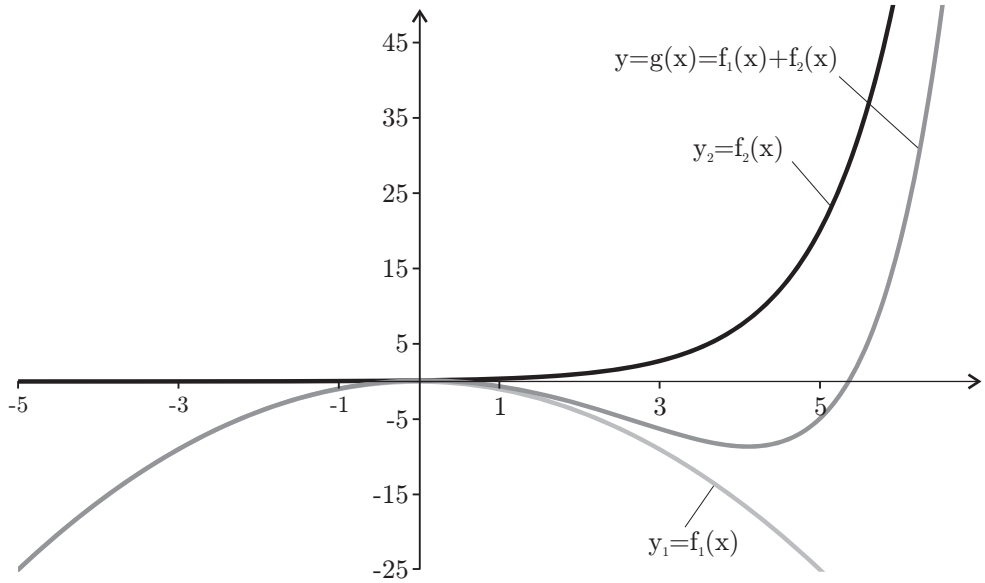


Figure 1.7: A problematic constellation for the weighted sum approach.

will always disregard one of the two functions, depending on the interval chosen. For small x, f_2 is negligible compared to f_1. For $x > 5$ it begins to outpace f_1 which, in turn, will now become negligible. Such functions cannot be added up properly using constant weights. Even if we would set w_1 to the really large number 10^{10}, f_1 will become insignificant for all $x > 40$, because $\left|\frac{-(40^2)*10^{10}}{e^{40-2}}\right| \approx 0.0005$. Therefore, weighted sums are only suitable to optimize functions that at least share the same big-**O** notation (see Section 30.1.3 on page 550). Often, it is not obvious how the objective functions will fall or rise. How can we, for instance, determine whether the objective maximizing the food piled by an Artificial Ant rises in comparison to the objective minimizing the distance walked by the simulated insect? And even if the shape of the objective functions and their complexity class were clear, the question about how to set the weights w properly still remains open in most cases [487]. In the same paper, Das and Dennis [487] also show that with weighted sum approaches, not necessarily all elements considered optimal in terms of Pareto domination will be found.

Pareto Optimization

The mathematical foundations for multi-objective optimization which considers conflicting criteria in a fair way has been laid by Vilfredo Pareto [1615] 110 years ago [1225]. Pareto optimality[17] became an important notion in economics, game theory, engineering, and social sciences [390, 2219, 1587, 752]. It defines the frontier of solutions that can be reached by trading-off conflicting objectives in an optimal manner. From this front, a decision maker (be it a human or an algorithm) can finally choose the configurations that, in his opinion, suit best [715, 716, 375, 1961, 877, 760, 177]. The notation of *optimal* in the Pareto sense is strongly based on the definition of domination:

Definition 1.13 (Domination). An element x_1 dominates (is preferred to) an element x_2 ($x_1 \vdash x_2$) if x_1 is better than x_2 in at least one objective function and not worse with respect to all other objectives. Based on the set F of objective functions f, we can write:

$$x_1 \vdash x_2 \Leftrightarrow \forall i : 0 < i \leq n \Rightarrow \omega_i f_i(x_1) \leq \omega_i f_i(x_2) \ \wedge \ \exists j : 0 < j \leq n : \omega_j f_j(x_1) < \omega_j f_j(x_2) \tag{1.6}$$

$$\omega_i = \begin{cases} 1 & \text{if } f_i \textit{ should be minimized} \\ -1 & \text{if } f_i \textit{ should be maximized} \end{cases} \tag{1.7}$$

Different from the weights in the weighted sum approach, the factors ω_i only carry sign information which allows us to maximize some objectives and to minimize some other criteria.

The Pareto domination relation defines a strict partial order (see Definition 27.31 on page 463) on the space of possible objective values. In contrast, the weighted sum approach imposes a total order by projecting it into the real numbers $\mathbb{R}$.

Definition 1.14 (Pareto Optimal). An element $\mathbf{x}^\star \in \mathbb{X}$ is Pareto optimal (and hence, part of the optimal set $\mathbf{X}^\star$) if it is not dominated by any other element in the problem space $\mathbb{X}$. In terms of Pareto optimization, $\mathbf{X}^\star$ is called the Pareto set or the Pareto Frontier.

$$\mathbf{x}^\star \in \mathbf{X}^\star \Leftrightarrow \nexists x \in \mathbb{X} : x \vdash \mathbf{x}^\star \tag{1.8}$$

Graphical Example 1

In Figure 1.8, we illustrate the impact of the definition of Pareto optimality on our first example (outlined in Section 1.2.2). We assume again that f_1 and f_2 should both be maximized and hence, $\omega_1 = \omega_2 = -1$. The areas shaded with dark gray are Pareto optimal and thus, represent the optimal set $\mathbf{X}^\star = [x_2, x_3] \cup [x_5, x_6]$ which here contains infinite many elements[18]. All other points are dominated, i. e., not optimal.

The points in the area between x_1 and x_2 (shaded in light gray) are dominated by other points in the same region or in $[x_2, x_3]$, since both functions f_1 and f_2 can be improved by increasing x. If we start at the leftmost point in $\mathbb{X}$ (which is position x_1), for instance, we can go one small step Δ to the right and will find a point $x_1 + \Delta$ dominating x_1 because $f_1(x_1 + \Delta) > f_1(x_1)$ and $f_2(x_1 + \Delta) > f_2(x_1)$. We can repeat this procedure and will always find a new dominating point until we reach x_2. x_2 demarks the global maximum of f_2, the point with the highest possible f_2 value, which cannot be dominated by any other point in $\mathbb{X}$ by definition (see Equation 1.6).

From here on, f_2 will decrease for a while, but f_1 keeps rising. If we now go a small step Δ to the right, we will find a point $x_2 + \Delta$ with $f_2(x_2 + \Delta) < f_2(x_2)$ but also $f_1(x_2 + \Delta) > f_1(x_2)$. One objective can only get better if another one degenerates. In order to increase f_1, f_2 would be decreased and vice versa and so the new point is not dominated by x_2. Although

[17] http://en.wikipedia.org/wiki/Pareto_efficiency [accessed 2007-07-03]

[18] In practice, of course, our computers can only handle finitely many elements

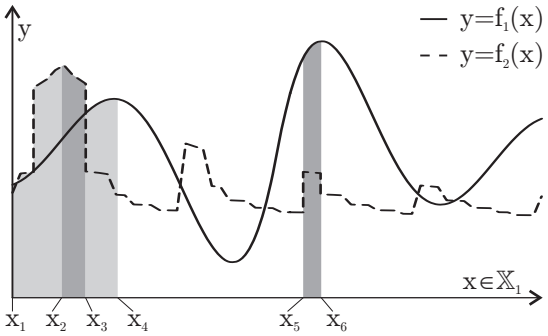


Figure 1.8: Optimization using the Pareto Frontier approach.

some of the $f_2(x)$ values of the other points $x \in [x_1, x_2)$ may be larger than $f_2(x_2 + \Delta)$, $f_1(x_2 + \Delta) > f_1(x)$ holds for all of them. This means that no point in $[x_1, x_2)$ can dominate any point in $[x_2, x_4]$ because f_1 keeps rising until x_4 is reached.

At x_3 however, f_2 steeply falls to a very low level. A level lower than $f_2(x_5)$. Since the f_1 values of the points in $[x_5, x_6]$ are also higher than those of the points in $(x_3, x_4]$, all points in the set $[x_5, x_6]$ (which also contains the global maximum of f_1) dominate those in $(x_3, x_4]$. For all the points in the white area between x_4 and x_5 and after x_6, we can derive similar relations. All of them are also dominated by the non-dominated regions that we have just discussed.

Graphical Example 2

Another method to visualize the Pareto relationship is outlined in Figure 1.9 for our second graphical example. For a certain resolution of the problem space $\mathbb{X}_2$, we have counted the number of elements that dominate each element $x \in \mathbb{X}_2$. The higher this number, the worst is the element x in terms of Pareto optimization. Hence, those solution candidates residing in the valleys of Figure 1.9 are better than those which are part of the hills. This Pareto ranking approach is also used in many optimization algorithms as part of the fitness assignment scheme (see Section 2.3.3 on page 112, for instance). A non-dominated element is, as the name says, not dominated by any other solution candidate. These elements are Pareto optimal and have a domination-count of zero. In Figure 1.9, there are four such areas $\mathbf{X_1^\star}$, $\mathbf{X_2^\star}$, $\mathbf{X_3^\star}$, and $\mathbf{X_4^\star}$.

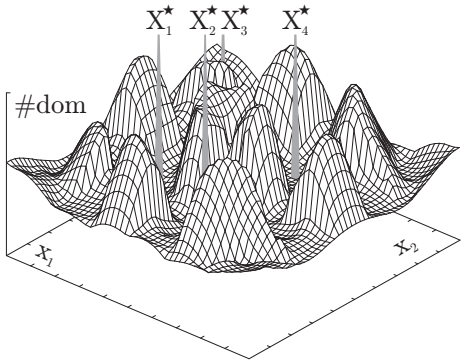


Figure 1.9: Optimization using the Pareto Frontier approach (second example).

If we compare Figure 1.9 with the plots of the two functions f_3 and f_4 in Figure 1.4, we can see that hills in the domination space occur at positions where both, f_3 and f_4 have high values. Conversely, regions of the problem space where both functions have small values are dominated by very few elements.

Besides these examples here, another illustration of the domination relation which may help understanding Pareto optimization can be found in Section 2.3.3 on page 112 (Figure 2.4 and Table 2.1).

Problems of Pure Pareto Optimization

The complete Pareto optimal set is often not the wanted result of an optimization algorithm. Usually, we are rather interested in some special areas of the Pareto front only.

Artificial Ant Example We can again take the Artificial Ant example to visualize this problem. In Section 1.2.2 on page 27 we have introduced multiple conflicting criteria in this problem.

1. Maximize the amount of food piled.
2. Minimize the distance covered or the time needed to find the food.
3. Minimize the size of the program driving the ant.

Pareto optimization may now yield for example:

1. A program consisting of 100 instructions, allowing the ant to gather 50 food items when walking a distance of 500 length units.
2. A program consisting of 100 instructions, allowing the ant to gather 60 food items when walking a distance of 5000 length units.
3. A program consisting of 10 instructions, allowing the ant to gather 1 food item when walking a distance of 5 length units.
4. A program consisting of 0 instructions, allowing the ant to gather 0 food item when walking a distance of 0 length units.

The result of the optimization process obviously contains two useless but non-dominated individuals which occupy space in the population and the non-dominated set. We also invest processing time in evaluating them, and even worse, they may dominate solutions that are not optimal but fall into the space behind the interesting part of the Pareto front. Furthermore, memory restrictions usually force us to limit the size of the list of non-dominated solutions found during the search. When this size limit is reached, some optimization algorithms use a clustering technique to prune the optimal set while maintaining diversity. On one hand, this is good since it will preserve a broad scan of the Pareto frontier. In this case on the other hand, a short but dumb program is of course very different from a longer, intelligent one. Therefore, it will be kept in the list and other solutions which differ less from each other but are more interesting for us will be discarded.

Furthermore, non-dominated elements have a higher probability of being explored further. This then leads inevitably to the creation of a great proportion of useless offspring. In the next generation, these useless offspring will need a good share of the processing time to be evaluated.

Thus, there are several reasons to force the optimization process into a wanted direction. In Section 22.2.2 on page 390 you can find an illustrative discussion on the drawbacks of strict Pareto optimization in a practical example (evolving web service compositions).

1.2.3 Constraint Handling

Such a region of interest is one of the reasons for one further extension of the definition of optimization problems: In many scenarios, p inequality constraints g and q equality constraints h may be imposed additional to the objective functions. Then, a solution candidate x is *feasible*, if and only if $g_i(x) \geq 0 \; \forall i = 1, 2, .., p$ and $h_i(x) = 0 \; \forall i = 1, 2, .., q$ holds. Obviously, only

a feasible individual can be a solution, i. e., an optimum, for a given optimization problem. Comprehensive reviews on techniques for such problems have been provided by Michalewicz [1406], Michalewicz and Schoenauer [1410], Ceollo Coello [358], and Ceollo Coello et al. [361] in the context of Evolutionary Computation.

Death Penalty

Probably the easiest way of dealing with constraints is to simply reject all infeasible solution candidates right away and not considering them any further in the optimization process. This *death penalty* [1406, 1408] can only work in problems where the feasible regions are very large and will lead the search to stagnate in cases where this is not the case. Also, the information which could be gained from the infeasible individuals is discarded with them and not used during the optimization.

Penalty Functions

Maybe one of the most popular approach for dealing with constraints, especially in the area of single-objective optimization, goes back to Courant [458] who introduced the idea of *penalty functions* in 1943. Here, the constraints are combined with the objective function f, resulting in a new function f' which is then actually optimized. The basic idea is that this combination is done in a way which ensures that an infeasible solution candidate has always a worse f'-value than a feasible one with the same objective values. In [458], this is achieved by defining f' as $f'(x) = f(x) + v\left[h(x)\right]^2$. Various similar approaches exist. Carroll [345, 346], for instance, chose a penalty function of the form $f'(x) = f(x) + v\sum_{i=1}^{p}\left[g_i(x)\right]^{-1}$ which ensures that the function g does not become zero or negative.

There are practically no limits for the ways in which a penalty for infeasibility can be integrated into the objective functions. Several researchers suggest *dynamic penalties* which incorporate the index of the current iteration of the optimizer [1063, 1560] or *adaptive penalties* which additionally utilize population statistics [1876, 1877, 875, 159]. Rigorous discussions on penalty functions have been contributed by Fiacco and McCormick [665] and Smith and Coit [1901].

Constraints as Additional Objectives

Another idea for handling constraints would be to consider them as new objective functions. If $g(x) \geq 0$ must hold, for instance, we can transform this to a new objective function $f^*(x) = \min\{-g(x), 0\}$ subject to minimization. The minimum is needed since there is no use in maximizing g further than 0 and hence, after it reached 0, the optimization pressure must be removed. An approach similar to this is Deb's Goal Programming method [536, 533].

The Method of Inequalities

General inequality constraints can also be processed according to the *Method of Inequalities* (MOI) introduced by Zakian [2304, 2305, 2306, 2307, 2308] in his seminal work on computer-aided control systems design (CACSD) [1814, 2200, 2315]. In the MOI, an area of interest is specified in form of a goal range $[\check{r}_i, \hat{r}_i]$ for each objective function f_i.

Pohlheim [1651] outlines how this approach can be combined with Pareto optimization: Based on the inequalities, three categories of solution candidates can be defined and each element $x \in \mathbb{X}$ belongs to one of them:

1. It fulfills all of the goals, i. e.,

$$\check{r}_i \leq f_i(x) \leq \hat{r}_i \; \forall i \in [1, |F|] \tag{1.9}$$

2. It fulfills some (but not all) of the goals, i. e.,

$$(\exists i \in [1,|F|] : \check{r}_i \leq f_i(x) \leq \hat{r}_i) \wedge (\exists j \in [1,|F|] : (f_j(x) < \check{r}_j) \vee (f_j(x) > \hat{r}_j)) \quad (1.10)$$

3. It fulfills none of the goals, i. e.,

$$(f_i(x) < \check{r}_i) \vee (f_i(x) > \hat{r}_i) \ \forall i \in [1,|F|] \quad (1.11)$$

Using these groups, a new comparison mechanism is created:

1. The solution candidates that fulfill all goals are preferred instead of all other individuals that either fulfill some or no goals.
2. The solution candidates that are not able to fulfill any of the goals succumb to those which fulfill at least some goals.
3. Only the solutions that are in the same group are compared on basis on the Pareto domination relation.

By doing so, the optimization process will be driven into the direction of the interesting part of the Pareto frontier. Less effort will be spent in creating and evaluating individuals in parts of the problem space that most probably do not contain any valid solution.

Graphical Example 1

In Figure 1.10, we apply the Pareto-based Method of Inequalities to our first graphical example. We impose the same goal ranges on both objectives $\hat{r}_1 = \hat{r}_2$ and $\check{r}_1 = \check{r}_2$. By doing so, the second non-dominated region from the Pareto example Figure 1.8 suddenly becomes infeasible, since f_1 rises over $\hat{r}_1$ there. Also, the greater part of the first optimal area from this example is infeasible because f_2 drops under $\check{r}_2$. In the whole domain $\mathbb{X}$ of the optimization problem, only the regions $[x_1, x_2]$ and $[x3, x4]$ fulfill all the target criteria. To these elements, Pareto comparisons are applied. It turns out that the elements in $[x3, x4]$ dominate all the elements $[x_1, x_2]$ since they provide higher values in f_1 for same values in f_2. If we scan through $[x_3, x_4]$ from left to right, we can see the f_1 rises while f_2 degenerates, which is why the elements in this area cannot dominated each other and, hence, are all optimal.

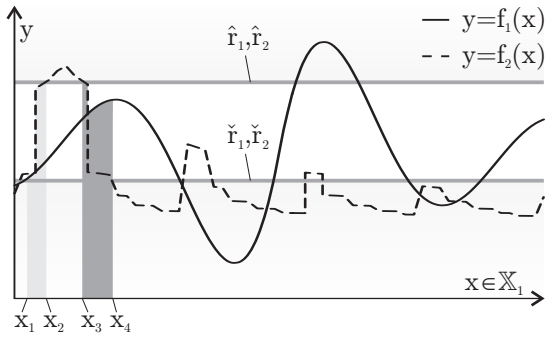


Figure 1.10: Optimization using the Pareto-based Method of Inequalities approach (first example).

Graphical Example 2

In Figure 1.11 we apply the Pareto-based Method of Inequalities to our second graphical example from Section 1.2.2. We apply two different ranges of interest $[\check{r}_3, \hat{r}_3]$ and $[\check{r}_4, \hat{r}_4]$ on f_3 and f_4 as sketched in Fig. 1.11.a.

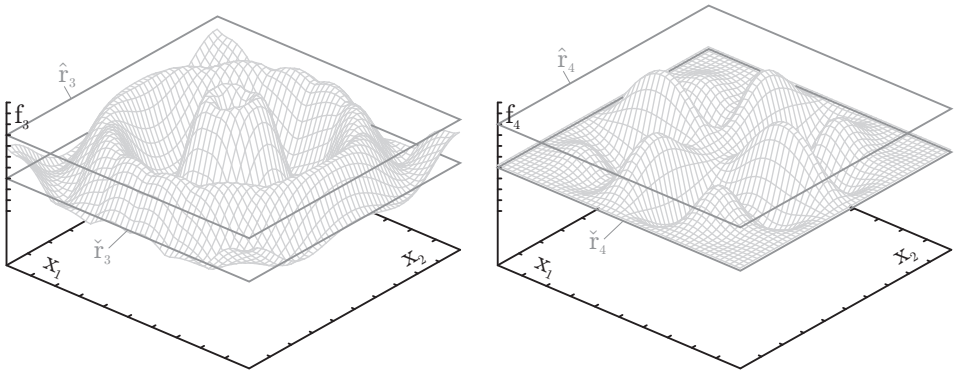


Fig. 1.11.a: The ranges applied to f_3 and f_4.

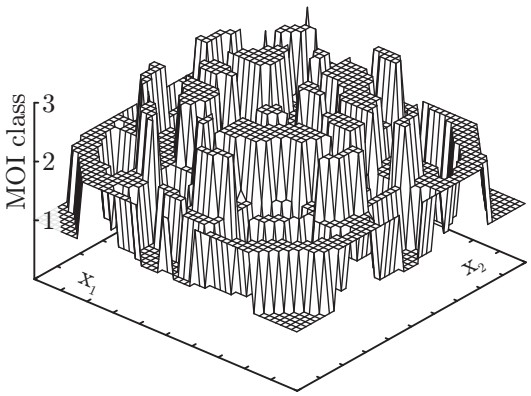


Fig. 1.11.b: The Pareto-based Method of Inequalities class division.

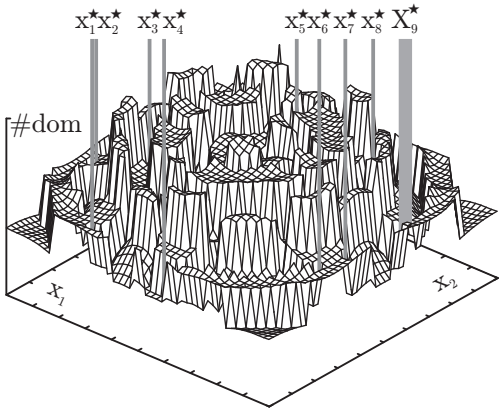


Fig. 1.11.c: The Pareto-based Method of Inequalities ranking.

Figure 1.11: Optimization using the Pareto-based Method of Inequalities approach (first example).

Like we did in the second example for Pareto optimization, we want to plot the quality of the elements in the problem space. Therefore, we first assign a number $c \in \{1, 2, 3\}$ to each of its elements in Fig. 1.11.b. This number corresponds to the classes to which the elements belong, i. e., 1 means that a solution candidate fulfills all inequalities, for an element of class 2, at least some of the constraints hold, and the elements in class 3 fail all requirements. Based on this class division, we can then perform a modified Pareto counting where each element dominates all the elements in higher classes Fig. 1.11.c. The result is that multiple single optima $\mathbf{x_1^\star}$, $\mathbf{x_2^\star}$, $\mathbf{x_3^\star}$, etc., and even a set of adjacent, non-dominated elements $\mathbf{X_9^\star}$ occurs. These elements are, again, situated at the bottom of the illustrated landscape whereas the worst solution candidates reside on hill tops.

A good overview on techniques for the Method of Inequalities is given by Whidborne et al. [2200].

Limitations and Other Methods

Other approaches for incorporating constraints into optimization are *Goal Attainment* [2233, 714] and *Goal Programming*[19] [377, 376]. Especially interesting in our context are methods

[19] http://en.wikipedia.org/wiki/Goal_programming [accessed 2007-07-03]

which have been integrated into evolutionary algorithms [2002, 536, 533, 1804, 1651], such as the popular Goal Attainment approach by Fonseca and Fleming [714] which is similar to the Pareto-MOI we have adopted from Pohlheim [1651]. Again, an overview on this subject is given by Ceollo Coello et al. in [361].

1.2.4 Unifying Approaches

External Decision Maker

All approaches for defining what optima are and how constraints should be considered are rather specific and bound to certain mathematical constructs. The more general concept of an External Decision Maker which (or who) decides which solution candidates prevail has been introduced by Fonseca and Fleming [715, 716]. One of the ideas behind "externalizing" the assessment process on what is good and what is bad is that Pareto optimization imposes only a partial order[20] on the solution candidates. In a partial order, elements may exists which neither succeed nor precede each other. As we have seen in Section 1.2.2, there can, for instance, be two individuals $x_1, x_2 \in \mathbb{X}$ with neither $x_1 \vdash x_2$ nor $x_2 \vdash x_1$. A special case of this situation is the non-dominated set, the so-called *Pareto frontier* which we try to estimate with the optimization process.

Most fitness assignment processes, however, require some sort of total order[21], where each individual is either better or worse than each other (except for the case of identical solution candidates which are, of course, equal to each other). The fitness assignment algorithms can create such a total order by themselves. One example for doing this is the Pareto ranking which we will discuss later in Section 2.3.3 on page 112, where the number of individuals dominating a solution candidate denotes its fitness.

While this method of ordering is a good default approach able of directing the search into the direction of the Pareto frontier and delivering a broad scan of it, it neglects the fact that the user of the optimization most often is not interested in the whole optimal set but has *preferences*, certain regions of interest [717]. This region will then exclude the infeasible (but Pareto optimal) programs for the Artificial Ant as discussed in Section 1.2.2. What the user wants is a detailed scan of these areas, which often cannot be delivered by pure Pareto optimization.

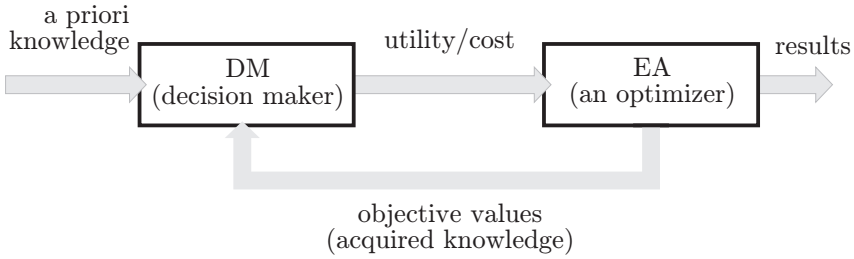


Figure 1.12: An external decision maker providing an evolutionary algorithm with utility values.

Here comes the External Decision Maker as an expression of the user's preferences [712] into play, as illustrated in Figure 1.12. The task of this decision maker is to provide a cost function $u : \mathbb{Y} \mapsto \mathbb{R}$ (or utility function, if the underlying optimizer is maximizing) which maps the space of objective values $\mathbb{Y}$ (which is usually $\mathbb{R}^n$) to the space of real numbers

[20] A definition of *partial order* relations is specified in Definition 27.31 on page 463.
[21] The concept of *total orders* is elucidated in Definition 27.32 on page 464.

$\mathbb{R}$. Since there is a total order defined on the real numbers, this process is another way of resolving the "incomparability-situation". The structure of the decision making process u can freely be defined and may incorporate any of the previously mentioned methods. u could, for example, be reduced to compute a weighted sum of the objective values, to perform an implicit Pareto ranking, or to compare individuals based on pre-specified goal-vectors. Furthermore, it may even incorporate forms of artificial intelligence, other forms of multi-criterion Decision Making, and even interaction with the user. This technique allows focusing the search onto solutions which are not only optimal in the Pareto sense, but also feasible and interesting from the viewpoint of the user.

Fonseca and Fleming make a clear distinction between fitness and cost values. Cost values have some meaning outside the optimization process and are based on user preferences. Fitness values on the other hand are an internal construct of the search with no meaning outside the optimizer (see Definition 1.35 on page 46 for more details). If External Decision Makers are applied in evolutionary algorithms or other search paradigms that are based on fitness measures, these will be computed using the values of the cost function instead of the objective functions [718, 712, 713].

Prevalence Optimization

We have now discussed various approaches which define optima in terms of multi-objective optimization and steer the search process into their direction. Let us subsume all of them in general approach. From the concept of Pareto optimization to the Method of Inequalities, the need to *compare* elements of the problem space in terms of their quality as solution for a given problem winds like a read thread through this matter. Even the weighted sum approach and the External Decision Maker do nothing else than mapping multi-dimensional vectors to the real numbers in order to make them comparable.

If we compare two solution candidates x_1 und x_2, either x_1 is better than x_2, vice versa, or both are of equal quality. Hence, there are three possible relations between two elements of the problem space. These two results can be expressed with a comparator function cmp_F.

Definition 1.15 (Comparator Function). A comparator function $\text{cmp} : \mathbb{A}^2 \mapsto \mathbb{R}$ maps all pairs of elements $(a_1, a_2) \in \mathbb{A}^2$ to the real numbers $\mathbb{R}$according to two complementing partial orders[22] R_1 and R_2:

$$R_1(a_1, a_2) \Leftrightarrow \text{cmp}(a_1, a_2) < 0 \; \forall a_1, a_2 \in \mathbb{A} \tag{1.12}$$

$$R_2(a_1, a_2) \Leftrightarrow \text{cmp}(a_1, a_2) > 0 \; \forall a_1, a_2 \in \mathbb{A} \tag{1.13}$$

$$\overline{R_1(a_1, a_2)} \wedge \overline{R_2(a_1, a_2)} \Leftrightarrow \text{cmp}(a_1, a_2) = 0 \; \forall a_1, a_2 \in \mathbb{A} \tag{1.14}$$

$$\text{cmp}(a, a) = 0 \qquad \forall a \in \mathbb{A} \tag{1.15}$$

R_1 (and hence, $\text{cmp}(a_1, a_2) < 0$) is equivalent to the precedence relation and R_2 denotes succession.

From the three defining equations, many features of cmp can be deduced. It is, for instance, transitive, i. e., $\text{cmp}(a_1, a_2) < 0 \wedge \text{cmp}(a_2, a_3) < 0 \Rightarrow \text{cmp}(a_1, a_3)) < 0$. Provided with the knowledge of the objective functions $f \in F$, such a comparator function cmp_F can be imposed on the problem spaces of our optimization problems:

Definition 1.16 (Prevalence Comparator Function). A prevalence comparator function $\text{cmp}_F : \mathbb{X}^2 \mapsto \mathbb{R}$ maps all pairs $(x_1, x_2) \in \mathbb{X}^2$ of solution candidates to the real numbers $\mathbb{R}$ according to Definition 1.15.

The subscript F in cmp_F illustrates that the comparator has access to all the values of the objective functions in addition to the problem space elements which are its parameters. As shortcut for this comparator function, we introduce the prevalence notation as follows:

[22] Partial orders are introduced in Definition 27.30 on page 463.

Definition 1.17 (Prevalence). An element x_1 prevails over an element x_2 ($x_1 \succ x_2$) if the application-dependent prevalence comparator function $\text{cmp}_F(x_1, x_2) \in \mathbb{R}$ returns a value less than 0.

$$(x_1 \succ x_2) \Leftrightarrow \text{cmp}_F(x_1, x_2) < 0 \; \forall x_1, x_2, \in \mathbb{X} \tag{1.16}$$

$$(x_1 \succ x_2) \wedge (x_2 \succ x_3) \Rightarrow x_1 \succ x_3 \; \forall x_1, x_2, x_3 \in \mathbb{X} \tag{1.17}$$

It is easy to see that we can define Pareto domination relations and Method of Inequalities-based comparisons, as well as the weighted sum combination of objective values based on this notation. Together with the fitness assignment strategies which will be introduced later in this book (see Section 2.3 on page 111), it covers many of the most sophisticated multi-objective techniques that are proposed, for instance, in [715, 1128, 2002]. By replacing the Pareto approach with prevalence comparisons, all the optimization algorithms(especially many of the evolutionary techniques) relying on domination relations can be used in their original form while offering the new ability of scanning special regions of interests of the optimal frontier.

Since the comparator function cmp_F and the prevalence relation impose a partial order on the problem space $\mathbb{X}$ like the domination relation does, we can construct the optimal set in a way very similar to Equation 1.8:

$$\mathbf{x}^\star \in \mathbf{X}^\star \Leftrightarrow \nexists x \in \mathbb{X} : x \neq \mathbf{x}^\star \wedge x \succ \mathbf{x}^\star \tag{1.18}$$

For illustration purposes, we will exercise the prevalence approach on the examples of the weighted sum $\text{cmp}_{F,F,weightedS}$ method[23] with the weights w_i as well as on the domination-based Pareto optimization[24] $\text{cmp}_{F,Pareto}$ with the objective directions ω_i:

$$\text{cmp}_{F,weightedS}(x_1, x_2) = \sum_{i=1}^{|F|} (w_i f_i(x_2) - w_i f_i(x_1)) \equiv g(x_2) - g(x_1) \tag{1.19}$$

$$\text{cmp}_{F,Pareto}(x_1, x_2) = \begin{cases} -1 & \text{if } x_1 \vdash x_2 \\ 1 & \text{if } x_2 \vdash x_1 \\ 0 & \text{otherwise} \end{cases} \tag{1.20}$$

Artificial Ant Example

With the prevalence comparator, we can also easily solve the problem stated in Section 1.2.2 by no longer encouraging the evolution of useless programs for Artificial Ants while retaining the benefits of Pareto optimization. The comparator function simple can be defined in a way that they will always be prevailed by useful programs. It therefore may incorporate the knowledge on the importance of the objective functions. Let f_1 be the objective function with an output proportional to the food piled, f_2 would denote the distance covered in order to find the food, and f_3 would be the program length. Equation 1.21 demonstrates one possible comparator function for the Artificial Ant problem.

$$\text{cmp}_{F,ant}(x_1, x_2) = \begin{cases} -1 & \text{if } (f_1(x_1) > 0 \wedge f_1(x_2) = 0) \vee \\ & \quad (f_2(x_1) > 0 \wedge f_2(x_2) = 0) \vee \\ & \quad (f_3(x_1) > 0 \wedge f_1(x_2) = 0) \\ 1 & \text{if } (f_1(x_2) > 0 \wedge f_1(x_1) = 0) \vee \\ & \quad (f_2(x_2) > 0 \wedge f_2(x_1) = 0) \vee \\ & \quad (f_3(x_2) > 0 \wedge f_1(x_1) = 0) \\ \text{cmp}_{F,Pareto}(x_1, x_2) & \text{otherwise} \end{cases} \tag{1.21}$$

[23] See Equation 1.4 on page 29 for more information on weighted sum optimization.
[24] Pareto optimization was defined in Equation 1.6 on page 31.

Later in this book, we will discuss some of the most popular optimization strategies. Although they are usually implemented based on Pareto optimization, we will always introduce them using prevalence.

1.3 The Structure of Optimization

After we have discussed what optima are and have seen a crude classification of global optimization algorithms, let us now take a look on the general structure common to all optimization processes. This structure consists of a number of well-defined spaces and sets as well as the mappings between them. Based on this structure of optimization, we will introduce the abstractions *fitness landscapes*, *problem landscape*, and *optimization problem* which will lead us to a more thorough definition of what optimization is.

1.3.1 Spaces, Sets, and Elements

In this section, we elaborate on the relation between the (possibly different) representations of solution candidates for search and for evaluation. We will show how these representations are connected and introduce *fitness* as a relative utility measures defined on sets of solution candidates. You will find that the general model introduced here applies to all the global optimization methods mentioned in this book, often in a simplified manner. One example for this structure of optimization processes is given in Figure 1.13 by using a genetic algorithm which encodes the coordinates of points in a plane into bit strings as an illustration.

The Problem Space and the Solutions therein

Whenever we tackle an optimization problem, we first have to define the type of the possible solutions. For deriving a controller for the Artificial Ant problem, we could choose programs or artificial neural networks as solution representation. If we are to find the root of a mathematical function, we would go for real numbers $\mathbb{R}$ as solution candidates and when configuring or customizing a car for a sales offer, all possible solutions are elements of the power set of all optional features. With this initial restriction to a certain type of results, we have specified the problem space $\mathbb{X}$.

Definition 1.18 (Problem Space). The problem space $\mathbb{X}$ (phenome) of an optimization problem is the set containing all elements x which could be its solution.

Usually, more than one problem space can be defined for a given optimization problem. A few lines before, we said that as problem space for finding the root of a mathematical function, the real number $\mathbb{R}$ would be fine. On the other hand, we could as well restrict ourselves to the natural numbers $\mathbb{N}$ or widen the search to the whole complex plane $\mathbb{C}$. This choice has major impact: On one hand, it determines which solutions we can possible find. On the other hand, it also has subtle influence on the search operations. Between each two different points in $\mathbb{R}$, for instance, there are infinitely many other numbers, while in $\mathbb{N}$, there are not.

In dependence on genetic algorithms, we often refer to the problem space synonymously *phenome*. The problem space $\mathbb{X}$ is often restricted by

1. *logical constraints* that rule out elements which cannot be solutions, like programs of zero length when trying to solve the Artificial Ant problem and
2. *practical constraints* that prevent us, for instance, from taking *all* real numbers into consideration in the minimization process of a real function. On our off-the-shelf CPUs or with the Java programming language, we can only use 64 bit floating point numbers. With these 64 bit, it is only possible to express numbers up to a certain precision and we cannot have more than 15 or so decimals.

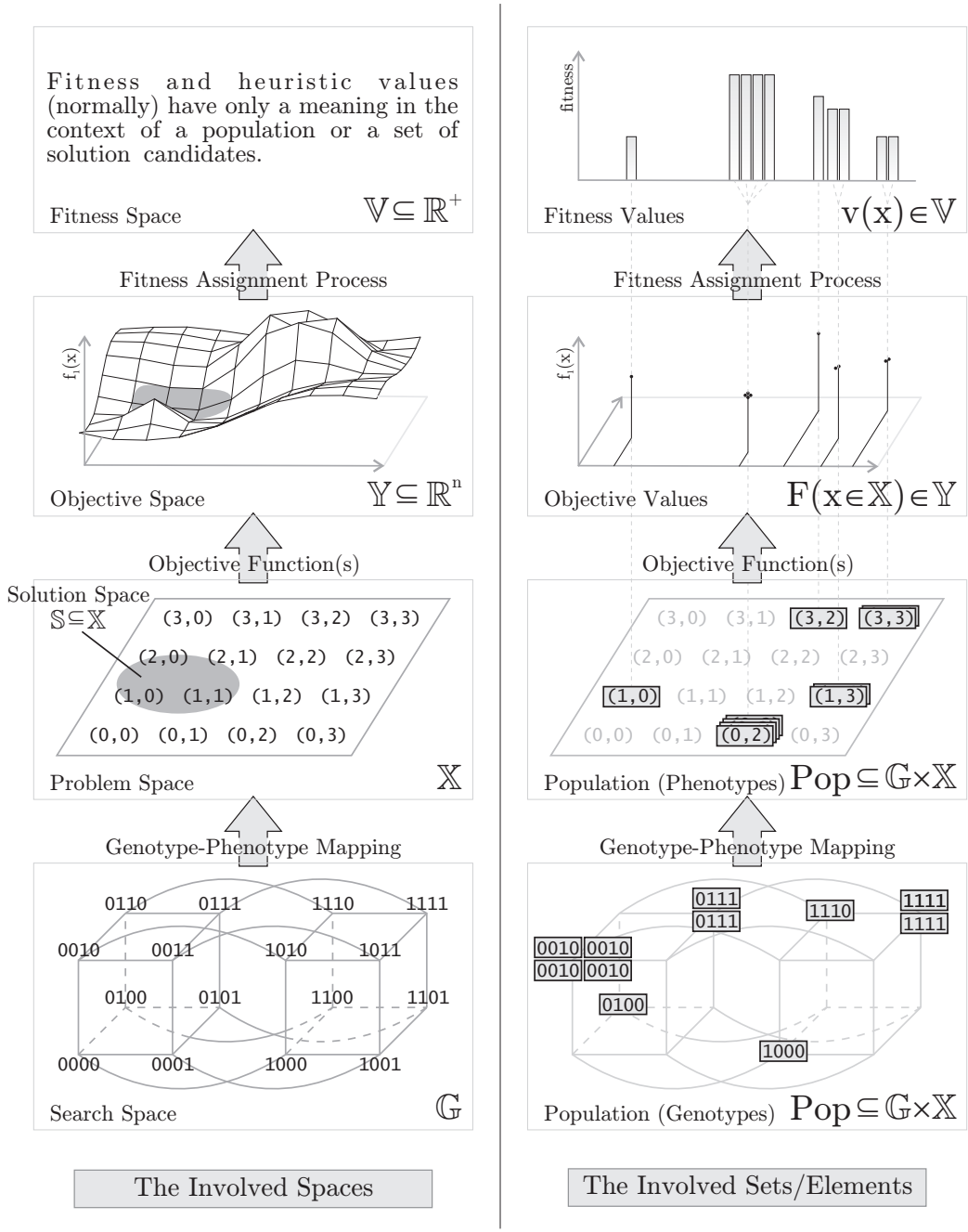


Figure 1.13: Spaces, Sets, and Elements involved in an optimization process.

Definition 1.19 (Solution Candidate). A solution candidate x is an element of the problem space $\mathbb{X}$ of a certain optimization problem.

In the context of evolutionary algorithms, solution candidates are usually called *phenotypes*. In this book, we will use both terms synonymously. Somewhere inside the problem space, the solutions of the optimization problem will be located (if the problem can actually be solved, that is).

Definition 1.20 (Solution Space). We call the union of all solutions of an optimization problem its solution space $\mathbb{S}$.

$$\mathbf{X}^\star \subseteq \mathbb{S} \subseteq \mathbb{X} \tag{1.22}$$

This solution space contains (and can be equal to) the global optimal set $\mathbf{X}^\star$. There may exist valid solutions $x \in \mathbb{S}$ which are not elements of the $\mathbf{X}^\star$, especially in the context of constraint optimization (see Section 1.2.3).

The Search Space

Definition 1.21 (Search Space). The search space $\mathbb{G}$ of an optimization problem is the set of all elements g which can be processed by the search operations.

As previously mentioned, the type of the solution candidates depends on the problem to be solved. Since there are many different applications for optimization, there are many different forms of problem spaces. It would be cumbersome to develop search operations time and again for each new problem space we encounter. Such an approach would not only be error-prone, it would also make it very hard to formulate general laws and to consolidate findings. Instead, we often reuse well-known search spaces for many different problems. Then, only a mapping between search and problem space has to be defined (see page 44). Although this is not always possible, it allows us to use more out-of-the-box software in many cases.

In dependence on genetic algorithms, we often refer to the search space synonymously as *genome*[25], a term coined by the German biologist Winkler [2241] as a portmanteau of the words gene[26] and chromosome [1267]. The genome is the whole hereditary information of organisms. This includes both, the genes and the non-coding sequences of the Deoxyribonucleic acid (DNA[27]), which is illustrated in Figure 1.14. Simply put, the DNA is a string of

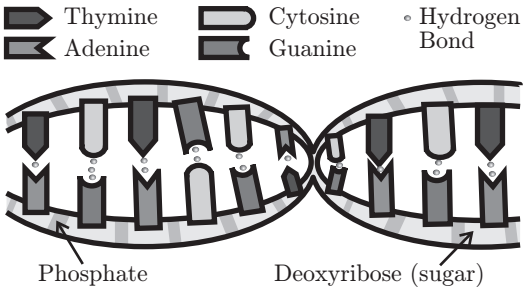


Figure 1.14: A sketch of a part of a DNA molecule.

base pairs that encodes the phenotypical characteristics of the creature it belongs to.

[25] http://en.wikipedia.org/wiki/Genome [accessed 2007-07-15]

[26] The words gene, genotype, and phenotype have, in turn, been introduced by the Danish biologist Johannsen [1056]. [2240]

[27] http://en.wikipedia.org/wiki/Dna [accessed 2007-07-03]

Definition 1.22 (Genotype). The elements $g \in \mathbb{G}$ of the search space $\mathbb{G}$ of a given optimization problem are called the *genotypes*.

The elements of the search space rarely are unstructured aggregations. Instead, they often consist of distinguishable parts, hierarchical units, or well-typed data structures. The same goes for the DNA in biology. It consists of *genes*, segments of nucleic acid, that contain the information necessary to produce RNA strings in a controlled manner[28]. A fish, for instance, may have a gene for the color of its scales. This gene, in turn, could have two possible "values" called *alleles*[29], determining whether the scales will be brown or gray. The genetic algorithm community has adopted this notation long ago and we can use it for arbitrary search spaces.

Definition 1.23 (Gene). The distinguishable units of information in a genotype that encode the phenotypical properties are called genes.

Definition 1.24 (Allele). An allele is a value of specific gene.

Definition 1.25 (Locus). The locus[30] is the position where a specific gene can be found in a genotype.

Figure 1.15 on page 45 refines the relations of genotypes and phenotypes from the initial example for the spaces in Figure 1.13 by also marking genes, alleles, and loci. In the car customizing problem also mentioned earlier, the first gene could identify the color of the automobile. Its locus would then be 0 and it could have the alleles `00`, `01`, `10`, and `11`, encoding for red, white, green, and blue, for instance. The second gene (at locus 1) with the alleles `0` or `1` may define whether or not the car comes with climate control, and so on.

The Search Operations

In some problems, the search space $\mathbb{G}$ may be identical to the problem space $\mathbb{X}$. If we go back to our previous examples, for instance, we will find that there exist a lot of optimization strategies that work directly on vectors of real numbers. When minimizing a real function, we could use such an approach (Evolution Strategies, for instance, see Chapter 5 on page 227) and set $\mathbb{G} = \mathbb{X} = \mathbb{R}$. Also, the configurations of cars may be represented as bit strings: Assume that such a configuration consists of k features, which can either be included or excluded from an offer to the customer. We can then search in the space of binary strings of this length $\mathbb{G} = \mathbb{B}^k = \{\texttt{true}, \texttt{false}\}^k$, which is exactly what genetic algorithms (discussed in Section 3.1 on page 141) do. By using their optimization capabilities, we do not need to mess with the search and selection techniques but can rely on well-researched standard operations.

Definition 1.26 (Search Operations). The search operations searchOp are used by optimization algorithms in order to explore the search space $\mathbb{G}$.

We subsume all search operations which are applied by an optimization algorithm in order to solve a given problem in the set *Op*. Search operations can be defined with different arities[31]. Equation 1.23, for instance, denotes an n-ary operator, i. e., one with n arguments. The result of a search operation is one element of the search space.

$$\text{searchOp} : \mathbb{G}^n \mapsto \mathbb{G} \tag{1.23}$$

[28] http://en.wikipedia.org/wiki/Gene [accessed 2007-07-03]
[29] http://en.wikipedia.org/wiki/Allele [accessed 2007-07-03]
[30] http://en.wikipedia.org/wiki/Locus_%28genetics%29 [accessed 2007-07-03]
[31] http://en.wikipedia.org/wiki/Arity [accessed 2008-02-15]

Mutation and crossover in genetic algorithms (see Chapter 3) are examples for unary and binary search operations, whereas Differential Evolution utilizes a ternary operator (see Section 5.5). Optimization processes are often initialized by creating random genotypes – usually the results of a search operation with zero arity (no parameters).

Search operations often involve randomized numbers. In such cases, it makes no sense to reason about their results like $\exists g_1, g_2 \in \mathbb{G} : g_2 = \text{searchOp}(g_1) \wedge \ldots$ Instead, we need to work with probabilities like $\exists g_1, g_2 \in \mathbb{G} : g_2 = P(\text{searchOp}(g_1)) > 0 \wedge \ldots$ Based on Definition 1.26, we will use the notation $Op(x)$ for the application of any of the operations searchOp $\in Op$ to the genotype x. With $Op^k(x)$ we denote k successive applications of (possibly different) search operators. If the parameter x is left away, i. e., just Op^k is written, this chain has to start with a search operation with zero arity. In the style of Badea and Stanciu [111] and Skubch [1897, 1898], we now can define:

Definition 1.27 (Completeness). A set Op of search operations searchOp is *complete* if and only if every point g_1 in the search space $\mathbb{G}$ can be reached from every other point $g_2 \in \mathbb{G}$ by applying only operations searchOp $\in Op$.

$$\forall g_1, g_2 \in \mathbb{G} \Rightarrow \exists k \in \mathbb{N} : P\big(g_1 = Op^k(g_2)\big) > 0 \tag{1.24}$$

Definition 1.28 (Weak Completeness). A set Op of search operations searchOp is *weakly complete* if and only if every point g in the search space $\mathbb{G}$ can be reached by applying only operations searchOp $\in Op$. A weakly complete set of search operations hence includes at least one parameterless function.

$$\forall g \in \mathbb{G} \Rightarrow \exists k \in \mathbb{N} : P\big(g = Op^k\big) > 0 \tag{1.25}$$

If the set of search operations is not complete, there are points in the search space which cannot be reached. Then, we are probably not able to explore the problem space adequately and possibly will not find satisfyingly good solution.

Definition 1.29 (Adjacency (Search Space)). A point g_2 is adjacent to a point g_1 in the search space $\mathbb{G}$ if it can be reached by applying a single search operation searchOp to g_1. Notice that the adjacency relation is not necessarily symmetric.

$$\text{adjacent}(g_2, g_1) = \begin{cases} \texttt{true} \text{ if } \exists \text{searchOp} \in Op : P(\text{searchOp}(g_1) = g_2) > 0 \\ \texttt{false} \text{ otherwise} \end{cases} \tag{1.26}$$

The Connection between Search and Problem Space

If the search space differs from the problem space, a translation between them is furthermore required. In our car example, we would need to transform the binary strings processed by the genetic algorithm to objects which represent the corresponding car configurations and can be processed by the objective functions.

Definition 1.30 (Genotype-Phenotype Mapping). The genotype-phenotype mapping (GPM, or ontogenic mapping [1619]) $\text{gpm} : \mathbb{G} \mapsto \mathbb{X}$ is a left-total[32] binary relation which maps the elements of the search space $\mathbb{G}$ to elements in the problem space $\mathbb{X}$.

$$\forall g \in \mathbb{G}\ \exists x \in \mathbb{X} : \text{gpm}(g) = x \tag{1.27}$$

The only hard criterion we impose on genotype-phenotype mappings in this book is left-totality, i. e., that they map each element of the search space to at least one solution candidate. They *may* be functional relations if they are deterministic. Although it is possible to create mappings which involve random numbers and, hence, cannot be considered to be

[32] See Equation 27.51 on page 461 to 5 on page 462 for an outline of the properties of binary relations.

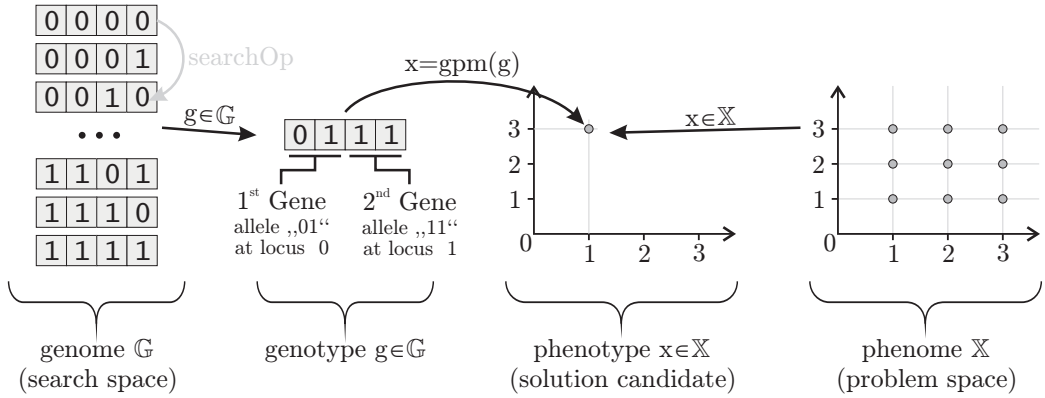


Figure 1.15: The relation of genome, genes, and the problem space.

functions in the mathematical sense of Section 27.7.1 on page 462. Then, Equation 1.27 would need to be rewritten to Equation 1.28.

$$\forall g \in \mathbb{G}\ \exists x \in \mathbb{X} : P(\text{gpm}(g) = x) > 0 \tag{1.28}$$

Genotype-phenotype mappings *should* further be surjective [1694], i. e., relate at least one genotype to each element of the problem space. Otherwise, some solution candidates can never be found and evaluated by the optimization algorithm and there is no guarantee whether the solution of a given problem can be discovered or not. If a genotype-phenotype mapping is injective, which means that it assigns distinct phenotypes to distinct elements of the search space, we say that it is free from *redundancy*. There are different forms of redundancy, some are considered to be harmful for the optimization process, others have positive influence[33]. Most often, GPMs are not bijective (since they are neither necessarily injective nor surjective). Nevertheless, if a genotype-phenotype mapping is bijective, we can construct an inverse mapping $\text{gpm}^{-1} : \mathbb{X} \mapsto \mathbb{G}$.

$$\text{gpm}^{-1}(x) = g \Leftrightarrow \text{gpm}(g) = x\ \forall x \in \mathbb{X}, g \in \mathbb{G} \tag{1.29}$$

Based on the genotype-phenotype mapping, we can also define an adjacency relation for the problem space, which, of course, is also not necessarily symmetric.

Definition 1.31 (Adjacency (Problem Space)). A point x_2 is adjacent to a point x_1 in the problem space $\mathbb{X}$ if it can be reached by applying a single search operation searchOp to their corresponding elements in the problem space.

$$\text{adjacent}(x_2, x_1) = \begin{cases} \texttt{true} & \text{if } \exists g_1, g_2 : x_1 = \text{gpm}(g_1) \wedge x_2 = \text{gpm}(g_2) \wedge \text{adjacent}(g_2, g_1) \\ \texttt{false} & \text{otherwise} \end{cases} \tag{1.30}$$

By the way, we now have the means to define the term *local optimum* clearer. The original Definition 1.8 only applies to single objective functions, but with the use of the adjacency relation adjacent, the prevalence criterion $\succ$, and the connection between the search space and the problem space gpm, we clarify it for multiple objectives.

Definition 1.32 (Local Optimum). A (local) optimum $x_l^\star \in \mathbb{X}$ of a set of objective functions F function is not worse than all points adjacent to it.

$$\forall x_l^\star \in \mathbb{G} \Rightarrow \left(\forall x \in \mathbb{X} : \text{adjacent}(x, x_l^\star) \Rightarrow \overline{x \succ x_l^\star}\right) \tag{1.31}$$

[33] See Section 1.4.5 on page 67 for more information.

The Objective Space and Optimization Problems

After the appropriate problem space has been defined, the search space has been selected and a translation between them (if needed) was created, we are almost ready to feed the problem to a global optimization algorithm. The main purpose of such an algorithm obviously is to find as many elements as possible from the solution space – We are interested in the solution candidates with the best possible evaluation results. This evaluation is performed by the set F of n objective functions $f \in F$, each contributing one numerical value describing the characteristics of a solution candidate x.[34]

Definition 1.33 (Objective Space). The objective space $\mathbb{Y}$ is the space spanned by the codomains of the objective functions.

$$F = \{f_i : \mathbb{X} \mapsto Y_i \, : \, 0 < i \leq n, Y_i \subseteq \mathbb{R}\} \Rightarrow \mathbb{Y} = Y_1 \times Y_2 \times .. \times Y_n \tag{1.32}$$

The set F maps the elements x of the problem space $\mathbb{X}$ to the objective space $\mathbb{Y}$ and, by doing so, gives the optimizer information about their qualities as solutions for a given problem.

Definition 1.34 (Optimization Problem). An optimization problem is defined by a five-tuple $(\mathbb{X}, F, \mathbb{G}, Op, \text{gpm})$ specifying the problem space $\mathbb{X}$, the objective functions F, the search space $\mathbb{G}$, the set of search operations Op, and the genotype-phenotype mapping gpm. In theory, such an optimization problem *can* always be solved if Op is complete and the gpmis surjective.

Generic search and optimization algorithms find optimal elements if provided with an optimization problem defined in this way. Evolutionary algorithms, which we will discuss later in this book, are generic in this sense. Other optimization methods, like genetic algorithmsfor example, may be more specialized and work with predefined search spaces and search operations.

Fitness as a Relative Measure of Utility

When performing a multi-objective optimization, i. e., $n = |F| > 1$, the elements of $\mathbb{Y}$ are vectors in $\mathbb{R}^n$. In Section 1.2.2 on page 27, we have seen that such vectors cannot always be compared directly in a consistent way and that we need some (comparative) measure for what is "good". In many optimization techniques, especially in evolutionary algorithms, this measure is used to map the objective space to a subset $\mathbb{V}$ of the positive real numbers $\mathbb{R}^+$. For each solution candidate, this single real number represents its *fitness* as solution for the given optimization problem. The process of computing such a fitness value is often not solely depending on the absolute objective values of the solution candidates but also on those of the other phenotypes known. It could, for instance, be position of a solution candidate in the list of investigated elements sorted according to the Pareto relation. Hence, fitness values often only have a meaning inside the optimization process [712] and may change by time, even if the objective values stay constant. In deterministic optimization methods, the value of a heuristic function which approximates how many modifications we will have to apply to the element in order to reach a feasible solution can be considered as the fitness.

Definition 1.35 (Fitness). The fitness[35] value $v(x) \in \mathbb{V}$ of an element x of the problem space $\mathbb{X}$ corresponds to its utility as solution or its priority in the subsequent steps of the optimization process. The space spanned by all possible fitness values $\mathbb{V}$ is normally a subset of the positive real numbers $\mathbb{V} \subseteq \mathbb{R}^+$.

[34] See also Equation 1.3 on page 27.

[35] http://en.wikipedia.org/wiki/Fitness_(genetic_algorithm) [accessed 2008-08-10]

The origin of the term fitness has been borrowed biology[36] [1915, 1624] by the evolutionary algorithms community. When the first applications of genetic algorithms were developed, the focus was mainly on single-objective optimization. Back then, they called this single function fitness function and thus, set objective value ≡ fitness value. This point of view is obsolete in principle, yet you will find many contemporary publications that use this notion. This is partly due the fact that in simple problems with only one objective function, the old approach of using the objective values directly as fitness, i. e., $v(x) = f(x)\ \forall x \in \mathbb{X}$, can sometimes actually be applied. In multi-objective optimization processes, this is not possible and fitness assignment processes like those which we are going to elaborate on in Section 2.3 on page 111 are applied instead.

In the context of this book, fitness is subject to minimization, i. e., elements with smaller fitness are "better" than those with higher fitness. Although this definition differs from the biological perception of fitness, it complies with the idea that optimization algorithms are to find the minima of mathematical functions (if nothing else has been stated).

Futher Definitions

In order to ease the discussions of different global optimization algorithms, we furthermore define the data structure *individual*. Especially evolutionary algorithms, but also many other techniques, work on sets of such individuals. Their fitness assignment processes determine fitness values for the individuals relative to all elements of these *populations*.

Definition 1.36 (Individual). An individual p is a tuple $(p.g, p.x)$ of an element $p.g$ in the search space $\mathbb{G}$ and the corresponding element $p.x = \text{gpm}p.g$ in the problem space $\mathbb{X}$.

Besides this basic individual structure, many practical realizations of optimization algorithms use such a data structure to store additional information like the objective and fitness values. Then, we will consider individuals as tuples in $\mathbb{G} \times \mathbb{X} \times Z$, where Z is the space of the additional information stored – $Z = \mathbb{Y} \times \mathbb{V}$, for instance. In the algorithm definitions later in this book, we will often access the phenotypes $p.x$ without explicitly using the genotype-phenotype mapping, since the relation of $p.x$ and $p.g$ complies to Definition 1.36.

Definition 1.37 (Population). A population *Pop* is a list of individuals used during an optimization process.

$$Pop \subseteq \mathbb{G} \times \mathbb{X} : \ \forall p = (p.g, p.x) \in Pop \Rightarrow p.x = \text{gpm}(p.g) \tag{1.33}$$

As already mentioned, the fitness $v(x)$ of an element x in the problem space $\mathbb{X}$ often not solely depends on the element itself. Normally, it is rather a relative measure putting the features of x into the context of a set of solution candidates x. We denote this by writing $v(x, X)$. It is also possible that the fitness involves the whole individual data, including the genotypic and phenotypic structures. We can denote this by writing $v(p, Pop)$.

1.3.2 Fitness Landscapes and Global Optimization

A very powerful metaphor in global optimization is the fitness landscape[37]. Like many other abstractions in optimization, fitness landscapes have been developed and extensively been researched by evolutionary biologists [2261, 1099, 775, 502]. Basically, they are visualizations of the relationship between the genotypes or phenotypes in a given population and their corresponding reproduction probability. The idea of such visualizations goes back to Wright [2261], who used level contours diagrams in order to outline the effects of selection,

[36] http://en.wikipedia.org/wiki/Fitness_%28biology%29 [accessed 2008-02-22]
[37] http://en.wikipedia.org/wiki/Fitness_landscape [accessed 2007-07-03]

mutation, and crossover on the capabilities of populations to escape local optimal configurations. Similar abstractions arise in many other areas [1954], like in physics of disordered systems like spin-glasses [208, 1402], for instance.

In Chapter 2, we will discuss evolutionary algorithms, which are optimization methods inspired by natural evolution. The evolutionary algorithm research community has widely adopted the fitness landscapes as relation between individuals and their objective values [1431, 623]. Langdon and Poli [1242][38] explain that fitness landscapes can be imagined as a view on a countryside from far above. The height of each point is then analogous to its objective value. An optimizer can then be considered as a short-sighted hiker who tries to find the lowest valley or the highest hilltop. Starting from a random point on the map, she wants to reach this goal by walking the minimum distance.

As already mentioned, evolutionary algorithms were first developed as single-objective optimization methods. Then, the objective values were directly used as fitness and the "reproduction probability", i. e., the chance of a solution candidate for being subject of further investigation, was proportional to them. In multi-objective optimization applications with more sophisticated fitness assignment and selection processes, this simple approach does not reflect the biological metaphor correctly anymore.

In the context of this book we will book, we therefore deviate from this view. Since it would possibly be confusing for the reader if we used a different definition for fitness landscapes than the rest of the world, we introduce the new term *problem landscape* and keep using the term *fitness landscape* in the traditional manner. In Figure 1.19 on page 57, you can find some examples for fitness landscapes.

Definition 1.38 (Problem Landscape).
The problem landscape $\Phi : \mathbb{X} \times \mathbb{N} \mapsto [0,1] \subset \mathbb{R}^+$ maps all the points x in a problem space $\mathbb{X}$ to the cumulative probability of reaching them until (inclusively) the τ^{th} evaluation of a solution candidate. The problem landscape thus depends on the optimization problem and on the algorithm applied in order to solve the problem.

$$\Phi(x,\tau) = P\big(x \text{ has been visited until the } \tau^{th} \text{ individual evaluation}\big) \ \forall x \in \mathbb{X}, \tau \in \mathbb{N} \quad (1.34)$$

This definition of problem landscape is very similar to the performance measure definition used by Wolpert and Macready [2244, 2245] in their No Free Lunch Theorem which will be discussed in Section 1.4.10 on page 76. In our understanding, problem landscapes are not only closer to the original meaning of fitness landscapes in biology, they also have another advantage. According to this definition, all entities involved in an optimization process directly influence the problem landscape. The choice of the search operations in the search space $\mathbb{G}$, the way the initial elements are picked, the genotype-phenotype mapping, the objective functions, the fitness assignment process, and the way individuals are selected for further exploration all have impact on Φ. We can furthermore make the following assumptions about $\Phi x \tau$, since it is basically a some form of cumulative distribution function (see Definition 28.18 on page 470).

$$\Phi(x,\tau_1) \geq \Phi(x,\tau_2) \ \forall \tau_1 < \tau_2 \wedge x \in \mathbb{X}, \tau_1, \tau_2 \in \mathbb{N} \quad (1.35)$$

$$0 \leq \Phi(x,\tau) \leq 1 \ \forall x \in \mathbb{X}, \tau \in \mathbb{N} \quad (1.36)$$

Referring back to Definition 1.34, we can now also define what optimization algorithms are.

Definition 1.39 (Optimization Algorithm). An optimization algorithm is a transformation $(\mathbb{X}, F, \mathbb{G}, Op, \text{gpm}) \mapsto \Phi$ of an optimization problem $(\mathbb{X}, F, \mathbb{G}, Op, \text{gpm})$ to a problem landscape Φ that will find at least one *local* optimum $x_l^\star$ for each optimization problem

[38] This part of [1242] is also online available at `http://www.cs.ucl.ac.uk/staff/W.Langdon/FOGP/intro_pic/landscape.html` [accessed 2008-02-15].

$(\mathbb{X}, F, \mathbb{G}, Op, \text{gpm})$ with a weakly complete set of search operations Op and a surjective genotype-phenotype mapping gpm if granted infinite processing time and if such an optimum exists (see Equation 1.37).

$$\exists x_l^\star \in \mathbb{X} : \lim_{\tau \to \infty} \Phi(x_l^\star, \tau) = 1 \tag{1.37}$$

An optimization algorithm is characterized by

1. the way it assigns fitness to the individuals,
2. the ways it selects them for further investigation,
3. the way it applies the search operations, and
4. the way it builds and treats its state information.

The first condition in Definition 1.40, the completeness of Op, is mandatory because the search space $\mathbb{G}$ cannot be explored fully otherwise. If the genotype-phenotype mapping gpm is not surjective, there exist points in the problem space $\mathbb{X}$ which can never be evaluated. Only if both conditions hold, it is guaranteed that an optimization algorithm can find at least one local optimum.

The best optimization algorithm for a given problem $(\mathbb{X}, F, \mathbb{G}, Op, \text{gpm})$ is the one with the highest values of $\Phi(\mathbf{x}^\star, \tau)$ for the optimal elements $\mathbf{x}^\star$ in the problem space and for the lowest values of τ. It may be interesting that this train of thought indicates that finding the best optimization algorithm for a given optimization problem is, itself, a multi-objective optimization problem.

Definition 1.40 (Global Optimization Algorithm). Global optimization algorithms are optimization algorithms that employs measures that prevent convergence to local optima and increase the probability of finding a global optimum.

For a perfect global optimization algorithm (given an optimization problem with weakly complete search operations and a surjective genotype-phenotype mapping), Equation 1.38 would hold. In reality, it can be considered questionable whether such an algorithm can actually be built.

$$\forall x_1, x_2 \in \mathbb{X} : x_1 \succ x_2 \Rightarrow \lim_{\tau \to \infty} \Phi(x_1, \tau) > \lim_{\tau \to \infty} \Phi(x_2, \tau) \tag{1.38}$$

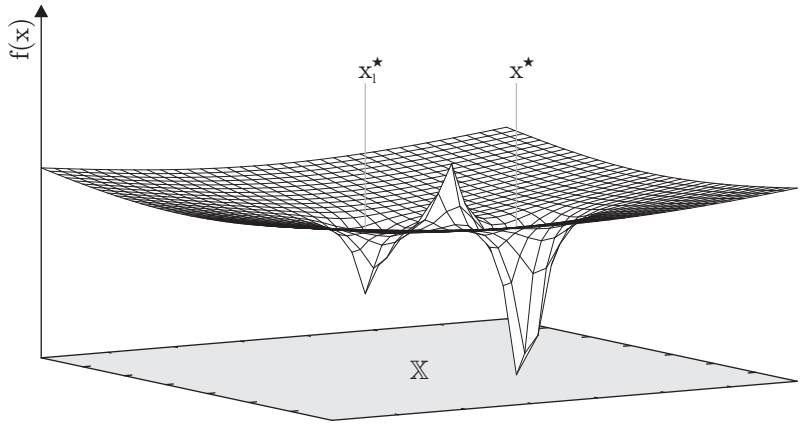


Figure 1.16: An example optimization problem.

Let us now give a simple example for problem landscapes and how they are influenced by the optimization algorithm applied to them. Figure 1.16 illustrates one objective function,

defined over a finite subset $\mathbb{X}$ of the two-dimensional real plane, which we are going to optimize. We use the problem space $\mathbb{X}$ also as search space $\mathbb{G}$, so we can do not need a genotype-phenotype mapping. For optimization, we will use a very simple hill climbing algorithm[39], which initially randomly creates one solution candidate uniformly distributed in $\mathbb{X}$. In each iteration, it creates a new solution candidate from the known one using an unary search operation. The old and the new candidate are compared, and the better one is kept. Hence, we do not need to differentiate between fitness and objective values. In the example, *better* means *has lower fitness*. In Figure 1.16, we can spot one local optimum $x_l^\star$ and one global optimum $\mathbf{x}^\star$. Between them, there is a hill, an area of very bad fitness. The rest of the problem space exhibits a small gradient into the direction its center. The optimization algorithm will likely follow this gradient and sooner or later discover $x_l^\star$ or $\mathbf{x}^\star$. The chances of $x_l^\star$ are higher, since it is closer to the center of $\mathbb{X}$.

With this setting, we have recorded the traces of two experiments with 1.3 million runs of the optimizer (8000 iterations each). From these records, we can approximate the problem landscapes very good.

In the first experiment, depicted in Figure 1.17, we used a search operation $\text{searchOp}_1 : \mathbb{X} \mapsto \mathbb{X}$ which created a new solution candidate normally distributed around the old one. In all experiments, we had divided $\mathbb{X}$ in a regular lattice. $\text{searchOp}_2 : \mathbb{X} \mapsto \mathbb{X}$, used in the second experiment, the new solution candidates are direct neighbors of the old ones in this lattice. The problem landscape Φ produced by this operator is shown in Figure 1.18. Both operators are complete, since each point in the search space can be reached from each other point by applying them.

$$\text{searchOp}_1(x) \equiv (x_1 + \text{random}_n(), x_2 + \text{random}_n()) \tag{1.39}$$

$$\text{searchOp}_1(x) \equiv (x_1 + \text{random}_u(-1, 1)\,, x_2 + \text{random}_u(-1, 1)) \tag{1.40}$$

In both experiments, the first probabilities of the elements of the search space of being discovered are very low, near to zero in the first few iterations. To put it precise, since our problem space is a 36×36 lattice, this probability is $1/36^2$ in the first iteration. Starting with the tenth or so iteration, small peaks begin to form around the places where the optima are located. These peaks grow

Well, as already mentioned, this idea of problem landscapes and optimization reflects solely the author's views. Notice also that it is not always possible to define problem landscapes for problem spaces which are *uncountable infinitely* large. Since the local optimum $x_l^\star$ at the center of the large basin and the gradient points straighter into its direction, it has a higher probability of being found than the global optimum $\mathbf{x}^\star$. The difference between the two search operators tested becomes obvious starting with approximately the 2000^{th} iteration. In the hill climber with the operator utilizing the normal distribution, the Φ value of the global optimum begins to rise farther and farther, finally surpassing the one of the local optimum. Even if the optimizer gets trapped in the local optimum, it will still eventually discover the global optimum and if we had run this experiment longer, the according probability would have converge to 1. The reason for this is that with the normal distribution, all points in the search space have a non-zero probability of being found from all other points in the search space. In other words, all elements of the search space are adjacent.

The operator based on the uniform distribution is only able to create points in the direct neighborhood of the known points. Hence, if an optimizer gets trapped in the local optimum, it can never escape. If it arrives at the global optimum, it will never discover the local one. In Fig. 1.18.l, we can see that $\Phi(x_l^\star, 8000) \approx 0.7$ and $\Phi(\mathbf{x}^\star, 8000) \approx 0.3$. One of the two points will be the result of the optimization process.

From the example we can draw four conclusions:

1. Optimization algorithms discover good elements with higher probability than elements with bad characteristics. Well, this is what they should do.

[39] Hill climbing algorithms are discussed thoroughly in Chapter 10.

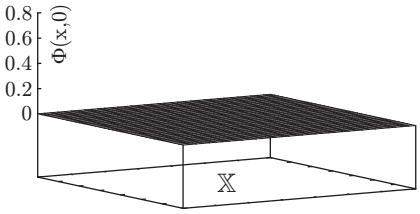


Fig. 1.17.a: $\Phi(x, 1)$

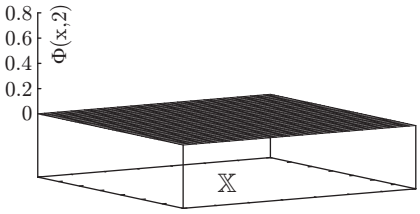


Fig. 1.17.b: $\Phi(x, 2)$

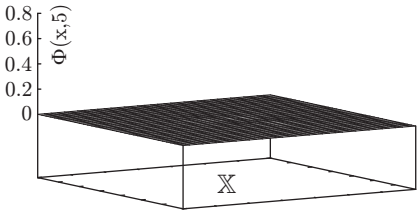


Fig. 1.17.c: $\Phi(x, 5)$

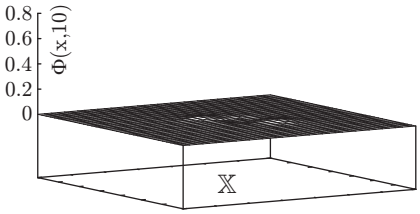


Fig. 1.17.d: $\Phi(x, 10)$

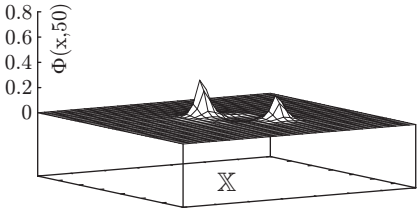


Fig. 1.17.e: $\Phi(x, 50)$

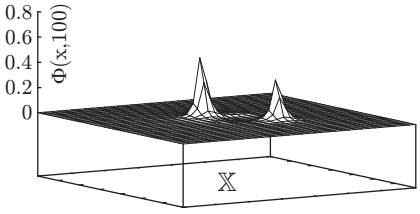


Fig. 1.17.f: $\Phi(x, 100)$

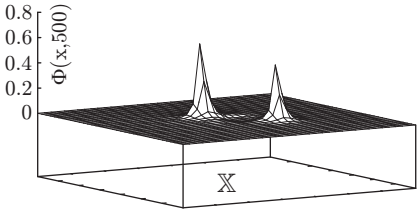


Fig. 1.17.g: $\Phi(x, 500)$

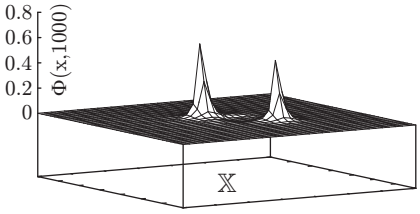


Fig. 1.17.h: $\Phi(x, 1000)$

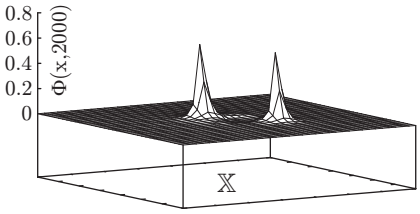


Fig. 1.17.i: $\Phi(x, 2000)$

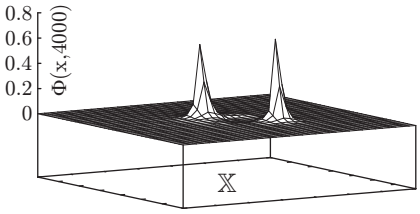


Fig. 1.17.j: $\Phi(x, 4000)$

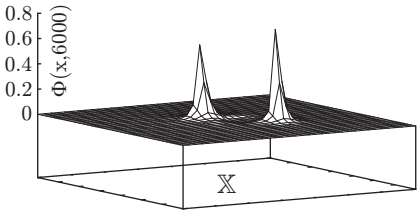


Fig. 1.17.k: $\Phi(x, 6000)$

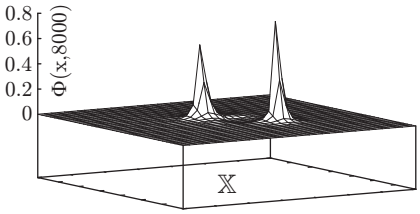


Fig. 1.17.l: $\Phi(x, 8000)$

Figure 1.17: The problem landscape of the example problem derived with searchOp_1.

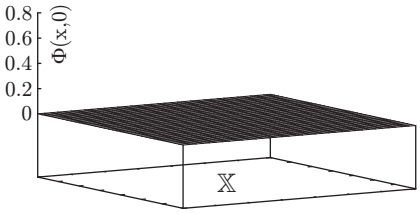


Fig. 1.18.a: $\Phi(x, 1)$

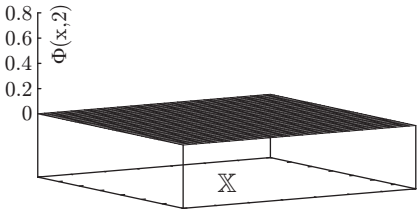


Fig. 1.18.b: $\Phi(x, 2)$

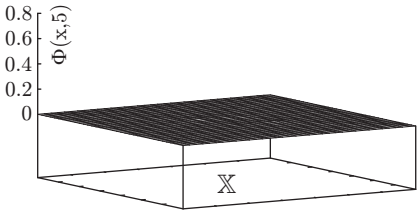


Fig. 1.18.c: $\Phi(x, 5)$

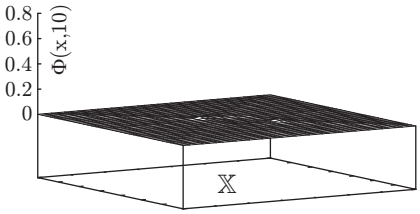


Fig. 1.18.d: $\Phi(x, 10)$

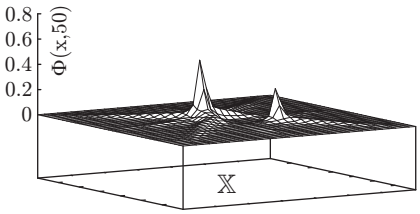


Fig. 1.18.e: $\Phi(x, 50)$

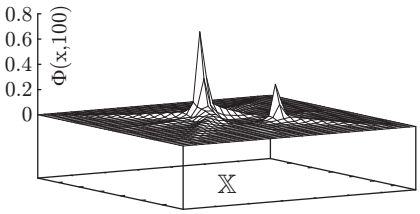


Fig. 1.18.f: $\Phi(x, 100)$

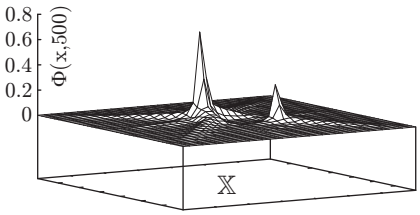


Fig. 1.18.g: $\Phi(x, 500)$

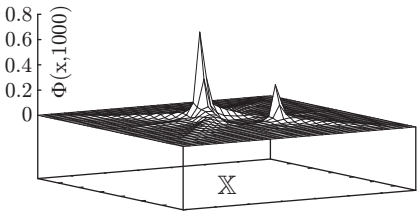


Fig. 1.18.h: $\Phi(x, 1000)$

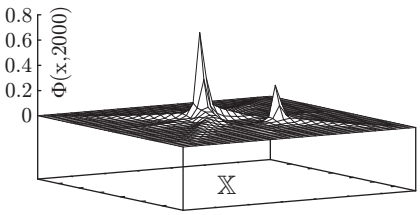


Fig. 1.18.i: $\Phi(x, 2000)$

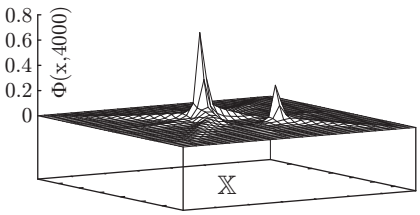


Fig. 1.18.j: $\Phi(x, 4000)$

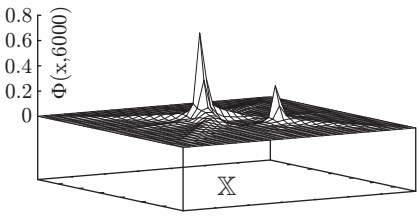


Fig. 1.18.k: $\Phi(x, 6000)$

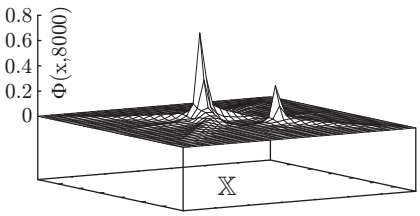


Fig. 1.18.l: $\Phi(x, 8000)$

Figure 1.18: The problem landscape of the example problem derived with searchOp_2.

2. The success of optimization depends very much on the way the search is conducted.
3. It also depends on the time (or the number of iterations) the optimizer allowed to use.
4. Hill climbing algorithms are no global optimization algorithms since they have no means of preventing getting stuck at local optima.

1.3.3 Gradient Descend

Definition 1.41 (Gradient). A gradient[40] of a scalar field $f : \mathbb{R}^n \mapsto \mathbb{R}$ is a vector field which points into the direction of the greatest increase of the scalar field. It is denoted by ∇f or $\text{grad}(f)$.

Optimization algorithms depend on some form of gradient in objective or fitness space in order to find good individuals. In most cases, the problem space $\mathbb{X}$ is not a vector space over the real numbers $\mathbb{R}$, so we cannot directly differentiate the objective functions with Nabla operator[41] ∇F. Generally, samples of the search space are used to approximate the gradient. If we compare to elements x_1 and x_2 of problem space and find $x_1 \succ x_2$, we can assume that there is some sort of gradient facing downwards from x_2 to x_1. When descending this gradient, we can hope to find an x_3 with $x_3 \succ x_1$ and finally the global minimum.

1.3.4 Other General Features

There are some further common semantics and operations that are shared by most optimization algorithms. Many of them, for instance, start out by randomly creating some initial individuals which are then refined iteratively. Optimization processes which are not allowed to run infinitely have to find out when to terminate. In this section we define and discuss general abstractions for such commonalities.

Iterations

Global optimization algorithms often iteratively evaluate solution candidates in order to approach the optima. We distinguish between evaluations τ and iterations t.

Definition 1.42 (Evaluation). The value $\tau \in \mathbb{N}_0$ denotes the number of solution candidates for which the set of objective functions F has been evaluated.

Definition 1.43 (Iteration). An iteration[42] refers to one round in a loop of an algorithm. It is one repetition of a specific sequence of instruction inside an algorithm.

Algorithms are referred to as *iterative* if most of their work is done by cyclic repetition of one main loop. In the context of this book, an iterative optimization algorithm starts with the first step $t = 0$. The value $t \in \mathbb{N}_0$ is the index of the iteration currently performed by the algorithm and $t + 1$ refers to the following step. One example for iterative algorithm is Algorithm 1.1. In some optimization algorithms like genetic algorithms, for instance, iterations are referred to as *generations*.

There often exists a well-defined relation between the number of performed solution candidate evaluations τ and the index of the current iteration t in an optimization process: Many global optimization algorithms generate and evaluate a certain number of individuals per generation.

[40] http://en.wikipedia.org/wiki/Gradient [accessed 2007-11-06]
[41] http://en.wikipedia.org/wiki/Del [accessed 2008-02-15]
[42] http://en.wikipedia.org/wiki/Iteration [accessed 2007-07-03]

Termination Criterion

The termination criterion terminationCriterion() is a function with access to all the information accumulated by an optimization process, including the number of performed steps t, the objective values of the best individuals, and the time elapsed since the start of the process. With terminationCriterion(), the optimizers determine when they have to halt.

Definition 1.44 (Termination Criterion). When the termination criterion function terminationCriterion() $\in \{\texttt{true}, \texttt{false}\}$ evaluates to `true`, the optimization process will stop and return its results.

Some possible criteria that can be used to decide whether an optimizer should terminate or not are [1975, 1634, 2325, 2326]:

1. The user may grant the optimization algorithm a maximum computation time. If this time has been exceeded, the optimizer should stop. Here we should note that the time needed for single individuals may vary, and so will the times needed for iterations. Hence, this time threshold can sometimes not be abided exactly.
2. Instead of specifying a time limit, a total number of iterations $\hat{t}$ or individual evaluations $\hat{\tau}$ may be specified. Such criteria are most interesting for the researcher, since she often wants to know whether a qualitatively interesting solution can be found for a given problem using at most a predefined number of samples from the problem space.
3. An optimization process may be stopped when no improvement in the solution quality could be detected for a specified number of iterations. Then, the process most probably has converged to a (hopefully good) solution and will most likely not be able to make further progress.
4. If we optimize something like a decision maker or classifier based on a sample data set, we will normally divide this data into a training and a test set. The training set is used to guide the optimization process whereas the test set is used to verify its results. We can compare the performance of our solution when fed with the training set to its properties if fed with the test set. This comparison helps us detect when most probably no further generalization can be achieved by the optimizer and we should terminate the process.
5. Obviously, we can terminate an optimization process if it has already yielded a sufficiently good solution.

In practical applications, we can apply any combination of the criteria above in order to determine when to halt. How the termination criterion is tested in an iterative algorithm is illustrated in Algorithm 1.1.

Algorithm 1.1: Example Iterative Algorithm

Input: [implicit] terminationCriterion(): the termination criterion
Data: t: the iteration counter

```
1 begin
2     t ⟵ 0
      // initialize the data of the algorithm
3     while ¬terminationCriterion() do
          // perform one iteration - here happens the magic
4         t ⟵ t + 1
5 end
```

Minimization

Many optimization algorithms have been developed for single-objective optimization in their original form. Such algorithms may be used for both, minimization or maximization. Without loss of generality we will present them as minimization processes since this is the most commonly used notation. An algorithm that maximizes the function f may be transformed to a minimization using $-f$ instead.

Note that using the prevalence comparisons as introduced in Section 1.2.4 on page 38, multi-objective optimization processes can be transformed into single-objective minimization processes. Therefore $x_1 \succ x_2 \Leftrightarrow \text{cmp}_F(x_1, x_2) < 0$.

Modeling and Simulating

While there are a lot of problems where the objective functions are mathematical expressions that can directly be computed, there exist problem classes far away from such simple function optimization that require complex models and simulations.

Definition 1.45 (Model). A model[43] is an abstraction or approximation of a system that allows us to reason and to deduce properties of the system.

Models are often simplifications or idealization of real-world issues. They are defined by leaving away facts that probably have only minor impact on the conclusions drawn from them. In the area of global optimization, we often need two types of abstractions:

1. The models of the potential solutions shape the problem space $\mathbb{X}$. Examples are
   a) programs in Genetic Programming, for example for the Artificial Ant problem,
   b) construction plans of a skyscraper,
   c) distributed algorithms represented as programs for Genetic Programming,
   d) construction plans of a turbine,
   e) circuit diagrams for logical circuits, and so on.
2. Models of the environment in which we can test and explore the properties of the potential solutions, like
   a) a map on which the Artificial Ant will move which is driven by the evolved program,
   b) an abstraction from the environment in which the skyscraper will be built, with wind blowing from several directions,
   c) a model of the network in which the evolved distributed algorithms can run,
   d) a physical model of air which blows through the turbine,
   e) the model of an energy source the other pins which will be attached to the circuit together with the possible voltages on these pins.

Models themselves are rather static structures of descriptions and formulas. Deriving concrete results (objective values) from them is often complicated. It often makes more sense to bring the construction plan of a skyscraper to life in a simulation. Then we can test the influence of various wind strengths and directions on building structure and approximate the properties which define the objective values.

Definition 1.46 (Simulation). A simulation[44] is the computational realization of a model. Whereas a model describes abstract connections between the properties of a system, a simulation realizes these connections.

Simulations are executable, live representations of models that can be as meaningful as real experiments. They allow us to reason if a model makes sense or not and how certain objects behave in the context of a model.

[43] http://en.wikipedia.org/wiki/Model_%28abstract%29 [accessed 2007-07-03]
[44] http://en.wikipedia.org/wiki/Simulation [accessed 2007-07-03]

1.4 Problems in Optimization

1.4.1 Introduction

The classification of optimization algorithms in Section 1.1.1 and the table of contents of this book enumerate a wide variety of optimization algorithms. Yet, the approaches introduced here resemble only a small fraction of the actual number of available methods. It is a justified question to ask why there are so many different approaches, why is this variety needed? One possible answer is simply because there are so many different kinds of optimization tasks. Each of them puts different obstacles into the way of the optimizers and comes with own, characteristic difficulties.

In this chapter we want to discuss the most important of these complications, the major problems that may be encountered during optimization. Some of subjects in the following text concern global optimization in general (multi-modality and overfitting, for instance), others apply especially to nature-inspired approaches like genetic algorithms (epistasis and neutrality, for example). Neglecting even a single one them during the design or process of optimization can render the whole efforts invested useless, even if highly efficient optimization techniques are applied. By giving clear definitions and comprehensive introductions to these topics, we want to raise the awareness of scientists and practitioners in the industry and hope to help them to use optimization algorithms more efficiently.

In Figure 1.19, we have sketched a set of different types of fitness landscapes (see Section 1.3.2) which we are going to discuss. The objective values in the figure are subject to minimization and the small bubbles represent solution candidates under investigation. An arrow from one bubble to another means that the second individual is found by applying one search operation to the first one.

The Term "Difficult"

Before we go more into detail about what makes these landscapes *difficult*, we should establish the term in the context of optimization. The degree of difficulty of solving a certain problem with a dedicated algorithm is closely related to its *computational complexity*[45], i. e., the amount of resources such as time and memory required to do so. The computational complexity depends on the number of input elements needed for applying the algorithm. This dependency is often expressed in form of approximate boundaries with the Big-**O**-family notations introduced by Bachmann [96] and made popular by Landau [1236]. Problems can further be divided into *complexity classes*. One of the most difficult complexity classes owning to its resource requirements is $\mathcal{NP}$, the set of all decision problems which are solvable in polynomial time by non-deterministic Turing machines [773]. Although many attempts have been made, no algorithm has been found which is able to solve an $\mathcal{NP}$-complete [773] problem in polynomial time on a deterministic computer. One approach to obtaining near-optimal solutions for problems in $\mathcal{NP}$ in reasonable time is to apply metaheuristic, randomized optimization procedures.

As already stated, optimization algorithms are guided by objective functions. A function is *difficult* from a mathematical perspective in this context if it is not continuous, not differentiable, or if it has multiple maxima and minima. This understanding of difficulty comes very close to the intuitive sketches in Figure 1.19.

In many real world applications of metaheuristic optimization, the characteristics of the objective functions are not known in advance. The problems are usually $\mathcal{NP}$ or have

[45] see Section 30.1.3 on page 550

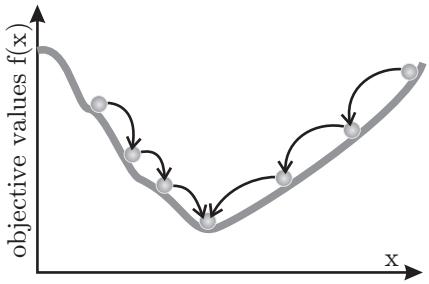


Fig. 1.19.a: Best Case

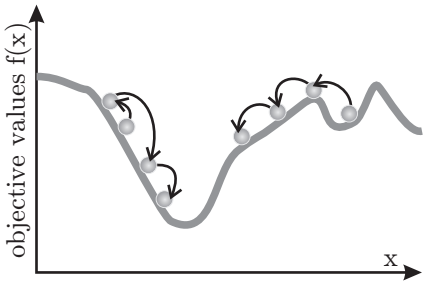


Fig. 1.19.b: Low Variation

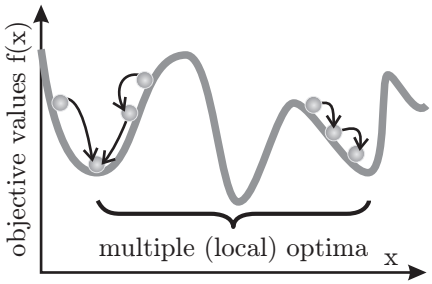


Fig. 1.19.c: Multimodal

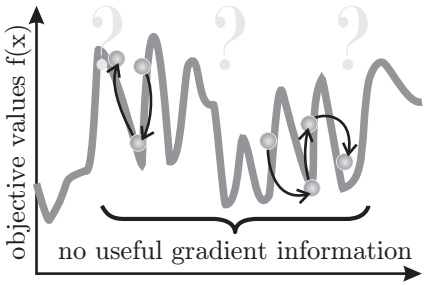


Fig. 1.19.d: Rugged

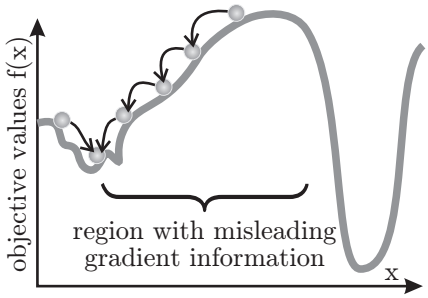


Fig. 1.19.e: Deceptive

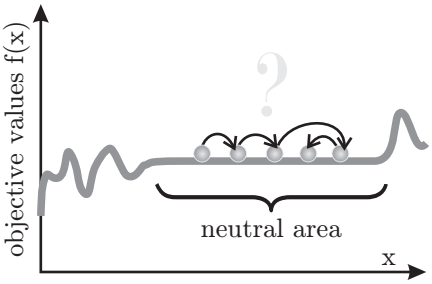


Fig. 1.19.f: Neutral

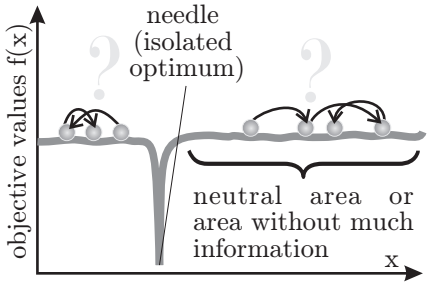


Fig. 1.19.g: Needle-In-A-Haystack

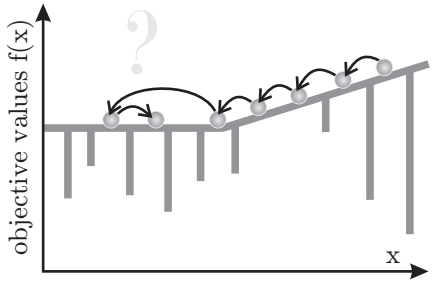


Fig. 1.19.h: Nightmare

Figure 1.19: Different possible properties of fitness landscapes (minimization).

unknown complexity. It is therefore only rarely possible to derive boundaries for the performance or the runtime of optimizers in advance, let alone exact estimates with mathematical precision.

Most often, experience, rules of thumb, and empirical results based on models obtained from related research areas such as biology are the only guides available. In this chapter, we discuss many such models and rules, providing a better understanding of when the application of a metaheuristic is feasible and when not, as well as with indicators on how to avoid defining problems in a way that makes them *difficult*.

1.4.2 Premature Convergence

Introduction

An optimization algorithm has *converged* if it cannot reach new solution candidates anymore or if it keeps on producing solution candidates from a "small"[46] subset of the problem space. Meta-heuristc global optimization algorithms will usually converge at some point in time. In nature, a similar phenomenon can be observed according to [1196]: The *niche preemption principle* states that a niche in a natural environment tends to become dominated by a single species [1347]. One of the problems in global optimization (and basically, also in nature) is that it is often not possible to determine whether the best solution currently known is a situated on local or a global optimum and thus, if convergence is acceptable. In other words, it is usually not clear whether the optimization process can be stopped, whether it should concentrate on refining the current optimum, or whether it should examine other parts of the search space instead. This can, of course, only become cumbersome if there are multiple (local) optima, i. e., the problem is *multimodal* as depicted in Fig. 1.19.c.

A mathematical function is multimodal if it has multiple maxima or minima [1863, 2327, 512]. A set of objective functions (or a vector function) F is multimodal if it has multiple (local or global) optima – depending on the definition of "optimum" in the context of the corresponding optimization problem.

The Problem

An optimization process has *prematurely converged* to a local optimum if it is no longer able to explore other parts of the search space than the area currently being examined *and* there exists another region that contains a superior solution [2075, 1824]. Figure 1.20 illustrates examples for premature convergence.

The existence of multiple global optima itself is not problematic and the discovery of only a subset of them can still be considered as successful in many cases. The occurrence of numerous local optima, however, is more complicated.

Domino Convergence

The phenomenon of *domino convergence* has been brought to attention by Rudnick [1773] who studied it in the context of his BinInt problem [1773, 2036] which is discussed in Section 21.2.5. In principle, domino convergence occurs when the solution candidates have features which contribute to significantly different degrees to the total fitness. If these features are encoded in separate genes (or building blocks) in the genotypes, they are likely to be treated with different priorities, at least in randomized or heuristic optimization methods.

Building blocks with a very strong positive influence on the objective values, for instance, will quickly be adopted by the optimization process (i. e., "converge"). During this time, the alleles of genes with a smaller contribution are ignored. They do not come into play until

[46] according to a suitable metric like numbers of modifications or mutations which need to be applied to a given solution in order to leave this subset

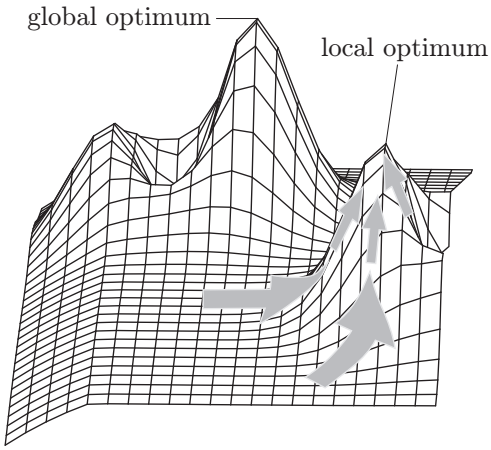


Fig. 1.20.a: Example 1: Maximization

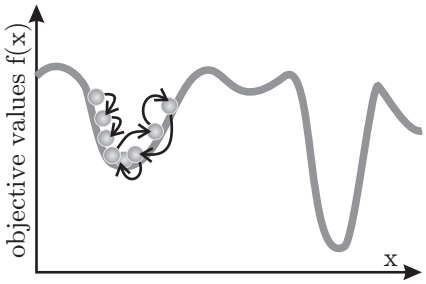


Fig. 1.20.b: Example 2: Minimization

Figure 1.20: Premature convergence in the objective space.

the optimal alleles of the more "important" blocks have been accumulated. Rudnick [1773] called this sequential convergence phenomenon *domino convergence* due to its resemblance to a row of falling domino stones [2036].

Let us consider the application of a genetic algorithm in such a scenario. Mutation operators from time to time destroy building blocks with strong positive influence which are then reconstructed by the search. If this happens with a high enough frequency, the optimization process will never get to optimize the lower salient blocks because repairing and rediscovering those with higher importance takes precedence. Thus, the mutation rate of the EA limits the probability of finding the global optima in such a situation.

In the worst case, the contributions of the less salient genes may almost look like noise and they are not optimized at all. Such a situation is also an instance of premature convergence, since the global optimum which would involve optimal configurations of all building blocks will not be discovered. In this situation, restarting the optimization process will not help because it will always turn out the same way. Example problems which are often likely to exhibit domino convergence are the Royal Road and the aforementioned BinInt problem, which you can find discussed in Section 21.2.4 and Section 21.2.5, respectively.

One Cause: Loss of Diversity

In biology, diversity is *the variety and abundance of organisms at a given place and time* [1598, 1348]. Much of the beauty and efficiency of natural ecosystems is based on a dazzling array of species interacting in manifold ways. Diversification is also a good strategy utilized by investors in the economy in order to increase their wealth.

In population-based global optimization algorithms, maintaining a set of diverse solution candidates is very important as well. Losing diversity means approaching a state where all the solution candidates under investigation are similar to each other. Another term for this state is convergence. Discussions about how diversity can be measured have been provided by Routledge [1771], Cousins [459], Magurran [1348], Morrison and De Jong [1462], Paenke et al. [1598], and Burke et al. [309, 311].

Preserving diversity is directly linked with maintaining a good balance between exploitation and exploration [1598] and has been studied by researchers from many domains, such as

1. Genetic Algorithms [1558, 1750, 1751],
2. Evolutionary Algorithms [253, 254, 1262, 1471, 1943, 1892],

3. Genetic Programming [510, 871, 872, 310, 311, 273],
4. Tabu Search [812, 816], and
5. Particle Swarm Optimization [2226].

Exploration vs. Exploitation

The operations which create new solutions from existing ones have a very large impact on the speed of convergence and the diversity of the populations [637, 1910]. The step size in Evolution Strategy is a good example of this issue: setting it properly is very important and leads over to the "exploration versus exploitation" problem [940] which can be observed in other areas of global optimization as well.[47]

In the context of optimization, *exploration* means finding new points in areas of the search space which have not been investigated before. Since computers have only limited memory, already evaluated solution candidates usually have to be discarded in order to accommodate the new ones. Exploration is a metaphor for the procedure which allows search operations to find novel and maybe better solution structures. Such operators (like mutation in evolutionary algorithms) have a high chance of creating inferior solutions by destroying good building blocks but also a small chance of finding totally new, superior traits (which, however, is not guaranteed at all).

Exploitation, on the other hand, is the process of improving and combining the traits of the currently known solutions, as done by the crossover operator in evolutionary algorithms, for instance. Exploitation operations often incorporate small changes into already tested individuals leading to new, very similar solution candidates or try to merge building blocks of different, promising individuals. They usually have the disadvantage that other, possibly better, solutions located in distant areas of the problem space will not be discovered.

Almost all components of optimization strategies can either be used for increasing exploitation or in favor of exploration. Unary search operations that improve an existing solution in small steps can often be built, hence being exploitation operators. They can also be implemented in a way that introduces much randomness into the individuals, effectively making them exploration operators. Selection operations[48] in Evolutionary Computation choose a set of the most promising solution candidates which will be investigated in the next iteration of the optimizers. They can either return a small group of best individuals (exploitation) or a wide range of existing solution candidates (exploration).

Optimization algorithms that favor exploitation over exploration have higher convergence speed but run the risk of not finding the optimal solution and may get stuck at a local optimum. Then again, algorithms which perform excessive exploration may never improve their solution candidates well enough to find the global optimum or it may take them very long to discover it "by accident". A good example for this dilemma is the Simulated Annealing algorithm discussed in Chapter 12 on page 263. It is often modified to a form called *simulated quenching* which focuses on exploitation but loses the guaranteed convergence to the optimum. Generally, optimization algorithms should employ at least one search operation of explorative character and at least one which is able to exploit good solutions further. There exists a vast body of research on the trade-off between exploration and exploitation that optimization algorithms have to face [638, 945, 622, 1494, 49, 538].

Countermeasures

There is no general approach which can prevent premature convergence. The probability that an optimization process gets caught in a local optimum depends on the characteristics of the problem to be solved and the parameter settings and features of the optimization algorithms applied [2051, 1775].

[47] More or less synonymously to exploitation and exploration, the terms *intensifications* and *diversification* have been introduced by Glover [812, 816] in the context of Tabu Search.

[48] Selection will be discussed in Section 2.4 on page 121.

A very crude and yet, sometimes effective measure is restarting the optimization process at randomly chosen points in time. One example for this method is *GRASP*s, *Greedy Randomized Adaptive Search Procedures* [663, 652] (see Section 10.6 on page 256), which continuously restart the process of creating an initial solution and refining it with local search. Still, such approaches are likely to fail in domino convergence situations. Increasing the proportion of exploration operations may also reduce the chance of premature convergence.

In order to extend the duration of the evolution in evolutionary algorithms, many methods have been devised for steering the search away from areas which have already been frequently sampled. This can be achieved by integrating density metrics into the fitness assignment process. The most popular of such approaches are sharing and niching (see Section 2.3.4). The Strength Pareto Algorithms, which are widely accepted to be highly efficient, use another idea: they adapt the number of individuals that one solution candidate *dominates* as density measure [2329, 2332]. One very simple method aiming for convergence prevention is introduced in Section 2.4.8. Using low selection pressure furthermore decreases the chance of premature convergence but also decreases the speed with which good solutions are exploited.

Another approach against premature convergence is to introduce the capability of self-adaptation, allowing the optimization algorithm to change its strategies or to modify its parameters depending on its current state. Such behaviors, however, are often implemented not in order to prevent premature convergence but to speed up the optimization process (which may lead to premature convergence to local optima) [1776, 1777, 1778].

1.4.3 Ruggedness and Weak Causality

The Problem: Ruggedness

Optimization algorithms generally depend on some form of gradient in the objective or fitness space. The objective functions should be continuous and exhibit low total variation[49], so the optimizer can descend the gradient easily. If the objective functions are unsteady or fluctuating, i. e., going up and down, it becomes more complicated for the optimization process to find the right directions to proceed to. The more rugged a function gets, the harder it becomes to optimize it. For short, one could say ruggedness is multi-modality plus steep ascends and descends in the fitness landscape. Examples of rugged landscapes are Kauffman's NK fitness landscape (see Section 21.2.1), the p-Spin model discussed in Section 21.2.2, Bergman and Feldman's jagged fitness landscape [182], and the sketch in Fig. 1.19.d on page 57.

One Cause: Weak Causality

During an optimization process, new points in the search space are created by the search operations. Generally we can assume that the genotypes which are the input of the search operations correspond to phenotypes which have previously been selected. Usually, the better or the more promising an individual is, the higher are its chances of being selected for further investigation. Reversing this statement suggests that individuals which are passed to the search operations are likely to have a good fitness. Since the fitness of a solution candidate depends on its properties, it can be assumed that the features of these individuals are not so bad either. It should thus be possible for the optimizer to introduce slight changes to their

[49] http://en.wikipedia.org/wiki/Total_variation [accessed 2008-04-23]

properties in order to find out whether they can be improved any further[50]. Normally, such *exploitive* modifications should also lead to small changes in the objective values and hence, in the fitness of the solution candidate.

Definition 1.47 (Strong Causality). *Strong causality* (locality) means that small changes in the properties of an object also lead to small changes in its behavior [1713, 1714, 1759].

This principle (proposed by Rechenberg [1713, 1714]) should not only hold for the search spaces and operations designed for optimization, but applies to natural genomes as well. The offspring resulting from sexual reproduction of two fish, for instance, has a different genotype than its parents. Yet, it is far more probable that these variations manifest in a unique color pattern of the scales, for example, instead of leading to a totally different creature.

Apart from this straightforward, informal explanation here, causality has been investigated thoroughly in different fields of optimization, such as Evolution Strategy [1713, 597], structure evolution [1303, 1302], Genetic Programming [1758, 1759, 1007, 597], genotype-phenotype mappings [1854], search operators [597], and evolutionary algorithms in general [1955, 1765, 597].

In fitness landscapes with weak (low) causality, small changes in the solution candidates often lead to large changes in the objective values, i. e., ruggedness. It then becomes harder to decide which region of the problem space to explore and the optimizer cannot find reliable gradient information to follow. A small modification of a very bad solution candidate may then lead to a new local optimum and the best solution candidate currently known may be surrounded by points that are inferior to all other tested individuals.

The lower the causality of an optimization problem, the more rugged its fitness landscape is, which leads to a degeneration of the performance of the optimizer [1168]. This does not necessarily mean that it is impossible to find good solutions, but it may take very long to do so.

Fitness Landscape Measures

As measures for the ruggedness of a fitness landscape (or their general difficulty), many different metrics have been proposed. Wedge and Kell [2164] and Altenberg [45] provide nice lists of them in their work[51], which we summarize here:

- Weinberger [2169] introduced the autocorrelation function and the correlation length of random walks.
- The correlation of the search operators was used by Manderick et al. [1354] in conjunction with the autocorrelation.
- Jones and Forrest [1070, 1069] proposed the fitness distance correlation (FDC), the correlation of the fitness of an individual and its distance to the global optimum. This measure has been extended by researchers such as Clergue et al. [416, 2103].
- The probability that search operations create offspring fitter than their parents, as defined by Rechenberg [1713] and Beyer [196] (and called evolvability by Altenberg [42]), will be discussed in Section 1.4.5 on page 65 in depth.
- Simulation dynamics have been researched by Altenberg [42] and Grefenstette [855].
- Another interesting metric is the fitness variance of formae (Radcliffe and Surry [1695]) and schemas (Reeves and Wright [1717]).
- The error threshold method from theoretical biology [625, 1552] has been adopted Ochoa et al. [1557] for evolutionary algorithms. It is the "critical mutation rate beyond which structures obtained by the evolutionary process are destroyed by mutation more frequently than selection can reproduce them" [1557].

[50] We have already mentioned this under the subject of exploitation.

[51] Especially the one of Wedge and Kell [2164] is beautiful and far more detailed than this summary here.

- The negative slope coefficient (NSC) by Vanneschi et al. [2104, 2105] may be considered as an extension of Altenberg's evolvability measure.
- Davidor [489] uses the epistatic variance as a measure of utility of a certain representation in genetic algorithms. We discuss the issue of epistasis in Section 1.4.6.
- The genotype-fitness correlation (GFC) of Wedge and Kell [2164] is a new measure for ruggedness in fitness landscape and has been shown to be a good guide for determining optimal population sizes in Genetic Programming.

Autocorrelation and Correlation Length

As example, let us take a look at the autocorrelation function as well as the correlation length of random walks [2169]. Here we borrow its definition from Verel et al. [2114]:

Definition 1.48 (Autocorrelation Function). Given a random walk $(x_i, x_{i+1}, \dots)$, the autocorrelation function ρ of an objective function f is the autocorrelation function of the time series $(f(x_i), f(x_{i+1}), \dots)$.

$$\rho(k, f) = \frac{E[f(x_i)\, f(x_{i+k})] - E[f(x_i)]\, E[f(x_{i+k})]}{D^2[f(x_i)]} \tag{1.41}$$

where $E[f(x_i)]$ and $D^2[f(x_i)]$ are the expected value and the variance of $f(x_i)$.

The correlation length $\tau = -\frac{1}{\log \rho(1,f)}$ measures how the autocorrelation function decreases and summarizes the ruggedness of the fitness landscape: the larger the correlation length, the lower the total variation of the landscape. From the works of Kinnear, Jr. [1141] and Lipsitch [1293] from 18, however, we also know that correlation measures do not always represent the hardness of a problem landscape full.

Countermeasures

To the knowledge of the author, no viable method which can directly mitigate the effects of rugged fitness landscapes exists. In population-based approaches, using large population sizes and applying methods to increase the diversity can reduce the influence of ruggedness, but only up to a certain degree. Utilizing Lamarckian evolution [522, 2215] or the Baldwin effect [123, 929, 930, 2215], i. e., incorporating a local search into the optimization process, may further help to smoothen out the fitness landscape [864] (see Section 15.2 and Section 15.3, respectively).

Weak causality is often a home-made problem because it results to some extent from the choice of the solution representation and search operations. We pointed out that exploration operations are important for lowering the risk of premature convergence. Exploitation operators are as same as important for refining solutions to a certain degree. In order to apply optimization algorithms in an efficient manner, it is necessary to find representations which allow for iterative modifications with bounded influence on the objective values, i. e., exploitation. In Section 1.5.2, we present some further rules-of-thumb for search space and operation design.

1.4.4 Deceptiveness

Introduction

Especially annoying fitness landscapes show *deceptiveness* (or deceptivity). The gradient of deceptive objective functions leads the optimizer away from the optima, as illustrated in Fig. 1.19.e.

The term deceptiveness is mainly used in the genetic algorithm[52] community in the context of the Schema Theorem. Schemas describe certain areas (hyperplanes) in the search space. If an optimization algorithm has discovered an area with a better average fitness compared to other regions, it will focus on exploring this region based on the assumption that highly fit areas are likely to contain the true optimum. Objective functions where this is not the case are called deceptive [190, 821, 1285]. Examples for deceptiveness are the ND fitness landscapes outlined in Section 21.2.3, trap functions (see Section 21.2.3), and the fully deceptive problems given by Goldberg et al. [825, 541].

The Problem

If the information accumulated by an optimizer actually guides it away from the optimum, search algorithms will perform worse than a random walk or an exhaustive enumeration method. This issue has been known for a long time [2159, 1433, 1434, 2034] and has been subsumed under the No Free Lunch Theorem which wewill discuss in Section 1.4.10.

Countermeasures

Solving deceptive optimization tasks perfectly involves sampling many individuals with very bad features and low fitness. This contradicts the basic ideas of metaheuristics and thus, there are no efficient countermeasures against deceptivity. Using large population sizes, maintaining a very high diversity, and utilizing linkage learning (see Section 1.4.6) are, maybe, the only approaches which can provide at least a small chance of finding good solutions.

1.4.5 Neutrality and Redundancy

The Problem: Neutrality

Definition 1.49 (Neutrality). We consider the outcome of the application of a search operation to an element of the search space as neutral if it yields no change in the objective values [1718, 149].

It is challenging for optimization algorithms if the best solution candidate currently known is situated on a plane of the fitness landscape, i. e., all adjacent solution candidates have the same objective values. As illustrated in Fig. 1.19.f, an optimizer then cannot find any gradient information and thus, no direction in which to proceed in a systematic manner. From its point of view, each search operation will yield identical individuals. Furthermore, optimization algorithms usually maintain a list of the best individuals found, which will then overflow eventually or require pruning.

The degree of neutrality ν is defined as the fraction of neutral results among all possible products of the search operations applied to a specific genotype [149]. We can generalize this measure to areas G in the search space $\mathbb{G}$ by averaging over all their elements. Regions where ν is close to one are considered as *neutral*.

$$\forall g_1 \in \mathbb{G} \Rightarrow \nu(g_1) = \frac{|\{g_2 : P(g_2 = Op(g_1)) > 0 \wedge F(\text{gpm}(g_2)) = F(\text{gpm}(g_1))\}|}{|\{g_2 : P(g_2 = Op(g_1)) > 0\}|} \tag{1.42}$$

$$\forall G \subseteq \mathbb{G} \Rightarrow \nu(G) = \frac{1}{|G|} \sum_{g \in G} \nu(g) \tag{1.43}$$

[52] We are going to discuss genetic algorithms in Chapter 3 on page 141 and the Schema Theorem in Section 3.6 on page 150.

Evolvability

Another metaphor in global optimization borrowed from biological systems is evolvability[53] [500]. Wagner [2132, 2133] points out that this word has two uses in biology: According to Kirschner and Gerhart [1144], a biological system is evolvable if it is able to generate heritable, selectable phenotypic variations. Such properties can then be spread by natural selection and changed during the course of evolution. In its second sense, a system is evolvable if it can acquire new characteristics via genetic change that help the organism(s) to survive and to reproduce. Theories about how the ability of generating adaptive variants has evolved have been proposed by Riedl [1732], Altenberg [43], Wagner and Altenberg [2134], Bonner [247], and Conrad [439], amongst others. The idea of evolvability can be adopted for global optimization as follows:

Definition 1.50 (Evolvability). The evolvability of an optimization process in its current state defines how likely the search operations will lead to solution candidates with new (and eventually, better) objectives values.

The direct *probability of success* [1713, 196], i.e., the chance that search operators produce offspring fitter than their parents, is also sometimes referred to as *evolvability* in the context of evolutionary algorithms [45, 42].

Neutrality: Problematic and Beneficial

The link between evolvability and neutrality has been discussed by many researchers [2300, 2133]. The evolvability of neutral parts of a fitness landscape depends on the optimization algorithm used. It is especially low for hill climbing and similar approaches, since the search operations cannot directly provide improvements or even changes. The optimization process then degenerates to a random walk, as illustrated in Fig. 1.19.f on page 57. The work of Beaudoin et al. [161] on the ND fitness landscapes[54] shows that neutrality may "destroy" useful information such as correlation.

Researchers in molecular evolution, on the other hand, found indications that the majority of mutations in biology have no selective influence [732, 980] and that the transformation from genotypes to phenotypes is a many-to-one mapping. Wagner [2133] states that neutrality in natural genomes is beneficial if it concerns only a subset of the properties peculiar to the offspring of a solution candidate while allowing meaningful modifications of the others. Toussaint and Igel [2050] even go as far as declaring it a necessity for self-adaptation.

The theory of *punctuated equilibria*[55], in biology introduced by Eldredge and Gould [630, 629], states that species experience long periods of evolutionary inactivity which are interrupted by sudden, localized, and rapid phenotypic evolutions [118].[56] It is assumed that the populations explore neutral layers[57] during the time of stasis until, suddenly, a relevant change in a genotype leads to a better adapted phenotype [2098] which then reproduces quickly. Similar phenomena can be observed/are utilized in EAs [426, 1365].

"Uh?", you may think, "How does this fit together?" The key to differentiating between "good" and "bad" neutrality is its degree ν in relation to the number of possible solutions maintained by the optimization algorithms. Smith et al. [1913] have used illustrative examples similar to Figure 1.21 showing that a certain amount of neutral reproductions can foster the progress of optimization. In Fig. 1.21.a, basically the same scenario of premature convergence as in Fig. 1.20.a on page 59 is depicted. The optimizer is drawn to a local optimum from which it cannot escape anymore. Fig. 1.21.b shows that a little shot of neutrality

[53] http://en.wikipedia.org/wiki/Evolvability [accessed 2007-07-03]
[54] See Section 21.2.3 on page 333 for a detailed elaboration on the ND fitness landscape.
[55] http://en.wikipedia.org/wiki/Punctuated_equilibrium [accessed 2008-07-01]
[56] A very similar idea is utilized in the Extremal Optimization method discussed in Chapter 13.
[57] Or neutral networks, as discussed in Section 1.4.5.

could form a bridge to the global optimum. The optimizer now has a chance to escape the smaller peak if it is able to find and follow that bridge, i. e., the evolvability of the system has increased. If this bridge gets wider, as sketched in Fig. 1.21.c, the chance of finding the global optimum increases as well. Of course, if the bridge gets too wide, the optimization process may end up in a scenario like in Fig. 1.19.f on page 57 where it cannot find any direction. Furthermore, in this scenario we expect the neutral bridge to lead to somewhere useful, which is not necessarily the case in reality.

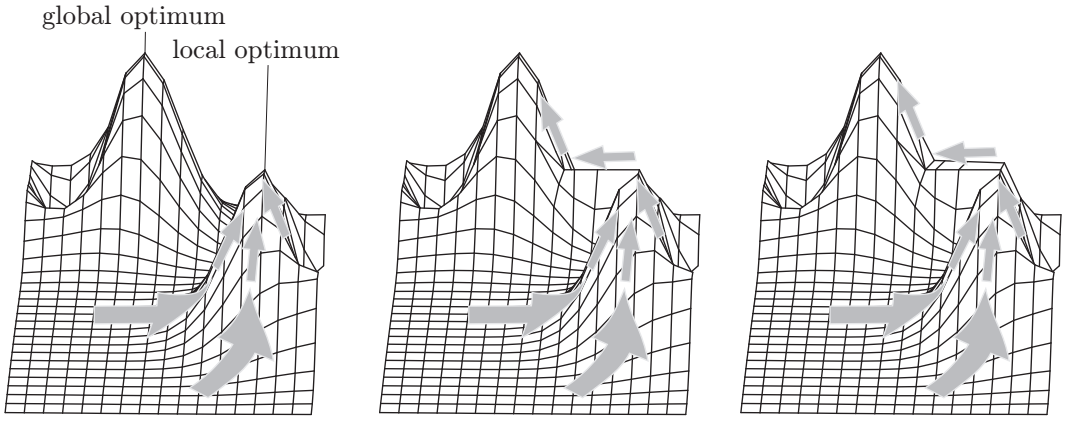


Fig. 1.21.a: Premature Convergence

Fig. 1.21.b: Small Neutral Bridge

Fig. 1.21.c: Wide Neutral Bridge

Figure 1.21: Possible positive influence of neutrality.

Recently, the idea of utilizing the processes of molecular[58] and evolutionary[59] biology as complement to Darwinian evolution for optimization gains interest [144]. Scientists like Hu and Banzhaf [967, 968] have begun to study the application of metrics such as the evolution rate of gene sequences [2281, 2257] to evolutionary algorithms. Here, the degree of neutrality (synonymous vs. non-synonymous changes) seems to play an important role.

Examples for neutrality in fitness landscapes are the ND family (see Section 21.2.3), the NKp and NKq models (discussed in Section 21.2.1), and the Royal Road (see Section 21.2.4). Another common instance of neutrality is *bloat* in Genetic Programming, which is outlined in Section 4.10.3 on page 224.

Neutral Networks

From the idea of neutral bridges between different parts of the search space as sketched by Smith et al. [1913], we can derive the concept of neutral networks.

Definition 1.51 (Neutral Network). Neutral networks are equivalence classes K of elements of the search space $\mathbb{G}$ which map to elements of the problem space $\mathbb{X}$ with the same objective values and are connected by chains of applications of the search operators Op [149].

$$\forall g_1, g_2 \in \mathbb{G} : g_1 \in K(g_2) \subseteq \mathbb{G} \Leftrightarrow \exists k \in \mathbb{N}_0 : P\big(g_2 = Op^k(g_1)\big) > 0 \wedge F(\text{gpm}(g_1)) = F(\text{gpm}(g_2)) \tag{1.44}$$

Barnett [149] states that a neutral network has the *constant innovation property* if

[58] http://en.wikipedia.org/wiki/Molecular_biology [accessed 2008-07-20]
[59] http://en.wikipedia.org/wiki/Evolutionary_biology [accessed 2008-07-20]

1. the rate of discovery of innovations keeps constant for a reasonably large amount of applications of the search operations [981], and
2. if this rate is comparable with that of an unconstrained random walk.

Networks with this property may prove very helpful if they connect the optima in the fitness landscape. Stewart [1962] utilizes neutral networks and the idea of punctuated equilibria in his *extrema selection*, a genetic algorithm variant that focuses on exploring individuals which are far away from the centroid of the set of currently investigated solution candidates (but have still good objective values). Then again, Barnett [148] showed that populations in genetic algorithm tend to dwell in neutral networks of high dimensions of neutrality regardless of their objective values, which (obviously) cannot be considered advantageous.

The convergence on neutral networks has furthermore been studied by Bornberg-Bauer and Chan [251], van Nimwegen et al. [2097, 2096], and Wilke [2225]. Their results show that the topology of neutral networks strongly determines the distribution of genotypes on them. Generally, the genotypes are "drawn" to the solutions with the highest degree of neutrality ν on the neutral network Beaudoin et al. [161].

Redundancy: Problematic and Beneficial

Definition 1.52 (Redundancy). Redundancy in the context of global optimization is a feature of the genotype-phenotype mapping and means that multiple genotypes map to the same phenotype, i. e., the genotype-phenotype mapping is not injective.

$$\exists g_1, g_2 : g_1 \neq g_2 \wedge \text{gpm}(g_1) = \text{gpm}(g_2) \tag{1.45}$$

The role of redundancy in the genome is as controversial as that of neutrality [2168]. There exist many accounts of its positive influence on the optimization process. Shipman et al. [1871, 1856], for instance, tried to mimic desirable evolutionary properties of RNA folding [980]. They developed redundant genotype-phenotype mappings using voting (both, via uniform redundancy and via a non-trivial approach), Turing machine-like binary instructions, Cellular automata, and random Boolean networks [1099]. Except for the trivial voting mechanism based on uniform redundancy, the mappings induced neutral networks which proved beneficial for exploring the problem space. Especially the last approach provided particularly good results [1871, 1856]. Possibly converse effects like epistasis (see Section 1.4.6) arising from the new genotype-phenotype mappings have not been considered in this study.

Redundancy can have a strong impact on the explorability of the problem space. When utilizing a one-to-one mapping, the translation of a slightly modified genotype will always result in a different phenotype. If there exists a many-to-one mapping between genotypes and phenotypes, the search operations can create offspring genotypes different from the parent which still translate to the same phenotype. The optimizer may now walk along a path through this neutral network. If many genotypes along this path can be modified to different offspring, many new solution candidates can be reached [1871]. One example for beneficial redundancy is the extradimensional bypass idea discussed in Section 1.5.2.

The experiments of Shipman et al. [1872, 1870] additionally indicate that neutrality in the genotype-phenotype mapping can have positive effects. In the Cartesian Genetic Programming method, neutrality is explicitly introduced in order to increase the evolvability (see Section 4.7.4 on page 201) [2110, 2297].

Yet, Rothlauf [1765] and Shackleton et al. [1856] show that simple uniform redundancy is not necessarily beneficial for the optimization process and may even slow it down. There is no use in introducing encodings which, for instance, represent each phenotypic bit with two bits in the genotype where `00` and `01` map to `0` and `10` and `11` map to `1`. Another example for this issue is given in Fig. 1.31.b on page 86.

Summary

Different from ruggedness which is always bad for optimization algorithms, neutrality has aspects that may further as well as hinder the process of finding good solutions. Generally we can state that degrees of neutrality ν very close to 1 degenerate optimization processes to random walks. Some forms of neutral networks accompanied by low (nonzero) values of ν can improve the evolvability and hence, increase the chance of finding good solutions.

Adverse forms of neutrality are often caused by bad design of the search space or genotype-phenotype mapping. Uniform redundancy in the genome should be avoided where possible and the amount of neutrality in the search space should generally be limited.

Needle-In-A-Haystack

One of the worst cases of fitness landscapes is the *needle-in-a-haystack* (NIAH) problem sketched in Fig. 1.19.g on page 57, where the optimum occurs as isolated spike in a plane. In other words, small instances of extreme ruggedness combine with a general lack of information in the fitness landscape. Such problems are extremely hard to solve and the optimization processes often will converge prematurely or take very long to find the global optimum. An example for such fitness landscapes is the all-or-nothing property often inherent to Genetic Programming of algorithms [2058], as discussed in Section 4.10.2 on page 223.

1.4.6 Epistasis

Introduction

In biology, *epistasis*[60] is defined as a form of interaction between different genes [1640]. The term was coined by Bateson [157] and originally meant that one gene suppresses the phenotypical expression of another gene. In the context of statistical genetics, epistasis was initially called "epistacy" by Fisher [677]. According to Lush [1335], the interaction between genes is epistatic if the effect on the fitness of altering one gene depends on the allelic state of other genes. This understanding of epistasis comes very close to another biological expression: *Pleiotropy*[61], which means that a single gene influences multiple phenotypic traits [2227]. In the area of global optimization, such fine-grained distinctions are usually not made and the two terms are often used more or less synonymously.

Definition 1.53 (Epistasis). In optimization, epistasis is the dependency of the contribution of one gene to the value of the objective functions on the allelic state of other genes. [491, 44, 1503]

We speak of minimal epistasis when every gene is independent of every other gene. Then, the optimization process equals finding the best value for each gene and can most efficiently be carried out by a simple greedy search (see Section 17.4.1) [491]. A problem is maximally epistatic when no proper subset of genes is independent of any other gene [1924, 1503]. Examples of problems with a high degree of epistasis are Kauffman's NK fitness landscape [1098, 1100] (Section 21.2.1), the p-Spin model [48] (Section 21.2.2), and the tunable model of Weise et al. [2185] (Section 21.2.7).

The Problem

As sketched in Figure 1.22, epistasis has a strong influence on many of the previously discussed problematic features. If one gene can "turn off" or affect the expression of other

[60] http://en.wikipedia.org/wiki/Epistasis [accessed 2008-05-31]
[61] http://en.wikipedia.org/wiki/Pleiotropy [accessed 2008-03-02]

genes, a modification of this gene will lead to a large change in the features of the phenotype. Hence, the *causality* will be weakened and *ruggedness* ensues in the fitness landscape. It also becomes harder to define search operations with exploitive character. Moreover, subsequent changes to the "deactivated" genes may have no influence on the phenotype at all, which would then increase the degree of *neutrality* in the search space. Epistasis is mainly an aspect of the way in which the genome $\mathbb{G}$ and the genotype-phenotype mapping are defined. It should be avoided where possible.

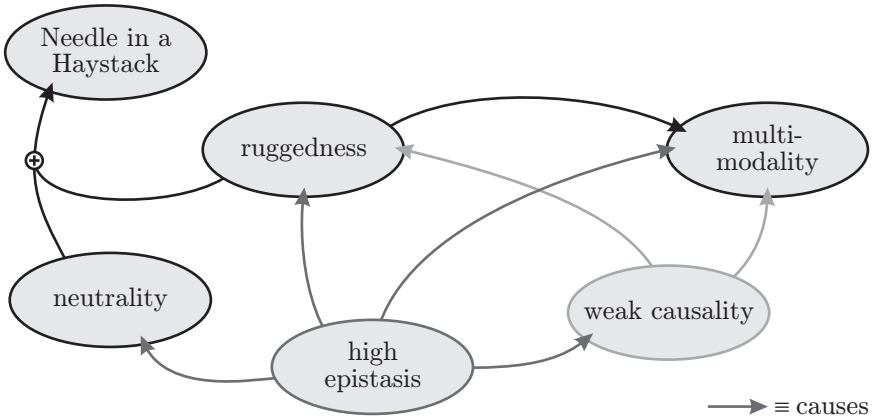


Figure 1.22: The influence of epistasis on the fitness landscape.

Generally, epistasis and conflicting objectives in multi-objective optimization should be distinguished from each other. Epistasis as well as pleiotropy is a property of the influence of the editable elements (the genes) of the genotypes on the phenotypes. Objective functions can conflict *without* the involvement of any of these phenomena. We can, for example, define two objective functions $f_1(x) = x$ and $f_2(x) = -x$ which are clearly contradicting regardless of whether they both are subject to maximization or minimization. Nevertheless, if the solution candidates x and the genotypes are simple real numbers and the genotype-phenotype mapping is an identity mapping, neither epistatic nor pleiotropic effects can occur.

Naudts and Verschoren [1504] have shown for the special case of length-two binary string genomes that deceptiveness does not occur in situations with low epistasis and also that objective functions with high epistasis are not necessarily deceptive. Another discussion about different shapes of fitness landscapes under the influence of epistasis is given by Beerenwinkel et al. [167].

Countermeasures

General

We have shown that epistasis is a root cause for multiple problematic features of optimization tasks. General countermeasures against epistasis can be divided into two groups. The symptoms of epistasis can be mitigated with the same methods which increase the chance of finding good solutions in the presence of ruggedness or neutrality – using larger populations and favoring explorative search operations. Epistasis itself is a feature which results from the choice of the search space structure, the search operations, and the genotype-phenotype mapping. Avoiding epistatic effects should be a major concern during their design. This can lead to a great improvement in the quality of the solutions produced by the optimization process [2181]. Some general rules for search space design are outlined in Section 1.5.2.

Linkage Learning

According to Winter et al. [2242], *linkage* is "the tendency for alleles of different genes to be passed together from one generation to the next" in genetics. This usually indicates that these genes are closely located in the same chromosome. In the context of evolutionary algorithms, this notation is not useful since identifying spatially close elements inside the genotypes $g \in \mathbb{G}$ is trivial. Instead, we are interested in alleles of different genes which have a joint effect on the fitness [1486, 1485].

Identifying these linked genes, i. e., learning their epistatic interaction, is very helpful for the optimization process. Such knowledge can be used to protect building blocks[62] from being destroyed by the search operations (such as crossover in genetic algorithms), for instance. Finding approaches for *linkage learning* has become an especially popular discipline in the area of evolutionary algorithms with binary [896, 1486, 1647] and real [546] genomes. Two important methods from this area are the *messy GA* (mGA, see Section 3.7) by Goldberg et al. [825] and the *Bayesian Optimization Algorithm* (BOA) [1633, 333]. Module acquisition [66] may be considered as such an effort.

1.4.7 Noise and Robustness

Introduction – Noise

In the context of optimization, three types of noise can be distinguished. The first form is noise in the training data used as basis for learning *(i)*. In many applications of machine learning or optimization where a model for a given system is to be learned, data samples including the input of the system and its measured response are used for training. Some typical examples of situations where training data is the basis for the objective function evaluation are

1. the usage of global optimization for building classifiers (for example for predicting buying behavior using data gathered in a customer survey for training),
2. the usage of simulations for determining the objective values in Genetic Programming (here, the simulated scenarios correspond to training cases), and
3. the fitting of mathematical functions to (x, y)-data samples (with artificial neural networks or symbolic regression, for instance).

Since no measurement device is 100% accurate and there are always random errors, noise is present in such optimization problems.

Besides inexactnesses and fluctuations in the input data of the optimization process, perturbations are also likely to occur during the application of its results. This category subsumes the other two types of noise: perturbations that may arise from *(ii)* inaccuracies in the process of realizing the solutions and *(iii)* environmentally induced perturbations during the applications of the products.

This issue can be illustrated by using the process of developing the perfect tire for a car as an example. As input for the optimizer, all sorts of material coefficients and geometric constants measured from all known types of wheels and rubber could be available. Since these constants have been measured or calculated from measurements, they include a certain degree of noise and imprecision *(i)*.

The result of the optimization process will be the best tire construction plan discovered during its course and it will likely incorporate different materials and structures. We would hope that the tires created according to the plan will not fall apart if, accidently, an extra 0.0001% of a specific rubber component is used *(ii)*. During the optimization process, the behavior of many construction plans will be simulated in order to find out about their utility. When actually manufactured, the tires should not behave unexpectedly when used

[62] See Section 3.6.5 for information on the Building Block Hypothesis.

in scenarios different from those simulated *(iii)* and should instead be applicable in all driving situations likely to occur.

The effects of noise in optimization have been studied by various researchers; Miller and Goldberg [1416, 1415], Lee and Wong [1268], and Gurin and Rastrigin [870] are some of them. Many global optimization algorithms and theoretical results have been proposed which can deal with noise. Some of them are, for instance, specialized

1. genetic algorithms [685, 2062, 2060, 1799, 1800, 1146],
2. Evolution Strategies [195, 100, 881], and
3. Particle Swarm Optimization [1606, 884] approaches.

The Problem: Need for Robustness

The goal of global optimization is to find the global optima of the objective functions. While this is fully true from a theoretical point of view, it may not suffice in practice. Optimization problems are normally used to find good parameters or designs for components or plans to be put into action by human beings or machines. As we have already pointed out, there will always be noise and perturbations in practical realizations of the results of optimization. There is no process in the world that is 100% accurate and the optimized parameters, designs, and plans have to tolerate a certain degree of imprecision.

Definition 1.54 (Robustness). A system in engineering or biology is*robust* if it is able to function properly in the face of genetic or environmental perturbations [2132].

Therefore, a local optimum (or even a non-optimal element) for which slight disturbances only lead to gentle performance degenerations is usually favored over a global optimum located in a highly rugged area of the fitness landscape [276]. In other words, local optima in regions of the fitness landscape with strong causality are sometimes better than global optima with weak causality. Of course, the level of this acceptability is application-dependent. Figure 1.23 illustrates the issue of local optima which are robust vs. global optima which are not. More examples from the real world are:

1. When optimizing the control parameters of an airplane or a nuclear power plant, the global optimum is certainly not used if a slight perturbation can have hazardous effects on the system [2062].
2. Wiesmann et al. [2218, 2217] bring up the topic of manufacturing tolerances in multilayer optical coatings. It is no use to find optimal configurations if they only perform optimal when manufactured to a precision which is either impossible or too hard to achieve on a constant basis.
3. The optimization of the decision process on which roads should be precautionary salted for areas with marginal winter climate is an example of the need for dynamic robustness. The global optimum of this problem is likely to depend on the daily (or even current) weather forecast and may therefore be constantly changing. Handa et al. [886] point out that it is practically infeasible to let road workers follow a constantly changing plan and circumvent this problem by incorporating multiple road temperature settings in the objective function evaluation.
4. Tsutsui et al. [2062, 2060] found a nice analogy in nature: The phenotypic characteristics of an individual are described by its genetic code. During the interpretation of this code, perturbations like abnormal temperature, nutritional imbalances, injuries, illnesses and so on may occur. If the phenotypic features emerging under these influences have low fitness, the organism cannot survive and procreate. Thus, even a species with good genetic material will die out if its phenotypic features become too sensitive to perturbations. Species robust against them, on the other hand, will survive and evolve.

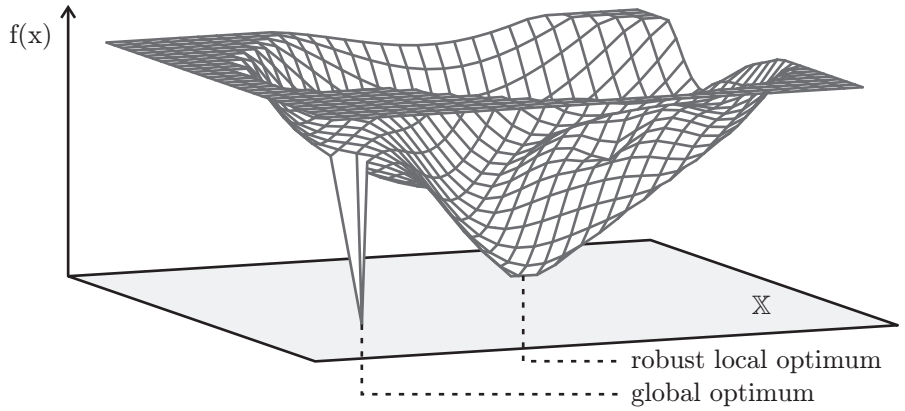


Figure 1.23: A robust local optimum vs. a "unstable" global optimum.

Countermeasures

For the special case where the phenome is a real vector space ($\mathbb{X} \subseteq \mathbb{R}^n$), several approaches for dealing with the need for robustness have been developed. Inspired by Taguchi methods[63] [1995], possible disturbances are represented by a vector $\boldsymbol{\delta} = (\delta_1, \delta_2, .., \delta_n)^T, \delta_i \in \mathbb{R}$ in the method suggested by Greiner [859, 860]. If the distributions and influences of the δ_i are known, the objective function $f(\mathbf{x}) : \mathbf{x} \in \mathbb{X}$ can be rewritten as $\tilde{f}(\mathbf{x}, \boldsymbol{\delta})$ [2218]. In the special case where $\boldsymbol{\delta}$ is normally distributed, this can be simplified to $\tilde{f}\Big((x_1 + \delta_1, x_2 + \delta_2, .., x_n + \delta_n)^T\Big)$. It would then make sense to sample the probability distribution of $\boldsymbol{\delta}$ a number of t times and to use the mean values of $\tilde{f}(\mathbf{x}, \boldsymbol{\delta})$ for each objective function evaluation during the optimization process. In cases where the optimal value $y^\star$ of the objective function f is known, Equation 1.46 can be minimized. This approach is also used in the work of Wiesmann et al. [2217, 2218] and basically turns the optimization algorithm into something like a maximum likelihood estimator (see Section 28.7.2 and Equation 28.252 on page 502).

$$f'(\mathbf{x}) = \frac{1}{t}\sum_{i=1}^{t}\left(y^\star - \tilde{f}(\mathbf{x}, \boldsymbol{\delta}_i)\right)^2 \tag{1.46}$$

This method corresponds to using multiple, different training scenarios during the objective function evaluation in situations where $\mathbb{X} \not\subseteq \mathbb{R}^n$. By adding random noise and artificial perturbations to the training cases, the chance of obtaining robust solutions which are stable when applied or realized under noisy conditions can be increased.

1.4.8 Overfitting and Oversimplification

In all scenarios where optimizers evaluate some of the objective values of the solution candidates by using training data, two additional phenomena with negative influence can be observed: overfitting and oversimplification.

Overfitting

The Problem

Definition 1.55 (Overfitting). Overfitting[64] is the emergence of an overly complicated model (solution candidate) in an optimization process resulting from the effort to provide the best results for as much of the available training data as possible [1805, 1905, 785, 564].

[63] http://en.wikipedia.org/wiki/Taguchi_methods [accessed 2008-07-19]
[64] http://en.wikipedia.org/wiki/Overfitting [accessed 2007-07-03]

A model (solution candidate) $m \in \mathbb{X}$ optimized based on a finite set of training data is considered to be overfitted if a less complicated, alternative model $m' \in \mathbb{X}$ exists which has a smaller error for the set of all possible (maybe even infinitely many), available, or (theoretically) producible data samples. This model m' may, however, have a larger error in the training data.

The phenomenon of overfitting is best known and can often be encountered in the field of artificial neural networks or in curve fitting[65] [2019, 1291, 1265, 1806, 1761]. The latter means that we have a set A of n training data samples (x_i, y_i) and want to find a function f that represents these samples as well as possible, i. e., $f(x_i) = y_i \; \forall (x_i, y_i) \in A$.

There exists exactly one polynomial[66] of the degree $n-1$ that fits to each such training data and goes through all its points.[67] Hence, when only polynomial regression is performed, there is exactly one perfectly fitting function of minimal degree. Nevertheless, there will also be an infinite number of polynomials with a higher degree than $n-1$ that also match the sample data perfectly. Such results would be considered as overfitted.

In Figure 1.24, we have sketched this problem. The function $f_1(x) = x$ shown in Fig. 1.24.b has been sampled three times, as sketched in Fig. 1.24.a. There exists no other polynomial of a degree of two or less that fits to these samples than f_1. Optimizers, however, could also find overfitted polynomials of a higher degree such as f_2 which also match the data, as shown in Fig. 1.24.c. Here, f_2 plays the role of the overly complicated model m which will perform as good as the simpler model m' when tested with the training sets only, but will fail to deliver good results for all other input data.

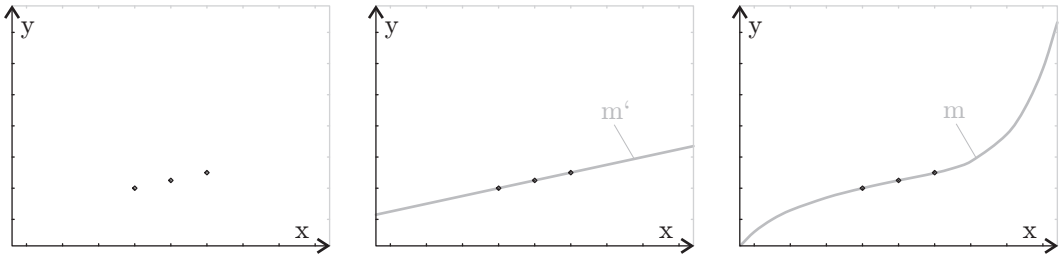


Fig. 1.24.a: Three sample points of f_1.

Fig. 1.24.b: $m' \equiv f_1(x) = x$.

Fig. 1.24.c: $m \equiv f_2(x)$.

Figure 1.24: Overfitting due to complexity.

A very common cause for overfitting is noise in the sample data. As we have already pointed out, there exists no measurement device for physical processes which delivers perfect results without error. Surveys that represent the opinions of people on a certain topic or randomized simulations will exhibit variations from the true interdependencies of the observed entities, too. Hence, data samples based on measurements will always contain some noise.

In Figure 1.25 we have sketched how such noise may lead to overfitted results. Fig. 1.25.a illustrates a simple physical process obeying some quadratic equation. This process has been measured using some technical equipment and the 100 noisy samples depicted in Fig. 1.25.b has been obtained. Fig. 1.25.c shows a function resulting from an optimization that fits the data perfectly. It could, for instance, be a polynomial of degree 99 that goes right through all the points and thus, has an error of zero. Although being a perfect match to the

[65] We will discuss overfitting in conjunction with Genetic Programming-based symbolic regression in Section 23.1 on page 397.

[66] http://en.wikipedia.org/wiki/Polynomial [accessed 2007-07-03]

[67] http://en.wikipedia.org/wiki/Polynomial_interpolation [accessed 2008-03-01]

measurements, this complicated model does not accurately represent the physical law that produced the sample data and will not deliver precise results for new, different inputs.

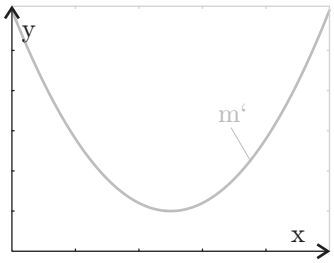


Fig. 1.25.a: The original physical process.

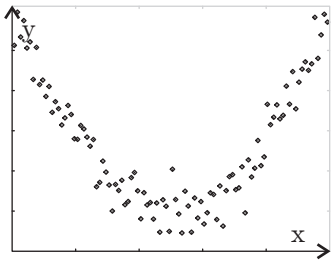


Fig. 1.25.b: The measurement/training data.

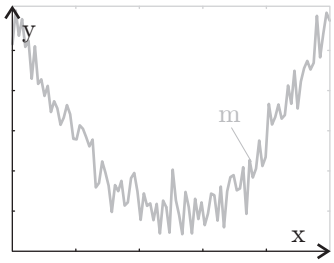


Fig. 1.25.c: The overfitted result.

Figure 1.25: Fitting noise.

From the examples we can see that the major problem that results from overfitted solutions is the loss of generality.

Definition 1.56 (Generality). A solution of an optimization process is general if it is not only valid for the sample inputs $a_1, a_2, \ldots, a_n$ which were used for training during the optimization process, but also for different inputs $a \neq a_i \ \forall i : 0 < i \leq n$ if such inputs a exist.

Countermeasures

There exist multiple techniques that can be utilized in order to prevent overfitting to a certain degree. It is most efficient to apply multiple such techniques together in order to achieve best results.

A very simple approach is to restrict the problem space $\mathbb{X}$ in a way that only solutions up to a given maximum complexity can be found. In terms of function fitting, this could mean limiting the maximum degree of the polynomials to be tested. Furthermore, the functional objective functions which solely concentrate on the error of the solution candidates should be augmented by penalty terms and non-functional objective functions putting pressure in the direction of small and simple models [564, 1108].

Large sets of sample data, although slowing down the optimization process, may improve the generalization capabilities of the derived solutions. If arbitrarily many training datasets or training scenarios can be generated, there are two approaches which work against overfitting:

1. The first method is to use a new set of (randomized) scenarios for each evaluation of each solution candidate. The resulting objective values then may differ largely even if the same individual is evaluated twice in a row, introducing incoherence and ruggedness into the fitness landscape.
2. At the beginning of each iteration of the optimizer, a new set of (randomized) scenarios is generated which is used for all individual evaluations during that iteration. This method leads to objective values which can be compared without bias. They can be made even more comparable if the objective functions are always normalized into some fixed interval, say $[0, 1]$.

In both cases it is helpful to use more than one training sample or scenario per evaluation and to set the resulting objective value to the average (or better median) of the outcomes.

Otherwise, the fluctuations of the objective values between the iterations will be very large, making it hard for the optimizers to follow a stable gradient for multiple steps.

Another simple method to prevent overfitting is to limit the runtime of the optimizers [1805]. It is commonly assumed that learning processes normally first find relatively general solutions which subsequently begin to overfit because the noise "is learned", too.

For the same reason, some algorithms allow to decrease the rate at which the solution candidates are modified by time. Such a decay of the learning rate makes overfitting less likely.

Dividing Data into Training and Test Sets If only one finite set of data samples is available for training/optimization, it is common practice to separate it into a set of training data A_t and a set of test cases A_c. During the optimization process, only the training data is used. The resulting solutions are tested with the test cases afterwards. If their behavior is significantly worse when applied to A_c than when applied to A_t, they are probably overfitted.

The same approach can be used to detect when the optimization process should be stopped. The best known solution candidates can be checked with the test cases in each iteration without influencing their objective values which solely depend on the training data. If their performance on the test cases begins to decrease, there are no benefits in letting the optimization process continue any further.

Oversimplification

The Problem

Oversimplification (also called overgeneralization) is the opposite of overfitting. Whereas overfitting denotes the emergence of overly complicated solution candidates, oversimplified solutions are not complicated enough. Although they represent the training samples used during the optimization process seemingly well, they are rough overgeneralizations which fail to provide good results for cases not part of the training.

A common cause for oversimplification is sketched in Figure 1.26: The training sets only represent a fraction of the set of possible inputs. As this is normally the case, one should always be aware that such an incomplete coverage may fail to represent some of the dependencies and characteristics of the data, which then may lead to oversimplified solutions. Another possible reason for oversimplification is that ruggedness, deceptiveness, too much neutrality, or high epistasis in the fitness landscape may lead to premature convergence and prevent the optimizer from surpassing a certain quality of the solution candidates. It then cannot adapt them completely even if the training data perfectly represents the sampled process. A third possible cause is that a problem space could have been chosen which does not include the correct solution.

Fig. 1.26.a shows a cubic function. Since it is a polynomial of degree three, four sample points are needed for its unique identification. Maybe not knowing this, only three samples have been provided in Fig. 1.26.b. By doing so, some vital characteristics of the function are lost. Fig. 1.26.c depicts a square function – the polynomial of the lowest degree that fits exactly to these samples. Although it is a perfect match, this function does not touch any other point on the original cubic curve and behaves totally differently at the lower parameter area.

However, even if we had included point P in our training data, it would still be possible that the optimization process would yield Fig. 1.26.c as a result. Having training data that correctly represents the sampled system does not mean that the optimizer is able to find a correct solution with perfect fitness – the other, previously discussed problematic phenomena can prevent it from doing so. Furthermore, if it was not known that the system which was to be modeled by the optimization process can best be represented by a polynomial of the third degree, one could have limited the problem space $\mathbb{X}$ to polynomials of degree two and less. Then, the result would likely again be something like Fig. 1.26.c, regardless of how many training samples are used.

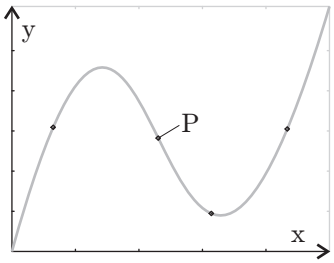


Fig. 1.26.a: The "real system" and the points describing it.

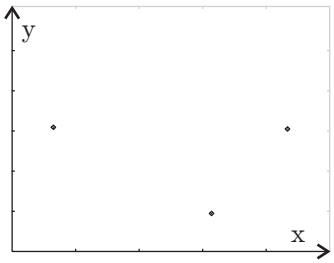


Fig. 1.26.b: The sampled training data.

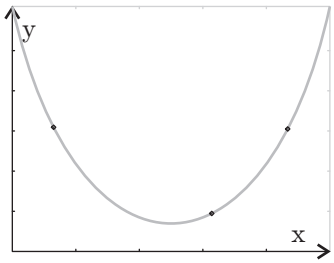


Fig. 1.26.c: The oversimplified result.

Figure 1.26: Oversimplification.

Countermeasures

In order to counter oversimplification, its causes have to be mitigated. Generally, it is not possible to have training scenarios which cover the complete input space of the evolved programs. By using multiple scenarios for each individual evaluation, the chance of missing important aspects is decreased. These scenarios can be replaced with new, randomly created ones in each generation, in order to decrease this chance even more. The problem space, i. e., the representation of the solution candidates, should further be chosen in a way which allows constructing a correct solution to the problem defined. Then again, releasing too many constraints on the solution structure increases the risk of overfitting and thus, careful proceeding is recommended.

1.4.9 Dynamically Changing Fitness Landscape

It should also be mentioned that there exist problems with dynamically changing fitness landscapes [282, 1465, 1729, 277, 278]. The task of an optimization algorithm is then to provide solution candidates with momentarily optimal objective values for each point in time. Here we have the problem that an optimum in iteration t will possibly not be an optimum in iteration $t + 1$ anymore.

Problems with dynamic characteristics can, for example, be tackled with special forms [2280] of

1. evolutionary algorithms [2053, 2224, 279, 280, 1463, 1464, 82],
2. genetic algorithms [817, 1457, 1458, 1459, 1146],
3. Particle Swarm Optimization [343, 344, 1280, 1605, 211],
4. Differential Evolution [1391, 2266], and
5. Ant Colony Optimization [868, 869]

The moving peaks benchmarks by Branke [277, 278] and Morrison and De Jong [1465] are good examples for dynamically changing fitness landscapes. You can find them discussed in Section 21.1.3 on page 328.

1.4.10 The No Free Lunch Theorem

By now, we know the most important problems that can be encountered when applying an optimization algorithm to a given problem. Furthermore, we have seen that it is arguable what actually an optimum is if multiple criteria are optimized at once. The fact that there

is most likely no optimization method that can outperform all others on all problems can, thus, easily be accepted. Instead, there exist a variety of optimization methods specialized in solving different types of problems. There are also algorithms which deliver good results for many different problem classes, but may be outperformed by highly specialized methods in each of them. These facts have been formalized by Wolpert and Macready [2244, 2245] in their *No Free Lunch Theorems*[68] (NFL) for search and optimization algorithms.

Initial Definitions

Wolpert and Macready [2245] consider single-objective optimization and define an optimization problem $\phi(g) \equiv f(\text{gpm}(g))$ as a mapping of a search space $\mathbb{G}$ to the objective space $\mathbb{Y}$.[69] Since this definition subsumes the problem space and the genotype-phenotype mapping, only skipping the possible search operations, it is very similar to our Definition 1.34 on page 46. They further call a time-ordered set d_m of m distinct visited points in $\mathbb{G} \times \mathbb{Y}$ a "sample" of size m and write $d_m \equiv \{(d_m^g(1), d_m^y(1)), (d_m^g(2), d_m^y(2)), \ldots, (d_m^g(m), d_m^y(m))\}$. $d_m^g(i)$ is the genotype and $d_m^y(i)$ the corresponding objective value visited at time step i. Then, the set $D_m = (\mathbb{G} \times \mathbb{Y})^m$ is the space of all possible samples of length m and $D = \cup_{m \geq 0} D_m$ is the set of all samples of arbitrary size.

An optimization algorithm a can now be considered to be a mapping of the previously visited points in the search space (i. e., a sample) to the next point to be visited. Formally, this means $a : D \mapsto \mathbb{G}$. Without loss of generality, Wolpert and Macready [2245] only regard unique visits and thus define $a : d \in D \mapsto g : g \notin d$.

Performance measures Ψ can be defined independently from the optimization algorithms only based on the values of the objective function visited in the samples d_m. If the objective function is subject to minimization, $\Psi(d_m^y) = \min\{d_m^y : i = 1..m\}$ would be the appropriate measure.

Often, only parts of the optimization problem ϕ are known. If the minima of the objective function f were already identified beforehand, for instance, its optimization would be useless. Since the behavior in wide areas of ϕ is not obvious, it makes sense to define a probability $P(\phi)$ that we are actually dealing with ϕ and no other problem. Wolpert and Macready [2245] use the handy example of the travelling salesman problem in order to illustrate this issue. Each distinct TSP produces a different structure of ϕ. Yet, we would use the same optimization algorithm a for all problems of this class without knowing the exact shape of ϕ. This corresponds to the assumption that there is a set of very similar optimization problems which we may encounter here although their exact structure is not known. We act as if there was a probability distribution over all possible problems which is non-zero for the TSP-alike ones and zero for all others.

The Theorem

The performance of an algorithm a iterated m times on an optimization problem ϕ can then be defined as $P(d_m^y | \phi, m, a)$, i. e., the conditional probability of finding a particular sample d_m^y. Notice that this measure is very similar to the value of the problem landscape $\Phi(x, \tau)$ introduced in Definition 1.38 on page 48 which is the cumulative probability that the optimizer has visited the element $x \in \mathbb{X}$ until (inclusively) the τ^{th} evaluation of the objective function(s).

Wolpert and Macready [2245] prove that the sum of such probabilities over all possible optimization problems ϕ is always identical for all optimization algorithms. For two optimizers a_1 and a_2, this means that

[68] http://en.wikipedia.org/wiki/No_free_lunch_in_search_and_optimization [accessed 2008-03-28]

[69] Notice that we have partly utilized our own notations here in order to be consistent throughout the book.

$$\sum_{\forall\phi} P(d_m^y\,|\phi, m, a_1\,) = \sum_{\forall\phi} P(d_m^y\,|\phi, m, a_2\,) \tag{1.47}$$

Hence, the average over all ϕ of $P(d_m^y\,|\phi, m, a\,)$ is independent of a.

Implications

From this theorem, we can immediately follow that, in order to outperform a_1 in one optimization problem, a_2 will necessarily perform worse in another. Figure 1.27 visualizes this issue. It shows that general optimization approaches like evolutionary algorithms can solve a variety of problem classes with reasonable performance. In this figure, we have chosen a performance measure Φ subject to maximization, i. e., the higher its values, the faster will the problem be solved. Hill climbing approaches, for instance, will be much faster than evolutionary algorithms if the objective functions are steady and monotonous, that is, in a smaller set of optimization tasks. Greedy search methods will perform fast on all problems with matroid[70] structure. Evolutionary algorithms will most often still be able to solve these problems, it just takes them longer to do so. The performance of hill climbing and greedy approaches degenerates in other classes of optimization tasks as a trade-off for their high utility in their "area of expertise".

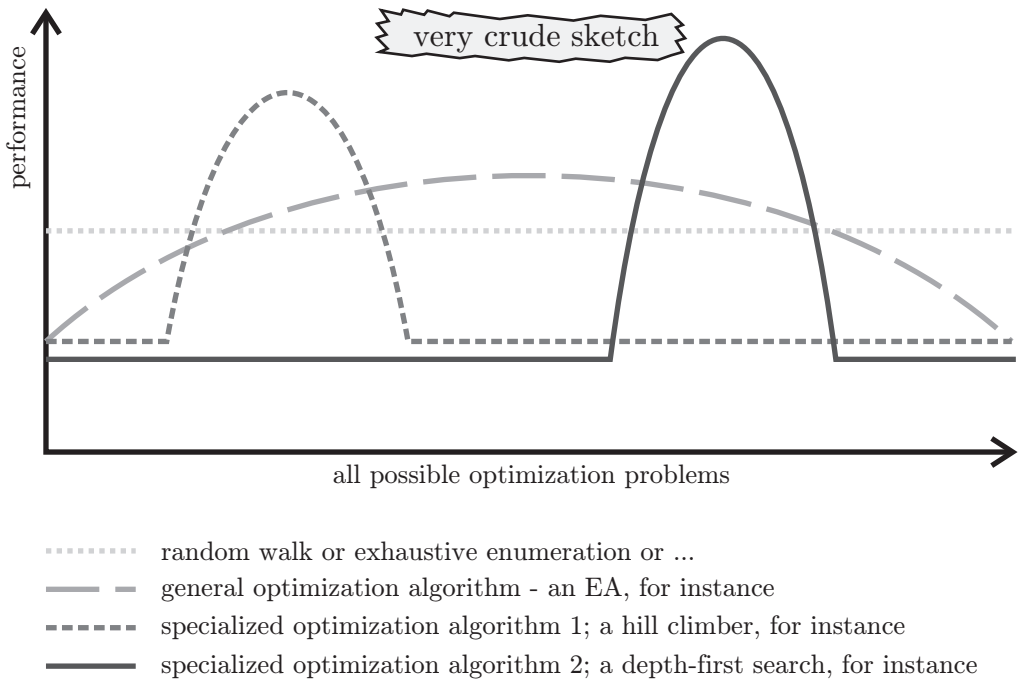


Figure 1.27: A visualization of the No Free Lunch Theorem.

One interpretation of the No Free Lunch Theorem is that it is impossible for any optimization algorithm to outperform random walks or exhaustive enumerations on all possible problems. For every problem where a given method leads to good results, we can construct a problem where the same method has exactly the opposite effect (see Section 1.4.4). As a matter of fact, doing so is even a common practice to find weaknesses of optimization algorithms and to compare them with each other, see Section 21.2.6, for example.

[70] http://en.wikipedia.org/wiki/Matroid [accessed 2008-03-28]

Another interpretation is that every useful optimization algorithm utilizes some form of problem-specific knowledge. Radcliffe [1696] states that without such knowledge, search algorithms cannot exceed the performance of simple enumerations. Incorporating knowledge starts with relying on simple assumptions like "if x is a good solution candidate, than we can expect other good solution candidates in its vicinity", i. e., strong causality. The more (correct) problem specific knowledge is integrated (correctly) into the algorithm structure, the better will the algorithm perform. On the other hand, knowledge correct for one class of problems is, quite possibly, misleading for another class. In reality, we use optimizers to solve a given set of problems and are not interested in their performance when (wrongly) applied to other classes.

The rough meaning of the NLF is that all black-box optimization methods perform equally well over the complete set of all optimization problems [1563]. In practice, we do not want to apply an optimizer to all possible problems but to only some, restricted classes. In terms of these classes, we can make statements about which optimizer performs better.

Today, there exists a wide range of work on No Free Lunch Theorems for many different aspects of machine learning. The website `http://www.no-free-lunch.org/`[71] gives a good overview about them. Further summaries, extensions, and criticisms have been provided by Köppen et al. [1173], Droste et al. [602, 601, 599, 600], Oltean [1563], and Igel and Toussaint [1008, 1009]. Radcliffe and Surry [1694] discuss the NFL in the context of evolutionary algorithms and the representations used as search spaces. The No Free Lunch Theorem is furthermore closely related to the Ugly Duckling Theorem[72] proposed by Watanabe [2159] for classification and pattern recognition.

1.4.11 Conclusions

The subject of this introductory chapter was the question about what makes optimization problems hard, especially for metaheuristic approaches. We have discussed numerous different phenomena which can affect the optimization process and lead to disappointing results.

If an optimization process has converged prematurely, it has been trapped in a non-optimal region of the search space from which it cannot "escape" anymore (Section 1.4.2). Ruggedness (Section 1.4.3) and deceptiveness (Section 1.4.4) in the fitness landscape, often caused by epistatic effects (Section 1.4.6), can misguide the search into such a region. Neutrality and redundancy (Section 1.4.5) can either slow down optimization because the application of the search operations does not lead to a gain in information or may also contribute positively by creating neutral networks from which the search space can be explored and local optima can be escaped from. Noise is present in virtually all practical optimization problems. The solutions that are derived for them should be robust (Section 1.4.7). Also, they should neither be too general (oversimplification, Section 1.4.8) nor too specifically aligned only to the training data (overfitting, Section 1.4.8). Furthermore, many practical problems are multi-objective, i. e., involve the optimization of more than one criterion at once (partially discussed in Section 1.2.2), or concern objectives which may change over time (Section 1.4.9).

In the previous section, we discussed the No Free Lunch Theorem and argued that it is not possible to develop the *one* optimization algorithm, the problem-solving machine which can provide us with near-optimal solutions in short time for every possible optimization task. This must sound very depressing for everybody new to this subject.

Actually, quite the opposite is the case, at least from the point of view of a researcher. The No Free Lunch Theorem means that there will always be new ideas, new approaches which will lead to better optimization algorithms to solve a given problem. Instead of being doomed to obsolescence, it is far more likely that most of the currently known optimization methods have at least one niche, one area where they are excellent. It also means that it

[71] accessed: 2008-03-28

[72] `http://en.wikipedia.org/wiki/Ugly_duckling_theorem` [accessed 2008-08-22]

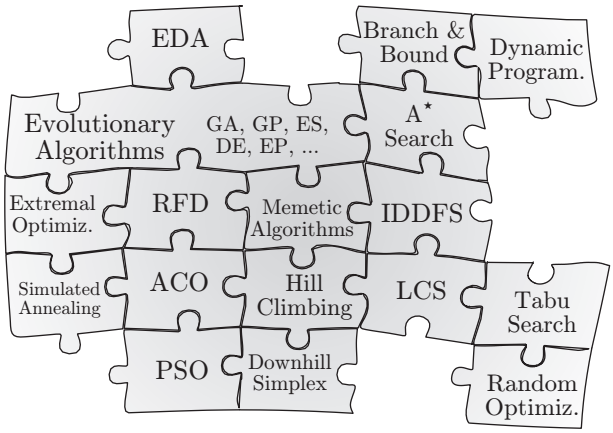


Figure 1.28: The puzzle of optimization algorithms.

is very likely that the "puzzle of optimization alorithms" will never be completed. There will always be a chance that an inspiring moment, an observation in nature, for instance, may lead to the invention of a new optimization algorithm which performs better in some problem areas than all currently known ones.

1.5 Formae and Search Space/Operator Design

Most global optimization algorithms share the premise that solutions to problems are either elements of a somewhat continuous space that can be approximated stepwise or that they can be composed of smaller modules which have good attributes even when occurring separately.

The design of the search space (or genome) $\mathbb{G}$ and the genotype-phenotype mapping gpm is vital for the success of the optimization process. It determines to what degree these expected features can be exploited by defining how the properties and the behavior of the solution candidates are encoded and how the search operations influence them. In this chapter, we will first discuss a general theory about how properties of individuals can be defined, classified, and how they are related. We will then outline some general rules for the design of the genome which are inspired by our previous discussion of the possible problematic aspects of fitness landscapes.

1.5.1 Forma Analysis

The Schema Theorem has been stated for genetic algorithms by Holland [940] in its seminal work [940, 512, 945]. In this section, we are going to discuss it in the more general version from Weicker [2167] as introduced by Radcliffe and Surry [1695] and Surry [1983] in [1692, 1696, 1691, 1691, 1695].

The different individuals p in the population Pop of the search and optimization algorithms are characterized by their properties ϕ. Whereas the optimizers themselves focus mainly on the phenotypical properties since these are evaluated by the objective functions, the properties of the genotypes may be of interest in an analysis of the optimization performance.

A rather structural property ϕ_1 of formulas $f : \mathbb{R} \mapsto \mathbb{R}$ in symbolic regression[73] would be whether it contains the mathematical expression $x+1$ or not. We can also declare a behavioral property ϕ_2 which is `true` if $|f(0) - 1| \leq 0.1$ holds, i. e., if the result of f is close to a value

[73] More information on symbolic regression can be found in Section 23.1 on page 397.

1 for the input 0, and `false` otherwise. Assume that the formulas were decoded from a binary search space $\mathbb{G} = \mathbb{B}^n$ to the space of trees that represent mathematical expression by a genotype-phenotype mapping. A genotypical property then would be if a certain sequence of bits occurs in the genotype $p.g$ and a phenotypical property is the number of nodes in the phenotype $p.x$, for instance. If we try to solve a graph-coloring problem, for example, a property $\phi_3 \in \{black, white, gray\}$ could denote the color of a specific vertex q as illustrated in Figure 1.29.

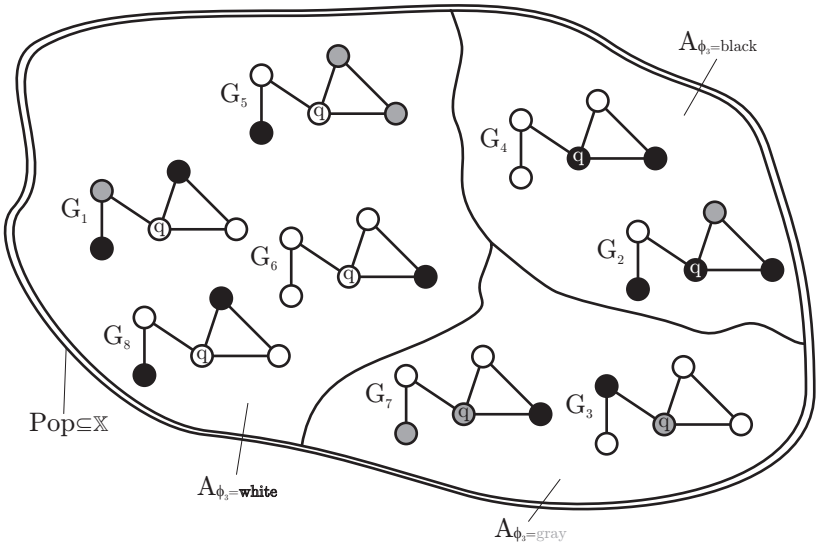


Figure 1.29: An graph coloring-based example for properties and formae.

In general, we can imagine the properties ϕ_i to be some sort of functions that map the individuals to property values. ϕ_1 and ϕ_2 would then both map the space of mathematical functions to the set $\mathbb{B} = \{\texttt{true}, \texttt{false}\}$ whereas ϕ_3 maps the space of all possible colorings for the given graph to the set $\{white, gray, black\}$. On the basis of the properties ϕ_i we can define equivalence relations[74] $\sim_{\phi_i}$:

$$p_1 \sim_{\phi_i} p_2 \Rightarrow \phi_i(p_1) = \phi_i(p_2) \;\; \forall p_1, p_2 \in \mathbb{G} \times \mathbb{X} \tag{1.48}$$

Obviously, for each two solution candidates and x_1 and x_2, either $x_1 \sim_{\phi_i} x_2$ or $x_1 \not\sim_{\phi_i} x_2$ holds. These relations divide the search space into equivalence classes $A_{\phi_i=v}$.

Definition 1.57 (Forma). An equivalence class $A_{\phi_i=v}$ that contains all the individuals sharing the same characteristic v in terms of the property ϕ_i is called a *forma* [1691] or *predicate* [2122].

$$A_{\phi_i=v} = \{\forall p \in \mathbb{G} \times \mathbb{X} : \phi_i(p) = v\} \tag{1.49}$$

$$\forall p_1, p_2 \in A_{\phi_i=v} \Rightarrow p_1 \sim_{\phi_i} p_2 \tag{1.50}$$

The number of formae induced by a property, i. e., the number of its different characteristics, is called its *precision* [1691]. The precision of ϕ_1 and ϕ_2 is 2, for ϕ_3 it is 3. We can define another property $\phi_4 \equiv f(0)$ denoting the value a mathematical function has for the input 0. This property would have an uncountable infinite large precision.

Two formae $A_{\phi_i=v}$ and $A_{\phi_j=w}$ are said to be *compatible*, written as $A_{\phi_i=v} \bowtie A_{\phi_j=w}$, if there can exist at least one individual which is an instance of both.

[74] See the definition of equivalence classes in Section 27.7.3 on page 464.

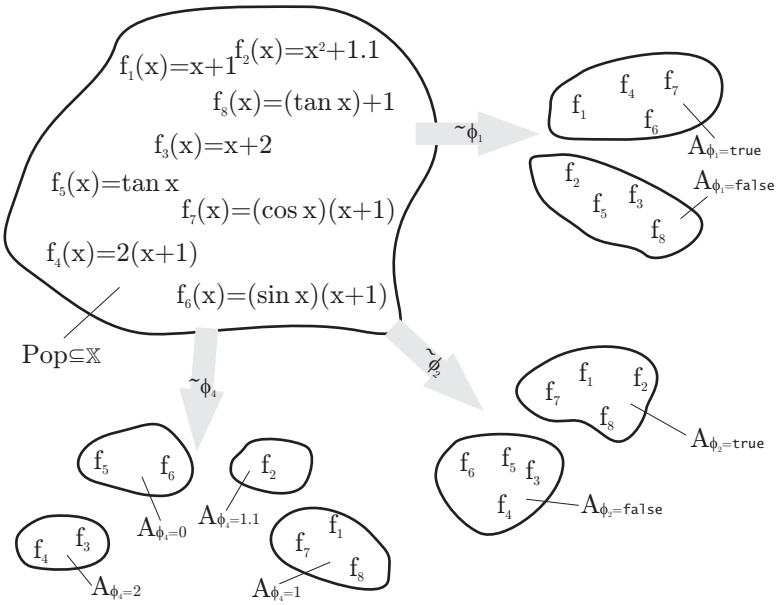


Figure 1.30: Example for formae in symbolic regression.

$$A_{\phi_i=v} \bowtie A_{\phi_j=w} \Leftrightarrow A_{\phi_i=v} \cap A_{\phi_j=w} \neq \emptyset \tag{1.51}$$

$$A_{\phi_i=v} \bowtie A_{\phi_j=w} \Leftrightarrow \exists\, p \in \mathbb{G} \times \mathbb{X} : p \in A_{\phi_i=v} \wedge p \in A_{\phi_j=w} \tag{1.52}$$

$$A_{\phi_i=v} \bowtie A_{\phi_i=w} \Rightarrow w = v \tag{1.53}$$

Of course, two different formae of the same property ϕ_i, i. e., two different characteristics of ϕi, are always incompatible. In our initial symbolic regression example hence $A_{\phi_1=\texttt{true}} \not\bowtie A_{\phi_1=\texttt{false}}$ since it is not possible that a function f contains a term $x+1$ and at the same time does not contain it. All formae of the properties ϕ1 and ϕ_2 on the other hand are compatible: $A_{\phi_1=\texttt{false}} \bowtie A_{\phi_2=\texttt{false}}$, $A_{\phi_1=\texttt{false}} \bowtie A_{\phi_2=\texttt{true}}$, $A_{\phi_1=\texttt{true}} \bowtie A_{\phi_2=\texttt{false}}$, and $A_{\phi_1=\texttt{true}} \bowtie A_{\phi_2=\texttt{true}}$ all hold. If we take ϕ_4 into consideration, we will find that there exist some formae compatible with some of ϕ_2 and some that are not, like $A_{\phi_2=\texttt{true}} \bowtie A_{\phi_4=1}$ and $A_{\phi_2=\texttt{false}} \bowtie A_{\phi_4=2}$, but $A_{\phi_2=\texttt{true}} \not\bowtie A_{\phi_4=0}$ and $A_{\phi_2=\texttt{false}} \not\bowtie A_{\phi_4=0.95}$.

The discussion of forma and their dependencies stems from the evolutionary algorithm community and there especially from the supporters of the Building Block Hypothesis. The idea is that the algorithm first discovers formae which have a good influence on the overall fitness of the solution candidates. The hope is that there are many compatible ones under these formae that are then gradually combined in the search process.

In this text we have defined formae and the corresponding terms on the basis of individuals p which are records that assign an element of the problem spaces $p.x \in \mathbb{X}$ to an element of the search space $p.g \in \mathbb{G}$. Generally, we will relax this notation and also discuss forma directly in the context of the search space $\mathbb{G}$ or problem space $\mathbb{X}$, when appropriate.

1.5.2 Genome Design

In software engineering, there are some design patterns[75] that describe good practice and experience values. Utilizing these patterns will help the software engineer to create well-organized, extensible, and maintainable applications.

Whenever we want to solve a problem with global optimization algorithms, we need to define the structure of a genome. The individual representation along with the genotype-

[75] http://en.wikipedia.org/wiki/Design_pattern_%28computer_science%29 [accessed 2007-08-12]

phenotype mapping is a vital part of genetic algorithms and has major impact on the chance of finding good solutions.

We have already discussed the basic problems that we may encounter during optimization. The choice of the search space, the search operations, and the genotype-phenotype mapping have major impact on the chance of finding good solutions. After formalizing the ideas of properties and formae, we will now outline some general best practices for the genome design from different perspectives. These principles can lead to finding better solutions or higher optimization speed if considered in the design phase [1765, 1525].

In Goldberg [821] defines two general design patterns for genotypes in genetic algorithm which we will state here in the context of the forma analysis [1525]:

1. The representations of the formae in the search space should be as short as possible and the representations of different, compatible phenotypic formae should not influence each other.
2. The alphabet of the encoding and the lengths of the different genes should be as small as possible.

Both rules target for minimal redundancy in the genomes. We have already mentioned in Section 1.4.5 on page 67 that uniform redundancy slows down the optimization process. Especially the second rule focuses on this cause of neutrality by discouraging the use of unnecessary large alphabets for encoding in a genetic algorithm. Palmer and Kershenbaum [1602, 1603] define additional rules for tree-representations in [1602, 1601], which have been generalized by Nguyen [1525]:

3. A good search space and genotype-phenotype mapping should be able to represent all phenotypes, i. e., be surjective (see Section 27.7 on page 461).

$$\forall x \in \mathbb{X} \Rightarrow \exists g \in \mathbb{G} : x = \text{gpm}(g) \tag{1.54}$$

4. The search space $\mathbb{G}$ should be unbiased in the sense that all phenotypes are represented by the same number of genotypes. This property allows to efficiently select an unbiased start population, giving the optimizer the chance of reaching all parts of the problem space.

$$\forall x_1, x_2 \in \mathbb{X} \Rightarrow |\{g \in \mathbb{G} : x_1 = \text{gpm}(g)\}| \approx |\{g \in \mathbb{G} : x_2 = \text{gpm}(g)\}| \tag{1.55}$$

5. The genotype-phenotype mapping should always yield *valid* phenotypes. The meaning of valid in this context is that if the problem space $\mathbb{X}$ is the set of all possible trees, only trees should be encoded in the genome. If we use the $\mathbb{R}^3$ as problem space, no vectors with fewer or more elements than three should be produced by the genotype-phenotype mapping. This form of validity does not imply that the individuals are also *correct* solutions in terms of the objective functions.
6. The genotype-phenotype mapping should be simple and bijective.
7. The representations in the search space should possess strong causality (locality), i. e., small changes in the genotype lead to small changes in the phenotype (see Section 1.4.3). Optimally, this would mean that:

$$\forall x_1, x_2 \in \mathbb{X}, g \in \mathbb{G} : x_1 = \text{gpm}(g) \wedge x_2 = \text{gpm}(\text{searchOp}(g)) \Rightarrow x_2 \approx x_1 \tag{1.56}$$

Ronald [1752] summarizes some further rules [1752, 1525]:

8. The genotypic representation should be aligned to a set of reproduction operators in a way that good configurations of formae are preserved by the search operations and do not easily get lost during the exploration of the search space.
9. The representations should minimize epistasis (see Section 1.4.6 on page 68 and the 1^{st} rule).
10. The problem should be represented at an appropriate level of abstraction.

11. If a direct mapping between genotypes and phenotypes is not possible, a suitable artificial embryogeny approach should be applied.

Let us now summarize some more conclusions for search spaces based on forma analysis as stated by Radcliffe [1692] and Weicker [2167].

12. Formae in Genotypic and Phenotypic Space

The optimization algorithms find new elements in the search space $\mathbb{G}$ by applying the search operations searchOp $\in$ *Op*. These operations can only create, modify, or combine genotypical formae since they usually have no information about the problem space. Most mathematical models dealing with the propagation of formae like the Building Block Hypothesis and the Schema Theorem[76] thus focus on the search space and show that highly fit *genotypical* formae will more probably be investigated further than those of low utility. Our goal, however, is to find highly fit formae in the *problem space* $\mathbb{X}$. Such properties can only be created, modified, and combined by the search operations if they correspond to genotypical formae. A good genotype-phenotype mapping should provide this feature.

It furthermore becomes clear that useful separate properties in phenotypic space can only be combined by the search operations properly if they are represented by separate formae in genotypic space too.

13. Compatibility of Formae

Formae of different properties should be compatible. Compatible Formae in phenotypic space should also be compatible in genotypic space. This leads to a low level of epistasis and hence will increase the chance of success of the reproduction operations.

14. Inheritance of Formae

The 8^{th} rule mentioned Formae should not get lost during the exploration of the search space. From a good binary search operation like recombination (crossover) in genetic algorithms, we can expect that if its two parameters g_1 and g_2 are members of a forma A, the resulting element will also be an instance of A.

$$\forall g_1, g_2 \in A \subseteq \mathbb{G} \Rightarrow \text{searchOp}(g_1, g_2) \in A \tag{1.57}$$

If we furthermore can assume that all instances of all formae A with minimal precision ($A \in \mathit{mini}$) of an individual are inherited by at least one parent, the binary reproduction operation is considered as *pure*.

$$\forall g_3 = \text{searchOp}(g_1, g_2) \in \mathbb{G},\ \forall A \in \mathit{mini} : g_3 \in A \Rightarrow g_1 \in A \vee g_2 \in A \tag{1.58}$$

If this is the case, all properties of a genotype g_3 which is a combination of two others g_1, g_2 can be traced back to at least one of its parents. Otherwise, searchOp also performs an implicit unary search step, a mutation in genetic algorithm, for instance. Such properties, although discussed here for binary search operations only, can be extended to arbitrary n-ary operators.

[76] See Section 3.6 for more information on the Schema Theorem.

15. Combinations of Formae

If genotypes $g_1, g_2, \ldots$ which are instances of different but compatible formae $A_1 \bowtie A_2 \bowtie \ldots$ are combined by a binary (or n-ary) search operation, the resulting genotype g should be an instance of both properties, i. e., the combination of compatible formae should be a forma itself.

$$\forall g_1 \in A_1, g_2 \in A_2, \cdots \Rightarrow \text{searchOp}(g_1, g_2, \ldots) \in A_1 \cap A_2 \cap \ldots \ (\neq \emptyset) \tag{1.59}$$

If this principle holds for many individuals and formae, useful properties can be combined by the optimization step by step, narrowing down the precision of the arising, most interesting formae more and more. This should lead the search to the most promising regions of the search space.

16. Reachability of Formae

The set of available search operations *Op* should include at least one unary search operation which is able to reach all possible formae. If the binary search operations in *Op* all are pure, this unary operator is the only one (apart from creation operations) able to introduce new formae which are not yet present in the population. Hence, it should be able to find any given forma.

17. Influence of Formae

One rule which, in my opinion, was missing in the lists given by Radcliffe [1692] and Weicker [2167] is that the absolute contributions of the single formae to the overall objective values of a solution candidate should to be too different. Let us divide the phenotypic formae into those with positive and those with negative or neutral contribution and let us, for simplification purposes, assume that those with positive contribution can be arbitrarily combined. If one of the positive formae has a contribution with an absolute value much lower than those of the other positive formae, we will trip into the *problem of domino convergence* discussed in Section 1.4.2 on page 58.

Then, the search will first discover the building blocks of higher value. This, itself, is not a problem. However, as we have already pointed out in Section 1.4.2, if the search is stochastic and performs exploration steps, chances are that alleles of higher importance get destroyed during this process and have to be rediscovered. The values of the less salient formae would then play no role. Thus, the chance of finding them strongly depends on how frequent the destruction of important formae takes place.

Ideally, we would therefore design the genome and phenome in a way that the different characteristics of the solution candidate all influence the objective values to a similar degree. Then, the chance of finding good formae increases.

(18.) Extradimensional Bypass

Minimal-sized genomes are not always the best approach. An interesting aspect of genome design supporting this claim is inspired by the works of the theoretical biologist Conrad [436, 438, 440, 437]. According to his extradimensional bypass principle, it is possible to transform a rugged fitness landscape with isolated peeks into one with connected saddle points by increasing the dimensionality of the search space [387, 342]. In [440] he states that the chance of isolated peeks in randomly created fitness landscapes decreases when their dimensionality grows.

This partly contradicts rule 1 and 2 which state that genomes should be as compact as possible. Conrad [440] does not suggest that nature includes useless sequences in the genome but either genes which allow for

1. new phenotypical characteristics or
2. redundancy providing new degrees of freedom for the evolution of a species.

In some cases, such an increase in freedom makes more than up for the additional "costs" arising from the enlargement of the search space. The extradimensional bypass can be considered as an example of positive neutrality (see Section 1.4.5).

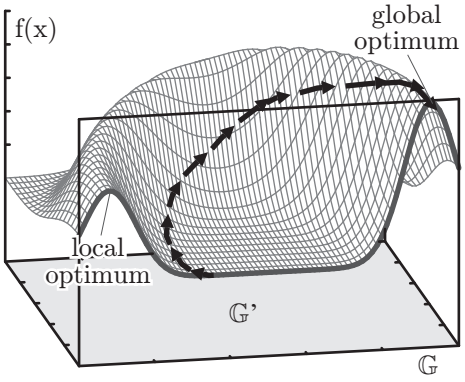


Fig. 1.31.a: Useful increase of dimensionality.

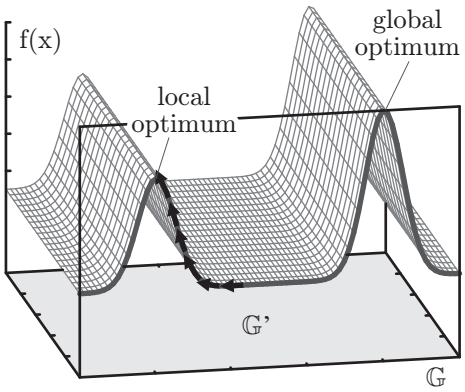


Fig. 1.31.b: Useless increase of dimensionality.

Figure 1.31: Examples for an increase of the dimensionality of a search space $\mathbb{G}$ (1d) to $\mathbb{G}'$ (2d).

In Fig. 1.31.a, an example for the extradimensional bypass (similar to Fig. 6 in [246]) is sketched. The original problem had a one-dimensional search space $\mathbb{G}$ corresponding to the horizontal axis up front. As can be seen in the plane in the foreground, the objective function had two peeks: a local optimum on the left and a global optimum on the right, separated by a larger valley. When the optimization process began climbing up the local optimum, it was very unlikely that it ever could escape this hill and reach the global one.

Increasing the search space to two dimensions ($\mathbb{G}'$), however, opened up a path way between them. The two isolated peeks became saddle points on a longer ridge. The global optimum is now reachable from all points on the local optimum.

Generally, increasing the dimension of the search space makes only sense if the added dimension has a non-trivial influence on the objective functions. Simply adding a useless new dimension (as done in Fig. 1.31.b) would be an example for some sort of uniform redundancy from which we already know (see Section 1.4.5) that it is not beneficial. Then again, adding useful new dimensions may be hard or impossible to achieve in most practical applications.

A good example for this issue is given by Bongard and Paul [246] who used an EA to evolve a neural network for the motion control of a bipedal robot. They performed runs where the evolution had control over some morphological aspects and runs where it had not. The ability to change the leg with of the robots, for instance, comes at the expense of an increase of the dimensions of the search spaced. Hence, one would expect that the optimization would perform worse. Instead, in one series of experiments, the results were much better with the extended search space. The runs did not converge to one particular leg shape but to a wide range of different structures. This led to the assumption that the morphology itself was not so much target of the optimization but the ability of changing it transformed the fitness landscape to a structure more navigable by the evolution.

In some other experimental runs of Bongard and Paul [246], this phenomenon could not be observed, most likely because

1. the robot configuration led to a problem of too high complexity, i. e., ruggedness in the fitness landscape and/or
2. the increase in dimensionality this time was too large to be compensated by the gain of evolvability.
   Further examples for possible benefits of "gradually complexifying" the search space are given by Malkin in his doctoral thesis [1351].

1.6 General Information

To all the optimization methods that are discussed in this book, you will find such a *General Information* section. Here we outline some of the applications of the respective approach, name the most important conferences, journals, and books as well as link to some online resources.

1.6.1 Areas Of Application

Some example areas of application of global optimization algorithms are:

Application	References
Chemistry, Chemical Engineering	[204, 1787, 691]
Biochemistry	[690]
Constraint Satisfaction Problems (CSP)	[1519]
Multi-Criteria Decision Making (MCDM)	[877, 375]
Biology	[691]
Engineering, Structural Optimization, and Design	[209, 691, 1814, 613, 1787, 690, 691, 379]
Economics and Finance	[613, 691, 1051]
Parameter Estimation	[690]
Mathematical Problems	[761]
Optics	[132, 2057]
Operations Research	[691, 878]
Networking and Communication	[450] Section 23.2 on page 401

This is just a small sample of the possible applications of global optimization algorithms. It has neither some sort of order nor a focus on some specific areas. In the general information sections of the following chapters, you will find many application examples for the algorithm discussed.

1.6.2 Conferences, Workshops, etc.

Some conferences, workshops and such and such on global optimization algorithms are:

AAAI: National Conference on Artificial Intelligence
http://www.aaai.org/Conferences/conferences.php [accessed 2007-09-06]
History: 2008: Chicago, Illinois, see [738]
2007: Vancouver, British Columbia, Canada, see [954]

2006: Boston, Massachusetts, USA, see [805]
2005: Pittsburgh, Pennsylvania, USA, see [1359]
2004: San Jose, California, USA, see [1381]
2002: Edmonton, Alberta, Canada, see [547]
2000: Austin, Texas, USA, see [1103]
1999: Orlando, Florida, USA, see [917]
1998: Madison, Wisconsin, USA, see [1472]
1997: Providence, Rhode Island, USA, see [1219, 3]
1996: Portland, Oregon, USA, see [410, 2]
1994: Seattle, WA, USA, see [906]
1993: Washington, DC, USA, see [668]
1992: San Jose, California, USA, see [1986]
1991: Anaheim, California, USA, see [530]
1990: Boston, Massachusetts, USA, see [563]
1988: St. Paul, Minnesota, USA, see [1435]
1987: Seattle, WA, USA, see [723]
1986: Philadelphia, PA, USA, see [1110, 1111]
1984: Austin, TX, USA, see [267]
1983: Washington, DC, USA, see [788]
1982: Pittsburgh, PA, USA, see [2143]
1980: Stanford University, California, USA, see [126]

AISB: Artificial Intelligence and Simulation of Behaviour + Workshop on Evolutionary Computing

http://www.aisb.org.uk/convention/index.shtml [accessed 2008-09-11]

History: 2008: Aberdeen, UK, see [866]
2007: Newcastle upon Tyne, UK, see [2030]
2006: Bristol, UK, see [2029]
2005: Hatfield, UK, see [2028]
2004: Leeds, UK, see [2027]
2003: Aberystwyth, UK, see [2026]
2002: Imperial College, UK, see [2025]
2001: York, UK, see [2024]
2000: Birmingham, UK, see [2023]
1997: Manchester, UK, see [447]
1996: Brighton, UK, see [695]
1995: Sheffield, UK, see [694]
1994: Leeds, UK, see [693]

HAIS: International Conference on Hybrid Artificial Intelligence Systems

http://gicap.ubu.es/hais2009/ [accessed 2009-03-02]

History: 2009: Salamanca, Spain, see [79]
2008: Burgos, Spain, see [443]
2007: Salamanca, Spain, see [442]
2006: Ribeirão Preto, SP, Brazil, see [117]

HIS: International Conference on Hybrid Intelligent Systems

http://www.softcomputing.net/hybrid.html [accessed 2007-09-01]
History: 2008: Barcelona, Spain, see [2267]
2007: Kaiserslautern, Germany, see [1170]
2006: Auckland, New Zealand, see [993]
2005: Rio de Janeiro, Brazil, see [1510]
2004: Kitakyushu, Japan, see [991]
2003: Melbourne, Australia, see [8]
2002: Santiago, Chile, see [7]
2001: Adelaide, Australia, see [6]

ICNC: International Conference on Advances in Natural Computation
History: 2007: Haikou, China, see [995, 996, 997, 998, 999]
2006: Xi'an, China, see [1052, 1053]
2005: Changsha, China, see [2151, 2152, 2153]

IAAI: Conference on Innovative Applications of Artificial Intelligence
http://www.aaai.org/Conferences/IAAI/iaai.php [accessed 2007-09-06]
History: 2006: Boston, Massachusetts, USA, see [805]
2005: Pittsburgh, Pennsylvania, USA, see [1359]
2004: San Jose, California, USA, see [1381]
2003: Acapulco, México, see [1731]
2002: Edmonton, Alberta, Canada, see [547]
2001: Seattle, Washington, USA, see [932]
2000: Austin, Texas, USA, see [1103]
1999: Orlando, Florida, USA, see [917]
1998: Madison, Wisconsin, USA, see [1472]
1997: Providence, Rhode Island, USA, see [1219]
1996: Portland, Oregon, USA, see [410]
1995: Montreal, Quebec, Canada, see [22]
1994: Seattle, Washington, USA, see [318]
1993: Washington, DC, USA, see [1]
1992: San Jose, California, USA, see [1844]
1991: Anaheim, California, USA, see [1907]
1990: Washington, DC, USA, see [1706]
1989: Stanford University, California, USA, see [1835]

KES: Knowledge-Based Intelligent Information & Engineering Systems
History: 2007: Vietri sul Mare, Italy, see [75, 76, 77]
2006: Bournemouth, UK, see [756, 757, 758]
2005: Melbourne, Australia, see [1129, 1130, 1131, 1132]
2004: Wellington, New Zealand, see [1514, 1515, 1516]
2003: Oxford, UK, see [1599, 1600]
2002: Podere d'Ombriano, Crema, Italy, see [481]
2001: Osaka and Nara, Japan, see [1037]
2000: Brighton, UK, see [962, 963]
1999: Adelaide, South Australia, see [1032]

1998: Adelaide, South Australia, see [1033, 1034, 1035]
1997: Adelaide, South Australia, see [1030, 1031]

MCDM: International Conference on Multiple Criteria Decision Making
http://project.hkkk.fi/MCDM/conf.html [accessed 2007-09-10]
History: 2008: Auckland, New Zealand, see [620]
2006: Chania, Crete, Greece, see [2333]
2004: Whistler, British Columbia, Canada, see [2165]
2002: Semmering, Austria, see [1334]
2000: Ankara, Turkey, see [1167]
1998: Charlottesville, Virginia, USA, see [877]
1997: Cape Town, South Africa, see [1963]
1995: Hagen, Germany, see [645]
1994: Coimbra, Portugal, see [419]
1992: Taipei, Taiwan, see [2069]
1990: Fairfax, USA, see [1916]
1988: Manchester, UK, see [1301]
1986: Kyoto, Japan, see [1500]
1984: Cleveland, Ohio, USA, see [876]
1982: Mons, Belgium, see [893]
1980: Newark, Delaware, USA, see [1467]
1979: Königswinter, Germany, see [644]
1977: Buffalo, New York, USA, see [2328]
1975: Jouy-en-Josas, France, see [2039]

Mendel: International Conference on Soft Computing
http://mendel-conference.org/ [accessed 2007-09-09]
History: 2009: Brno, Czech Republic, see [292]
2008: Brno, Czech Republic, see [291]
2007: Prague, Czech Republic, see [1590]
2006: Brno, Czech Republic, see [293]
2005: Brno, Czech Republic, see [2084]
2004: Brno, Czech Republic, see [2083]
2003: Brno, Czech Republic, see [2082]
2002: Brno, Czech Republic, see [2081]
2001: Brno, Czech Republic, see [2086]
2000: Brno, Czech Republic, see [1591]
1999: Brno, Czech Republic, see [2080]
1998: Brno, Czech Republic, see [2079]
1997: Brno, Czech Republic, see [2078]
1996: Brno, Czech Republic, see [2077]
1995: Brno, Czech Republic, see [2076]

MIC: Metaheuristics International Conference
History: 2007: Montreal, Canada, see [1449]
2005: Vienna, Austria, see [2115]

2003: Kyoto, Japan, see [988]
2001: Porto, Portugal, see [1721]
1999: Angra dos Reis, Brazil, see [1726]
1997: Sophia Antipolis, France, see [2124]
1995: Breckenridge, Colorado, USA, see [1589]

MICAI: Advances in Artificial Intelligence, The Mexican International Conference on Artificial Intelligence

http://www.micai.org/ [accessed 2008-06-29]

History: 2007: Aguascalientes, México, see [782]
2006: Apizaco, México, see [781, 493]
2005: Monterrey, México, see [783]
2004: Mexico City, México, see [1442]
2002: Mérida, Yucatán, México, see [425]
2000: Acapulco, México, see [325]

WOPPLOT: Workshop on Parallel Processing: Logic, Organization and Technology

History: 1992: Tutzing, Germany (?), see [2068]
1989: Neubiberg and Wildbad Kreuth, Germany, see [164]
1986: Neubiberg, see [163]
1983: Neubiberg, see [162]

In the general information sections of the following chapters, you will find many conferences and workshops that deal with the respective algorithms discussed, so this is just a small selection.

1.6.3 Journals

Some journals that deal (at least partially) with global optimization algorithms are:

Journal of Global Optimization, ISSN: 0925-5001 (Print) 1573-2916 (Online), appears monthly, publisher: Springer Netherlands, http://www.springerlink.com/content/100288/ [accessed 2007-09-20]

The Journal of the Operational Research Society, ISSN: 0160-5682, appears monthly, editor(s): John Wilson, Terry Williams, publisher: Palgrave Macmillan, The OR Society, http://www.palgrave-journals.com/jors/ [accessed 2007-09-16]

IEEE Transactions on Systems, Man, and Cybernetics (SMC), appears Part A/B: bimonthly, Part C: quaterly, editor(s): Donald E. Brown (Part A), Diane Cook (Part B), Vladimir Marik (Part C), publisher: IEEE Press, http://www.ieeesmc.org/ [accessed 2007-09-16]

Journal of Heuristics, ISSN: 1381-1231 (Print), 1572-9397 (Online), appears bi-monthly, publisher: Springer Netherlands, http://www.springerlink.com/content/102935/ [accessed 2007-09-16]

European Journal of Operational Research (EJOR), ISSN: 0377-2217, appears biweekly, editor(s): Roman Slowinski, Jesus Artalejo, Jean-Charles. Billaut, Robert Dyson, Lorenzo Peccati, publisher: North-Holland, Elsevier, http://www.elsevier.com/wps/find/journaldescription.cws_home/505543/description [accessed 2007-09-21]

Computers & Operations Research, ISSN: 0305-0548, appears monthly, editor(s): Stefan Nickel, publisher: Pergamon, Elsevier, http://www.elsevier.com/wps/find/journaldescription.cws_home/300/description [accessed 2007-09-21]

Applied Statistics, ISSN: 0035-9254, editor(s): Gilmour, Skinner, publisher: Blackwell Publishing for the Royal Statistical Society, http://www.blackwellpublishing.com/journal.asp?ref=0035-9254 [accessed 2007-09-16]

Applied Intelligence, ISSN: 0924-669X (Print), 1573-7497 (Online), appears bi-monthly, publisher: Springer Netherlands, http://www.springerlink.com/content/100236/ [accessed 2007-09-16]

Artificial Intelligence Review, ISSN: 0269-2821 (Print), 1573-7462 (Online), appears until 2005, publisher: Springer Netherlands, http://www.springerlink.com/content/100240/ [accessed 2007-09-16]

Journal of Artificial Intelligence Research (JAIR), ISSN: 11076-9757, editor(s): Toby Walsh, http://www.jair.org/ [accessed 2007-09-16]

Knowledge and Information Systems, ISSN: 0219-1377 (Print), 0219-3116 (Online), appears approx. eight times a year, publisher: Springer London, http://www.springerlink.com/content/0219-1377 [accessed 2007-09-16] and http://www.springer.com/west/home/computer/information+systems?SGWID=4-152-70-1136715-0 [accessed 2007-09-16]

SIAM Journal on Optimization (SIOPT), ISSN: 1052-6234 (print) / 1095-7189 (electronic), appears quarterly, editor(s): Nicholas I. M. Gould, publisher: Society for Industrial and Applied Mathematics, http://www.siam.org/journals/siopt.php [accessed 2008-06-14]

Applied Soft Computing, ISSN: 1568-4946, appears quarterly, editor(s): R. Roy, publisher: Elsevier B.V., http://www.sciencedirect.com/science/journal/15684946 [accessed 2008-06-15]

Advanced Engineering Informatics, ISSN: 1474-0346, appears quaterly, editor(s): J.C. Kunz, I.F.C. Smith, T. Tomiyama, publisher: Elsevier B.V., http://www.elsevier.com/wps/find/journaldescription.cws_home/622240/description [accessed 2008-08-01]

Journal of Machine Learning Research (JMLR), ISSN: 1533-7928, 1532-4435, appears 8 times/year, editor(s): Lawrence Saul and Leslie Pack Kaelbling, publisher: Microtome Publishing, http://jmlr.csail.mit.edu/ [accessed 2008-08-06]

Annals of Operations Research, ISSN: 0254-5330, 1572-9338, appears monthly, editor(s): Endre Boros, publisher: Springer, http://www.springerlink.com/content/0254-5330 [accessed 2008-10-27]

International Journal of Applied Metaheuristic Computing (IJAMC, appears starts in 2010, editor(s): Peng-Yeng Yin, publisher: Information Resources Management Association, http://www.igi-global.com/journals/details.asp?id=33344 [accessed 2009-01-02]

1.6.4 Online Resources

Some general, online available ressources on global optimization algorithms are:

http://www.mat.univie.ac.at/~neum/glopt.html [accessed 2007-09-20]

Last update:	up-to-date
Description:	Arnold Neumaier's global optimization website which includes links, publications, and software.

http://www.soft-computing.de/ [accessed 2008-05-18]

Last update:	up-to-date
Description:	Yaochu Jin's size on soft computing including links and conference infos.

http://web.ift.uib.no/~antonych/glob.html [accessed 2007-09-20]

Last update:	up-to-date
Description:	Web site with many links maintained by Gennady A. Ryzhikov.

http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO.htm [accessed 2007-11-06]
Last update: up-to-date
Description: A beautiful collection of test problems for global optimization algorithms

http://www.c2i.ntu.edu.sg/AI+CI/Resources/ [accessed 2008-20-25]
Last update: 2006-11-02
Description: A large collection of links about AI and CI.

1.6.5 Books

Some books about (or including significant information about) global optimization algorithms are:

Pardalos, Thoai, and Horst [1614]: *Introduction to Global Optimization*
Pardalos and Resende [1613]: *Handbook of Applied Optimization*
Floudas and Pardalos [691]: *Frontiers in Global Optimization*
Dzemyda, Saltenis, and Zilinskas [613]: *Stochastic and Global Optimization*
Gandibleux, Sevaux, Sörensen, and T'kindt [766]: *Metaheuristics for Multiobjective Optimisation*
Glover and Kochenberger [813]: *Handbook of Metaheuristics*
Törn and Žilinskas [2047]: *Global Optimization*
Chiong [391]: *Nature-Inspired Algorithms for Optimisation*
Floudas [690]: *Deterministic Global Optimization: Theory, Methods and Applications*
Chankong and Haimes [375]: *Multiobjective Decision Making Theory and Methodology*
Steuer [1961]: *Multiple Criteria Optimization: Theory, Computation and Application*
Haimes, Hall, and Freedman [878]: *Multiobjective Optimization in Water Resource Systems*
Charnes and Cooper [376]: *Management Models and Industrial Applications of Linear Programming*
Corne, Dorigo, Glover, Dasgupta, Moscato, Poli, and Price [448]: *New Ideas in Optimisation*
Gonzalez [832]: *Handbook of Approximation Algorithms and Metaheuristics*
Jain and Kacprzyk [1036]: *New Learning Paradigms in Soft Computing*
Tiwari, Knowles, Avineri, Dahal, and Roy [2044]: *Applications of Soft Computing – Recent Trends*
Chawdry, Roy, and Pant [379]: *Soft Computing in Engineering Design and Manufacturing*
Siarry and Michalewicz [1875]: *Advances in Metaheuristics for Hard Optimization*
Onwubolu and Babu [1580]: *New Optimization Techniques in Engineering*
Pardalos and Du [1612]: *Handbook of Combinatorial Optimization*
Reeves [1716]: *Modern Heuristic Techniques for Combinatorial Problems*
Corne, Oates, and Smith [450]: *Telecommunications Optimization: Heuristic and Adaptive Techniques*
Kontoghiorghes [1171]: *Handbook of Parallel Computing and Statistics*
Bui and Alam [299]: *Multi-Objective Optimization in Computational Intelligence: Theory and Practice*

2

Evolutionary Algorithms

2.1 Introduction

Definition 2.1 (Evolutionary Algorithm). Evolutionary algorithms[1] (EAs) are population-based metaheuristic optimization algorithms that use biology-inspired mechanisms like mutation, crossover, natural selection, and survival of the fittest in order to refine a set of solution candidates iteratively. [99, 104, 105]

The advantage of evolutionary algorithms compared to other optimization methods is their "black box" character that makes only few assumptions about the underlying objective functions. Furthermore, the definition of objective functions usually requires lesser insight to the structure of the problem space than the manual construction of an admissible heuristic. EAs therefore perform consistently well in many different problem categories.

2.1.1 The Basic Principles from Nature

In 1859, Darwin [485] published his book "On the Origin of Species"[2] in which he identified the principles of *natural selection* and *survival of the fittest* as driving forces behind the biological evolution. His theory can be condensed into ten observations and deductions [485, 1375, 2219]:

1. The individuals of a species posses great fertility and produce more offspring than can grow into adulthood.
2. Under the absence of external influences (like natural disasters, human beings, etc.), the population size of a species roughly remains constant.
3. Again, if no external influences occur, the food resources are limited but stable over time.
4. Since the individuals compete for these limited resources, a struggle for survival ensues.
5. Especially in sexual reproducing species, no two individuals are equal.
6. Some of the variations between the individuals will affect their fitness and hence, their ability to survive.
7. A good fraction of these variations are inheritable.
8. Individuals less fit are less likely to reproduce, whereas the fittest individuals will survive and produce offspring more probably.
9. Individuals that survive and reproduce will likely pass on their traits to their offspring.

[1] http://en.wikipedia.org/wiki/Artificial_evolution [accessed 2007-07-03]
[2] http://en.wikipedia.org/wiki/The_Origin_of_Species [accessed 2007-07-03]

10. A species will slowly change and adapt more and more to a given environment during this process which may finally even result in new species.

Evolutionary algorithms abstract from this biological process and also introduce a change in semantics by being *goal-driven* [2091]. The search space $\mathbb{G}$ in evolutionary algorithms is then an abstraction of the set of all possible DNA strings in nature and its elements $g \in \mathbb{G}$ play the role of the natural genotypes. Therefore, we also often refer to $\mathbb{G}$ as the *genome* and to the elements $g \in \mathbb{G}$ as *genotypes*. Like any creature is an instance of its genotype formed by embryogenesis[3], the solution candidates (or *phenotypes*) $x \in \mathbb{X}$ in the problem space $\mathbb{X}$ are instances of genotypes formed by the genotype-phenotype mapping: $x = \text{gpm}(g)$. Their fitness is rated according to objective functions which are subject to optimization and drive the evolution into specific directions.

2.1.2 The Basic Cycle of Evolutionary Algorithms

We can distinguish between single-objective and multi-objective evolutionary algorithms, where the latter means that we try to optimize multiple, possible conflicting criteria. Our following elaborations will be based on these MOEAs. The general area of Evolutionary Computation that deals with multi-objective optimization is called EMOO, evolutionary multi-objective optimization.

Definition 2.2 (MOEA). A multi-objective evolutionary algorithm (MOEA) is able to perform an optimization of multiple criteria on the basis of artificial evolution [359, 360, 2101, 534, 537, 716, 1471].

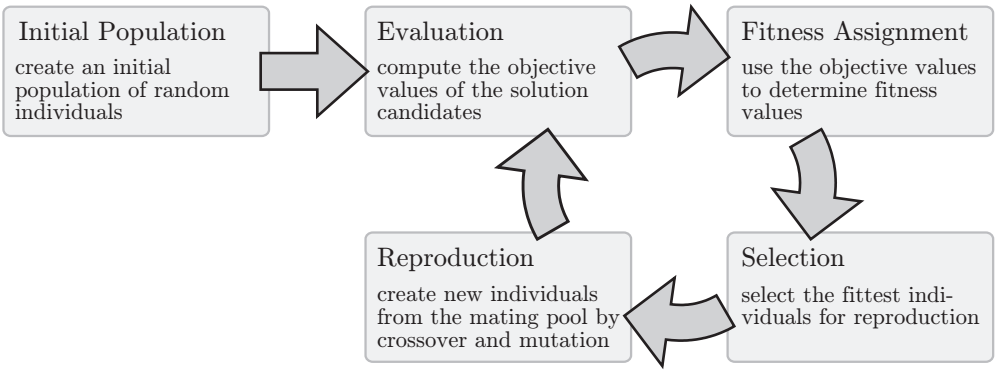


Figure 2.1: The basic cycle of evolutionary algorithms.

All evolutionary algorithms proceed in principle according to the scheme illustrated in Figure 2.1:

1. Initially, a population Pop of individuals p with a random genome $p.g$ is created.
2. The values of the objective functions $f \in F$ are computed for each solution candidate $p.x$ in Pop. This evaluation may incorporate complicated simulations and calculations.
3. With the objective functions, the utility of the different features of the solution candidates have been determined and a fitness value $v(p.x)$ can now be assigned to each of them. This fitness assignment process can, for instance, incorporate a prevalence comparator function cmp_F which uses the objective values to create an order amongst the individuals.

[3] http://en.wikipedia.org/wiki/Embryogenesis [accessed 2008-03-10]

4. A subsequent selection process filters out the solution candidates with bad fitness and allows those with good fitness to enter the mating pool with a higher probability. Since fitness is subject to minimization in the context of this book, the lower the $v(p.x)$-values are, the higher is the (relative) utility of the individual to whom they belong.
5. In the reproduction phase, offspring is created by varying or combining the genotypes $p.g$ of the selected individuals $p \in \mathit{Mate}$ by applying the search operations searchOp $\in \mathit{Op}$ (which are called *reproduction operations* in the context of EAs). These offspring are then subsequently integrated into the population.
6. If the terminationCriterion() is met, the evolution stops here. Otherwise, the algorithm continues at step 2.

In the following few paragraphs, we will discuss how the natural evolution of a species could proceed and put the artificial evolution of solution candidates in an EA into this context. When an evolutionary algorithm starts, there exists no information about what is good or what is bad. Basically, only some random genes $p.x = \text{create}()$ are coupled together as individuals in the initial population $\mathit{Pop}(t = 0)$. I think, back in the Eoarchean[4], the earth age 3.8 billion years ago where most probably the first single-celled life occurred, it was probably the same.

For simplification purposes, we will assume that the evolution does proceed stepwise in distinct generations. At the beginning of every generation, nature "instantiates" each genotype $p.g$ (given as DNA sequence) as a new phenotype $p.x = \text{gpm}(p.g)$ – a living organism – for example a fish. The survival of the genes of the fish depends on how good it performs in the ocean ($F(p.x) =?$), in other words, on how fit it is $v(p.x)$. Its fitness, however, is not only determined by one single feature of the phenotype like its size ($= f_1$). Although a bigger fish will have better chances to survive, size alone does not help if it is too slow to catch any prey ($= f_2$). Also its energy consumption f_3 should be low so it does not need to eat all the time. Other factors influencing the fitness positively are formae like sharp teeth f_4 and colors that blend into the environment f_5 so it cannot be seen too easily by sharks. If its camouflage is too good on the other hand, how will it find potential mating partners ($f_6 \nsim f_5$)? And if it is big, it will also have a higher energy consumption $f_1 \nsim f_3$. So there may be conflicts between the desired properties.

To sum it up, we could consider the life of the fish as the evaluation process of its genotype in an environment where good qualities in one aspect can turn out as drawbacks in other perspectives. In multi-objective evolutionary algorithms, this is exactly the same and I tried to demonstrate this by annotating the fish-story with the symbols previously defined in the global optimization theory sections. For each problem that we want to solve, we can specify multiple so-called objective functions $f \in F$. An objective function f represents one feature that we are interested in. Let us assume that we want to evolve a car (a pretty weird assumption, but let's stick with it). The genotype $p.g \in \mathbb{G}$ would be the construction plan and the phenotype $p.x \in \mathbb{X}$ the real car, or at least a simulation of it. One objective function f_a would definitely be *safety*. For the sake of our children and their children, the car should also be *environment-friendly*, so that's our second objective function f_b. Furthermore, a cheap price f_c, fast speed f_d, and a cool design f_e would be good. That makes five objective functions from which for example the second and the fourth are contradictory ($f_b \nsim f_d$).

After the fish genome is instantiated, nature "knows" about its phenotypic properties. Fitness, however, is always relative; it depends on your environment. I, for example, may be considered as a fit man in my department (computer science). If took a stroll to the department of sports science, that statement will probably not hold anymore. The same goes for the fish, its fitness depends on the other fish in the population (and its prey and predators). If one fish $p_1.x$ can beat another one $p_2.x$ in all categories, i.e., is bigger, stronger, smarter, and so on, we can clearly consider it as fitter ($p_1.x \succ p_2.x \Rightarrow \text{cmp}_F(p_1.x, p_2.x) < 0$) since it will have a better chance to survive. This relation is transitive but only forms a partial order since a fish that is strong but not very clever and a fish that is clever but not strong

[4] http://en.wikipedia.org/wiki/Eoarchean [accessed 2007-07-03]

maybe have the same probability to reproduce and hence, are not directly comparable[5]. Well, Ok, we cannot decide if a weak fish $p_3.x$ with a clever behavioral pattern is worse or better than a really strong but less cunning one $p_4.x$ ($\text{cmp}_F(p_3.x, p_4.x) = 0$). Both traits are furthered in the evolutionary process and maybe, one fish of the first kind will sometimes mate with one of the latter and produce an offspring which is both, intelligent and sporty[6].

Multi-objective evolutionary algorithms basically apply the same principles in their fitness assignment process "assignFitness". One of the most popular methods for computing the fitness is called *Pareto ranking*[7]. It does exactly what we've just discussed: It first chooses the individuals that are beaten by no one (we call this non-dominated set) and assigns a good (scalar) fitness value $v(p_1.x)$ to them. Then it looks at the rest of the population and picks those ($P \subset Pop$) which are not beaten by the remaining individuals and gives them a slightly worse fitness value $v(p.x) > v(p_1.x)\ \forall p \in P$ – and so on, until all solution candidates have received one scalar fitness.

Now, how fit a fish is does not necessarily determine directly if it can produce offspring. An intelligent fish may be eaten by a shark and a strong one can die from disease. The fitness[8] is only some sort of probability of reproduction. The process of selection is always stochastic, without guarantees – even a fish that is small, slow, and lacks any sophisticated behavior might survive and could produce even more offspring than a highly fit one.

The evolutionary algorithms work in exactly the same way – they use a selection algorithm "select" in order to pick the fittest individuals and place them into the mating pool *Mate*. The oldest selection scheme is called *Roulette wheel*[9]. In the original version of this algorithm (intended for fitness maximization), the chance of an individual p to reproduce is proportional to its fitness $v(p.x)$.

Last but not least, there is the reproduction phase. Fish reproduce sexually. Whenever a female fish and a male fish mate, their genes will be recombined by crossover. Furthermore, mutations may take place which. Most often, they affect the characteristics of resulting larva only slightly [1730]. Since fit fish produce offspring with higher probability, there is a good chance that the next generation will contain at least some individuals that have combined good traits from their parents and perform even better than them.

In evolutionary algorithms, we do not have such a thing as "gender". Each individual from the mating pool can potentially be recombined with every other one. In the car example, this means that we would modify the construction plans by copying the engine of one car and placing it into the car body of another one. Also, we could alter some features like the shape of the headlights randomly. This way, we receive new construction plans for new cars. Our chance that an *environment-friendly* engine inside a *cool-looking* car will result in a car that is more likely to be bought by the customer is good. If we iteratively perform the reproduction process "reproducePop" time and again, there is a high probability that the solutions finally found will be close to optimal.

2.1.3 The Basic Evolutionary Algorithm Scheme

After this informal outline about the artificial evolution and how we can use it as an optimization method, let us now specify the basic scheme common to all evolutionary algorithms. In principle, all EAs are variations and extensions of the basic approach "simpleEA" defined Algorithm 2.1, a cycle of evaluation, selection, and reproduction repeated in each iteration t. Algorithm 2.1 relies on functions and prototypes that we will introduce step by step.

[5] Which is a very comforting thought for all computer scientists.

[6] I wonder if the girls in the sports department are open to this kind of argumentation?

[7] Pareto comparisons are discussed in Section 1.2.2 on page 31 and elaborations on Pareto ranking can be found in Section 2.3.3.

[8] This definition is fitness is not fully compatible with biological one, see Section 2.1.5 for more information on that topic.

[9] The roulette wheel selection algorithm will be introduced in Section 2.4.3 on page 124.

Algorithm 2.1: $X^\star \longleftarrow \text{simpleEA}(\text{cmp}_F, ps)$

Input: cmp_F: the comparator function which allows us to compare the utility of two solution candidates
Input: ps: the population size
Data: t: the generation counter
Data: Pop: the population
Data: $Mate$: the mating pool
Data: v: the fitness function resulting from the fitness assigning process
Output: $X^\star$: the set of the best elements found

```
1  begin
2      t ⟵ 0
3      Pop ⟵ createPop(ps)
4      while ¬terminationCriterion() do
5          v ⟵ assignFitness(Pop, cmp_F)
6          Mate ⟵ select(Pop, v, ps)
7          t ⟵ t + 1
8          Pop ⟵ reproducePop(Mate)
9      return extractPhenotypes(extractOptimalSet(Pop))
10 end
```

1. The function "createPop(ps)", which will be introduced as Algorithm 2.18 in Section 2.5 on page 137, produces an initial, randomized population consisting of ps individuals in the first iteration $t = 0$.
2. The termination criterion "terminationCriterion()" checks whether the evolutionary algorithm should terminate or continue its work, see Section 1.3.4 on page 54.
3. Most evolutionary algorithms assign a scalar fitness $v(p.x)$ to each individual p by comparing its vector of objective values $F(p.x)$ to other individuals in the population Pop. The function v is built by a fitness assignment process "assignFitness", which we will discuss in Section 2.3 on page 111 in more detail. During this procedure, the genotype-phenotype mapping is implicitly carried out as well as simulations needed to compute the objective functions $f \in F$.
4. A selection algorithm "select" (see Section 2.4 on page 121) then chooses ps interesting individuals from the population Pop and inserts them into the mating pool $Mate$.
5. With "reproducePop", a new population is generated from the individuals inside the mating pool using mutation and/or recombination. More information on reproduction can be found in Section 2.5 on page 137 and in Definition 2.13.
6. The functions "extractOptimalSet" and "extractPhenotypes" which you can find introduced in Definition 19.2 on page 308 and Equation 19.1 on page 307 are used to extract all the non-prevailed individuals $p^\star$ from the final population and to return their corresponding phenotypes $p^\star.x$ only.

2.1.4 From the Viewpoint of Formae

Let us review our introductory fish example in terms of forma analysis. Fish can, for instance, be characterized by the properties "clever" and "strong". Crudely simplified, both properties may be `true` or `false` for a single individual and hence define two formae each. A third property can be the color, for which many different possible variations exist. Some of them may be good in terms of camouflage, others maybe good in terms of finding mating partners. Now a fish can be clever and strong at the same time, as well as weak and green. Here, a living fish allows nature to evaluate the utility of at least three different formae.

This fact has first been stated by Holland [940] for genetic algorithms and is termed *implicit parallelism* (or *intrinsic parallelism*). Since then, it has been studied by many different researchers [858, 853, 188, 2123]. If the search space and the genotype-phenotype mapping

are properly designed, the implicit parallelism in conjunction with the crossover/recombination operations is one of the reasons why evolutionary algorithms are such a successful class of optimization algorithms.

2.1.5 Does the natural Paragon Fit?

At this point it should be mentioned that the direct reference to Darwinian evolution in evolutionary algorithms is somehow controversial. Paterson [1619], for example, points out that "neither GAs [genetic algorithms] nor GP [Genetic Programming] are concerned with the evolution of new species, nor do they use natural selection." On the other hand, nobody would claim that the idea of selection has not been borrowed from nature although many additions and modifications have been introduced in favor for better algorithmic performance. The second argument concerning the development of different species depends on definition: According to Wikipedia [2219], a species is a class of organisms which are very similar in many aspects such as appearance, physiology, and genetics. In principle, there is some elbowroom for us and we may indeed consider even different solutions to a single problem in evolutionary algorithms as members of a different species – especially if the binary search operation crossover/recombination applied to their genomes cannot produce another valid solution candidate.

Another interesting difference was pointed out by Sharpe [1859] who states that natural evolution "only proceed[s] sufficiently fast to ensure survival" whereas evolutionary algorithms used for engineering need to be fast in order to be feasible and to compete with other problem solving techniques.

Furthermore, although the concept of fitness[10] in nature is controversial [1915], it is often considered as an *a posteriori measurement*. It then defines the ratio of the numbers of occurrences of a genotype in a population after and before selection or the number of offspring an individual has in relation to the number of offspring of another individual. In evolutionary algorithms, fitness is an *a priori quantity* denoting a value that determines the expected number of instances of a genotype that should survive the selection process. However, one could conclude that biological fitness is just an approximation of the a priori quantity arisen due to the hardness (if not impossibility) of directly measuring it.

My personal opinion (which may as well be wrong) is that the citation of Darwin here is well motivated since there are close parallels between Darwinian evolution and evolutionary algorithms. Nevertheless, natural and artificial evolution are still two different things and phenomena observed in either of the two do not necessarily carry over to the other.

2.1.6 Classification of Evolutionary Algorithms

The Family of Evolutionary Algorithms

The family of evolutionary algorithms encompasses five members, as illustrated in Figure 2.2. We will only enumerate them here in short. In depth discussions will follow in the next chapters.

1. **Genetic algorithms** (GAs) are introduced in Chapter 3 on page 141. GAs subsume all evolutionary algorithms which have bit strings as search space $\mathbb{G}$.
2. The set of evolutionary algorithms which explore the space of real vectors $\mathbb{X} \subseteq \mathbb{R}^n$ is called **Evolution Strategies** (ES, see Chapter 5 on page 227).
3. For **Genetic Programming** (GP), which will be elaborated on in Chapter 4 on page 157, we can provide two definitions: On one hand, GP includes all evolutionary algorithms that grow programs, algorithms, and these alike. On the other hand, also all EAs that evolve tree-shaped individuals are instances of Genetic Programming.

[10] http://en.wikipedia.org/wiki/Fitness_(biology) [accessed 2008-08-10]

4. **Learning Classifier Systems** (LCS), discussed in Chapter 7 on page 233, are online learning approaches that assign output values to given input values. They internally use a genetic algorithm to find new rules for this mapping.
5. **Evolutionary programming** (EP, see Chapter 6 on page 231) is an evolutionary approach that treats the instances of the genome as different species rather than as individuals. Over the decades, it has more or less merged into Genetic Programming and the other evolutionary algorithms.

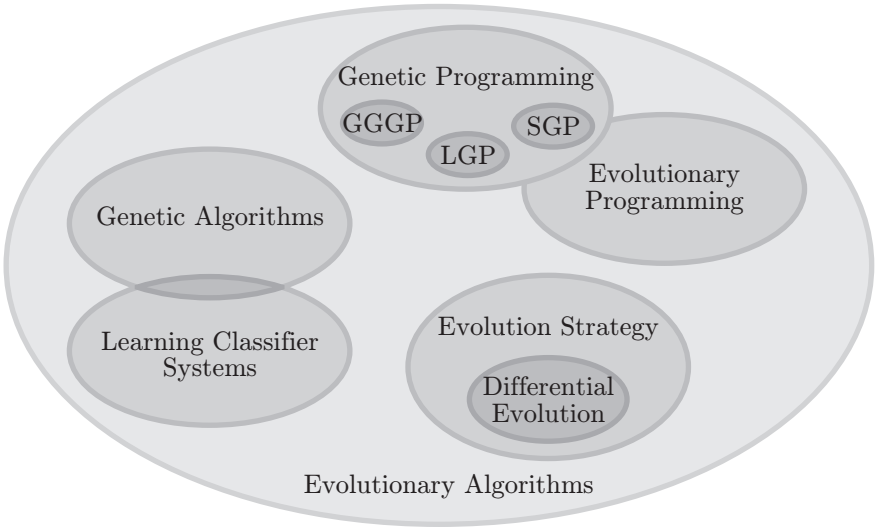


Figure 2.2: The family of evolutionary algorithms.

The early research [518] in genetic algorithms (see Section 3.1 on page 141), Genetic Programming (see Section 4.1.1 on page 157), and evolutionary programming (see Section 6.1 on page 231) date back to the 1950s and 60s. Besides the pioneering work listed in these sections, at least other important early contribution should not go unmentioned here: The Evolutionary Operation (EVOP) approach introduced by Box [260], Box and Draper [261] in the late 1950s. The idea of EVOP was to apply a continuous and systematic scheme of small changes in the control variables of a process. The effects of these modifications are evaluated and the process is slowly shifted into the direction of improvement. This idea was never realized as a computer algorithm, but Spendley et al. [1941] used it as basis for their simplex method which then served as progenitor of the downhill simplex algorithm[11] of Nelder and Mead [1517]. [518, 1276] Satterthwaite's REVOP [1815, 1816], a randomized Evolutionary Operation approach, however, was rejected at this time [518].

We now have classified different evolutionary algorithms according to their semantics, in other words, corresponding to their special search and problem spaces. All five major approaches can be realized with the basic scheme defined in Algorithm 2.1. To this simple structure, there exist many general improvements and extensions. Since these normally do not concern the search or problem spaces, they also can be applied to all members of the EA family alike. In the further text of this chapter, we will discuss the major components of many of today's most efficient evolutionary algorithms [357]. The distinctive features of these EAs are:

1. The population size or the number of populations used.

[11] We discuss Nelder and Mead [1517]'s downhill simplex optimization method in Chapter 16 on page 283.

2. The method of selecting the individuals for reproduction.
3. The way the offspring is included into the population(s).

Populations in Evolutionary Algorithms

There exist various way in which an evolutionary algorithm can process its population. Especially interesting is how the population $Pop(t+1)$ of the next iteration is formed as a combination of the current one $Pop(t)$ and its offspring. If it only contains this offspring, we speak of *extinctive selection* [1512, 1869]. Extinctive selection can be compared with ecosystems of small protozoa[12] which reproduce in a fissiparous[13] manner. In this case, of course, the elders will not be present in the next generation. Other comparisons can partly be drawn to the sexual reproducing to octopi, where the female dies after protecting the eggs until the larvae hatch, or to the black widow spider where the female devours the male after the insemination. Especially in the area of genetic algorithms, extinctive strategies are also known as *generational algorithms*.

Definition 2.3 (Generational). In evolutionary algorithms that are *generational* [1677], the next generation will only contain the offspring of the current one and no parent individuals will be preserved.

Extinctive evolutionary algorithms can further be divided into *left* and *right* selection [2264]. In left extinctive selections, the best individuals are not allowed to reproduce in order to prevent premature convergence of the optimization process. Conversely, the worst individuals are not permitted to breed in right extinctive selection schemes in order to reduce the selective pressure since they would otherwise scatter the fitness too much.

In algorithms that apply a *preservative selection* scheme, the population is a combination of the next population and the offspring [102, 1064, 1762, 2091]. The biological metaphor for such algorithms is that the lifespan of many organisms exceeds a single generation. Hence, parent and child individuals compete with each other for survival.

For Evolution Strategywhich you can find discussed in Chapter 5 on page 227, there exists a notation which also can be used describe the generation transition in evolutionary algorithms in general [934, 935, 1841, 102].

1. λ denotes the number of offspring created and
2. μ is the number of parent individuals.

Extinctive selection patterns are denoted as (μ, λ)-strategies and will create $\lambda \geq \mu$ child individuals from the μ available genotypes. From these, they only keep the μ best solution candidates and discard the μ parents as well as the $\lambda - \mu$ worst children.

In $(\mu + \lambda)$-strategy, again λ children are generated from μ parents, often with $\lambda > \mu$. Then, the parent and offspring populations are united (to a population of the size $\lambda + \mu$) and from this unison, only the μ best individuals will "survive". $(\mu + \lambda)$-strategies are thus preservative.

Steady-state evolutionary algorithms [1746, 499, 1538, 365, 1987, 2211], abbreviated by SSEA, are preservative evolutionary algorithms with values of λ that are relatively low in comparison with μ. Usually, λ is chosen in a way that a binary search operator crossover is applied exactly once per generation. Although steady-state evolutionary algorithms are often observed to produce better results than generational EAs. Chafekar et al. [365], for example, introduce steady-state evolutionary algorithms that are able to outperform generational NSGA-II (which you can find summarized in **??** on page ??) for some difficult problems. In experiments of Jones and Soule [1066] (primarily focused on other issues), steady-state algorithms showed better convergence behavior in a multi-modal landscape. Similar results

[12] http://en.wikipedia.org/wiki/Protozoa [accessed 2008-03-12]
[13] http://en.wikipedia.org/wiki/Binary_fission [accessed 2008-03-12]

have been reported by Chevreux [389] in the context of molecule design optimization. Different generational selection methods have been compared to the steady-state GENITOR approach by Goldberg and Deb [822]. On the other hand, with steady-state approaches, we run also the risk of premature convergence.

Even in preservative strategies, it is not granted that the best individuals will always survive. In principle, a $(\mu + \lambda)$ strategy can also mean that from $\mu + \lambda$ individuals, μ are chosen with a certain selection algorithm. Most are randomized, and even if such methods pick the best solution candidates with the highest probabilities, they may also select worse individuals. At this point, it is maybe interesting to mention that the idea that larger populations will always lead to better optimization results does not necessarily always hold, as shown by van Nimwegen and Crutchfield [2096].

Definition 2.4 (Elitism). An elitist evolutionary algorithm [512, 1261, 359] ensures that at least one copy of the best individual(s) of the current generation is propagated on to the next generation.

The main advantage of elitism is that its convergence is guaranteed, meaning that once the global optimum has been discovered, the evolutionary algorithm converges to that optimum. On the other hand, the risk of converging to a local optimum is also higher. Elitism is an additional feature of global optimization algorithms – a special type of preservative strategy – which is often realized by using a secondary population only containing the non-prevailed individuals. This population is updated at the end of each iteration. Such an archive-based elitism can be combined with both, generational and preservative strategies. Algorithm 2.2 specifies the basic scheme of elitist evolutionary algorithms.

Algorithm 2.2: $X^\star \longleftarrow \text{elitistEA}(\text{cmp}_F, ps, a)$

Input: cmp_F: the comparator function which allows us to compare the utility of two solution candidates
Input: ps: the population size
Input: as: the archive size
Data: t: the generation counter
Data: Pop: the population
Data: $Mate$: the mating pool
Data: Arc: the archive with the best individuals found so far
Data: v: the fitness function resulting from the fitness assigning process
Output: $X^\star$: the set of best solution candidates discovered

```
1  begin
2      t ⟵ 0
3      Arc ⟵ ∅
4      Pop ⟵ createPop(ps)
5      while ¬terminationCriterion() do
6          Arc ⟵ updateOptimalSetN(Arc, Pop)
7          Arc ⟵ pruneOptimalSet(Arc, as)
8          v ⟵ assignFitness(Pop, Arc, cmp_F)
9          Mate ⟵ select(Pop, Arc, v, ps)
10         t ⟵ t + 1
11         Pop ⟵ reproducePop(Mate)
12     return extractPhenotypes(extractOptimalSet(Pop ∪ Arc))
13 end
```

Let us now outline the new methods and changes introduced in Algorithm 2.2 in short.

1. The archive Arc is the set of best individuals found by the algorithm. Initially, it is the empty set $\emptyset$. Subsequently, it is updated with the function "updateOptimalSetN"

which inserts new, unprevailed elements from the population into it and also removes individuals from the archive which are superseded by those new optima. Algorithms that realize such updating are defined in Section 19.1 on page 307.
2. If the optimal set becomes too large – it might theoretically contain uncountable many individuals – "pruneOptimalSet" reduces it to a proper size, employing techniques like clustering in order to preserve the element diversity. More about pruning can be found in Section 19.3 on page 309.
3. You should also notice that both, the fitness assignment and selection processes, of elitist evolutionary algorithms may take the archive as additional parameter. In principle, such archive-based algorithms can also be used in non-elitist evolutionary algorithms by simply replacing the parameter *Arc* with $\emptyset$.

2.1.7 Configuration Parameters of evolutionary algorithms

Figure 2.3 illustrates the basic configuration parameters of evolutionary algorithms. The performance and success of an evolutionary optimization approach applied to a problem given by a set of objective functions F and a problem space $\mathbb{X}$ is defined by

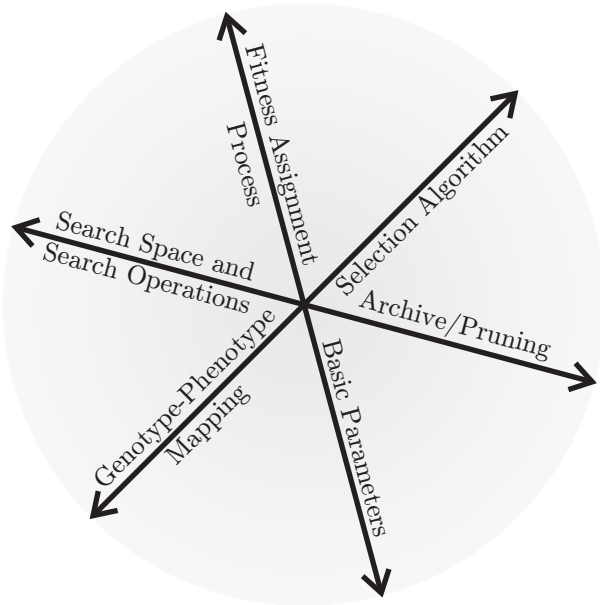


Figure 2.3: The configuration parameters of evolutionary algorithms.

1. its basic parameter settings like the population size ps or the crossover and mutation rates,
2. whether it uses an archive *Arc* of the best individuals found and, if so, which pruning technology is used to prevent it from overflowing,
3. the fitness assignment process "assignFitness" and the selection algorithm "select",
4. the choice of the search space $\mathbb{G}$ and the search operations *Op*,
5. and the genotype-phenotype mapping connecting the search Space and the problem space.

In Section 20.1, we go more into detail on how to state the configuration of an optimization algorithm in order to fully describe experiments and to make them reproducible.

2.2 General Information

2.2.1 Areas Of Application

Some example areas of application of evolutionary algorithms are:

Application	References
Function Optimization	[1562, 1673]
Multi-Objective Optimization	[715, 716, 357, 1054, 1804, 537]
Combinatorial Optimization	[254, 1762, 1270, 1338]
Engineering, Structural Optimization, and Design	[755, 1412, 1554]
Constraint Satisfaction Problems (CSP)	[2091, 1054, 716, 1804]
Economics and Finance	[388, 1975, 503, 640, 409]
Biology	[2075, 704]
Data Mining and Data Analysis	[2178, 445, 797, 444]
Mathematical Problems	[1094]
Electrical Engineering and Circuit Design	[488, 2075]
Chemistry, Chemical Engineering	[1061, 482, 389]
Scheduling	[1360, 374, 1227, 454, 250]
Robotics	[2158]
Image Processing	[322, 1532]
Networking and Communication	[1889, 1890, 453, 1497, 1684, 35] see Section 23.2 on page 401
Medicine	[411, 1911]
Ressource Minimization, Environment Surveillance/Protection	[886]
Military and Defense	[1393]
Evolving Behaviors, e.g., for Agents or Game Players	[1705]

For more information see also the application sections of the different members of the evolutionary algorithm family: genetic algorithms in Section 3.2.1 on page 142, Genetic Programming in Section 4.2.1 on page 160, Evolution Strategy in Section 5.2.1 on page 227, evolutionary programming in Section 6.2.1 on page 231, and Learning Classifier Systems in Section 7.2.1 on page 233.

2.2.2 Conferences, Workshops, etc.

Some conferences, workshops and such and such on evolutionary algorithms are:

BIOMA: International Conference on Bioinspired Optimization Methods and their Applications
http://bioma.ijs.si/ [accessed 2007-06-30]
History: 2008: Ljubljana, Slovenia, see [670]
2006: Ljubljana, Slovenia, see [669]
2004: Ljubljana, Slovenia, see [671]

CEC: Congress on Evolutionary Computation
http://ieeexplore.ieee.org/servlet/opac?punumber=7875 [accessed 2007-09-05]
History: 2008: Hong Kong, China, see [1409]
2007: Singapore, see [1005]

2006: Vancouver, BC, Canada, see [2291]
2005: Edinburgh, Scotland, UK, see [449]
2004: Portland, Oregon, USA, see [1004]
2003: Canberra, Australia, see [1803]
2002: Honolulu, HI, USA, see [703]
2001: Seoul, Korea, see [1003]
2000: La Jolla, California, USA, see [1002]
1999: Washington D.C., USA, see [69]
1998: Anchorage, Alaska, USA, see [1001]
1997: Indianapolis, IN, USA, see [106]
1996: Nagoya, Japan, see [1006]
1995: Perth, Australia, see [1000]
1994: Orlando, Florida, USA, see [1411]

Dagstuhl Seminar: Practical Approaches to Multi-Objective Optimization
History: 2006: Dagstuhl, Germany, see [283]
2004: Dagstuhl, Germany, see [281]

EA/AE: Conference on Artificial Evolution (Evolution Artificielle)
History: 2007: Tours, France, see [1441]
2005: Lille, France, see [2000]
2003: Marseilles, France, see [1283]
2001: Le Creusot, France, see [428]
1999: Dunkerque, France, see [711]
1997: Nîmes, France, see [894]
1995: Brest, France, see [41]
1994: Toulouse, France, see [40]

EMO: International Conference on Evolutionary Multi-Criterion Optimization
History: 2007: Matsushima/Sendai, Japan, see [1555]
2005: Guanajuato, México, see [422]
2003: Faro, Portugal, see [719]
2001: Zurich, Switzerland, see [2331]

EUROGEN: Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems
History: 2007: Jyväskylä, Finland, see [2072]
2005: Munich, Germany, see [1827]
2003: Barcelona, Spain, see [147]
2001: Athens, Greece, see [803]
1999: Jyväskylä, Finland, see [1413]
1997: Triest, Italy, see [1681]
1995: Las Palmas de Gran Canaria, Spain, see [1059]

EvoCOP: European Conference on Evolutionary Computation in Combinatorial Optimization
http://www.evostar.org/ [accessed 2007-09-05]
Co-located with EvoWorkshops and EuroGP.
History: 2009: Tübingen, Germany, see [455]
2008: Naples, Italy, see [2094]

2007: Valencia, Spain, see [456]
2006: Budapest, Hungary, see [843]
2005: Lausanne, Switzerland, see [1700]
2004: Coimbra, Portugal, see [842]
2003: Essex, UK, see [1701]
2002: Kinsale, Ireland, see [321]
2001: Lake Como, Milan, Italy, see [235]

EvoWorkshops: Applications of Evolutinary Computing: EvoCoMnet, EvoFIN, EvoIASP, EvoINTERACTION, EvoMUSART, EvoPhD, EvoSTOC and EvoTransLog

http://www.evostar.org/ [accessed 2007-08-05]

Co-located with EvoCOP and EuroGP.

History: 2009: Tübingen, Germany, see [802]
2008: Naples, Italy, see [801]
2007: Valencia, Spain, see [800]
2006: Budapest, Hungary, see [1768]
2005: Lausanne, Switzerland, see [1767]
2004: Coimbra, Portugal, see [1702]
2003: Essex, UK, see [1701]
2002: Kinsale, Ireland, see [321]
2001: Lake Como, Milan, Italy, see [235]
2000: Edinburgh, Scotland, UK, see [320]
1999: Göteborg, Sweden, see [1665]
1998: Paris, France, see [976]

FEA: International Workshop on Frontiers in Evolutionary Algorithms

Was part of the Joint Conference on Information Science

History: 2005: Salt Lake City, Utah, USA, see [1794]
2003: Cary, North Carolina, USA, see [639]
2002: Research Triangle Park, North Carolina, USA, see [353]
2000: Atlantic City, NJ, USA, see [2154]
1998: Research Triangle Park, North Carolina, USA, see [2021]
1997: Research Triangle Park, North Carolina, USA, see [1865]

FOCI: IEEE Symposium on Foundations of Computational Intelligence

History: 2007: Honolulu, Hawaii, USA, see [1388]

GECCO: Genetic and Evolutionary Computation Conference

http://www.sigevo.org/ [accessed 2007-08-30]

A recombination of the Annual Genetic Programming Conference (GP, see Section 4.2.2 on page 161) and the International Conference on Genetic Algorithms (ICGA, see Section 3.2.2 on page 143), also "contains" the International Workshop on Learning Classifier Systems (IWLCS, see Section 7.2.2 on page 234).

History: 2008: Atlanta, Georgia, USA, see [1117, 409, 1393, 1911, 1705]
2007: London, England, see [2037, 2038]
2006: Seattle, Washington, USA, see [352]
2005: Washington, D.C., USA, see [202, 199, 1764, 1766]
2004: Seattle, Washington, USA, see [544, 545, 1113]
2003: Chicago, Illinois, USA, see [334, 335]
2002: New York, USA, see [1245, 331, 154, 1572, 1326]
2001: San Francisco, California, USA, see [1937, 833]

2000: Las Vegas, Nevada, USA, see [2216, 2210]
1999: Orlando, Florida, USA, see [142, 1584, 1889]

GEM: International Conference on Genetic and Evolutionary Methods
see Section 3.2.2 on page 143

ICANNGA: International Conference on Adaptive and Natural Computing Algorithms
before 2005: International Conference on Artificial Neural Nets and Genetic Algorithms
History: 2007: Warsaw, Poland, see [173, 174]
2005: Coimbra, Portugal, see [1725]
2003: Roanne, France, see [1628]
2001: Prague, Czech Republic, see [1224]
1999: Portoroz, Slovenia, see [576]
1997: Norwich, England, see [1902]
1995: Alès, France, see [1627]
1993: Innsbruck, Austria, see [36]

ICNC: International Conference on Advances in Natural Computation
see Section 1.6.2 on page 89

Mendel: International Conference on Soft Computing
see Section 1.6.2 on page 90

PPSN: International Conference on Parallel Problem Solving from Nature
http://ls11-www.informatik.uni-dortmund.de/PPSN/ [accessed 2007-09-05]
History: 2008: Dortmund, Germany, see [1948]
2006: Reykjavik, Iceland, see [1779]
2004: Birmingham, UK, see [2285]
2002: Granada, Spain, see [867]
2000: Paris, France, see [1830]
1998: Amsterdam, The Netherlands, see [624]
1996: Berlin, Germany, see [2118]
1994: Jerusalem, Israel, see [492]
1992: Brussels, Belgium, see [1357]
1990: Dortmund, Germany, see [1842]

2.2.3 Journals

Some journals that deal (at least partially) with evolutionary algorithms are:

Evolutionary Computation, ISSN: 1063-6560, appears quaterly, editor(s): Marc Schoenauer, publisher: MIT Press, http://www.mitpressjournals.org/loi/evco [accessed 2007-09-16]

IEEE Transactions on Evolutionary Computation, ISSN: 1089-778X, appears bi-monthly, editor(s): Xin Yao, publisher: IEEE Computational Intelligence Society, http://ieee-cis.org/pubs/tec/ [accessed 2007-09-16]

Biological Cybernetics, ISSN: 0340-1200 (Print), 1432-0770 (Online), appears bi-monthly, publisher: Springer Berlin/Heidelberg, http://www.springerlink.com/content/100465/ [accessed 2007-09-16]

Complex Systems, ISSN: 0891-2513, appears quaterly, editor(s): Stephen Wolfram, publisher: Complex Systems Publications, Inc., http://www.complex-systems.com/ [accessed 2007-09-16]

Journal of Artificial Intelligence Research (JAIR) (see Section 1.6.3 on page 92)

New Mathematics and Natural Computation (NMNC), ISSN: 1793-0057, appears three times a year, editor(s): Paul P. Wang, publisher: World Scientific, http://www.worldscinet.com/nmnc/ [accessed 2007-09-19]

The Journal of the Operational Research Society (see Section 1.6.3 on page 91)

2.2.4 Online Resources

Some general, online available ressources on evolutionary algorithms are:

http://www.lania.mx/~ccoello/EMOO/ [accessed 2007-09-20]	
Last update:	up-to-date
Description:	EMOO Web page – Dr. Coello Coello's giant bibliography and paper repository for evolutionary multi-objective optimization.
http://www-isf.maschinenbau.uni-dortmund.de/links/ci_links.html [accessed 2007-10-14]	
Last update:	up-to-date
Description:	Computational Intelligence (CI)-related links and literature, maintained by Jörn Mehnen
http://www.aip.de/~ast/EvolCompFAQ/ [accessed 2007-09-16]	
Last update:	2001-04-01
Description:	Frequently Asked Questions of the comp.ai.genetic group by Heitkötter and Beasley [916].
http://nknucc.nknu.edu.tw/~hcwu/pdf/evolec.pdf [accessed 2007-09-16]	
Last update:	2005-02-19
Description:	Lecture Nodes on Evolutionary Computation by Wu [2264]
http://ls11-www.cs.uni-dortmund.de/people/beyer/EA-glossary/ [accessed 2008-04-10]	
Last update:	2002-02-25
Description:	Online glossary on terms and definitions in evolutionary algorithms by Beyer et al. [201]
http://www.illigal.uiuc.edu/web/ [accessed 2008-05-17]	
Last update:	up-to-date
Description:	The Illinois Genetic Algorithms Laboratory (IlliGAL)
http://www.peterindia.net/Algorithms.html [accessed 2008-05-17]	
Last update:	up-to-date
Description:	A large collection of links about evolutionary algorithms, Genetic Programming, genetic algorithms, etc.
http://www.fmi.uni-stuttgart.de/fk/evolalg/ [accessed 2008-05-17]	
Last update:	2003-07-08
Description:	The Evolutionary Computation repository of the University of Stuttgart.
http://dis.ijs.si/filipic/ec/ [accessed 2008-05-18]	
Last update:	2007-11-09
Description:	The Evolutionary Computation repository of the Jožf Stefan Institute in Slovenia
http://www.red3d.com/cwr/evolve.html [accessed 2008-05-18]	
Last update:	2002-07-27
Description:	Evolutionary Computation and its application to art and design by Craig Reynolds

http://surf.de.uu.net/encore/ [accessed 2008-05-18]	
Last update:	2004-08-26
Description:	ENCORE, the electronic appendix to The Hitch-Hiker's Guide to Evolutionary Computation, see [916]
http://www-isf.maschinenbau.uni-dortmund.de/links/ci_links.html [accessed 2008-05-18]	
Last update:	2006-09-13
Description:	A collection of links to computational intelligence / EAs
http://www.tik.ee.ethz.ch/sop/education/misc/moeaApplet/ [accessed 2008-10-25]	
Last update:	2008-06-30
Description:	An applet illustrating a multi-objective EA

2.2.5 Books

Some books about (or including significant information about) evolutionary algorithms are:

Bäck [99]: *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*
Bäck, Fogel, and Michalewicz [104]: *Handbook of Evolutionary Computation*
Ceollo Coello, Lamont, and van Veldhuizen [361]: *Evolutionary Algorithms for Solving Multi-Objective Problems*
Deb [537]: *Multi-Objective Optimization Using Evolutionary Algorithms*
Coello Coello and Lamont [424]: *Applications of Multi-Objective Evolutionary Algorithms*
Eiben and Smith [623]: *Introduction to Evolutionary Computing*
Dumitrescu, Lazzerini, Jain, and Dumitrescu [608]: *Evolutionary Computation*
Fogel [696]: *Evolutionary Computation: The Fossil Record*
Bäck, Fogel, and Michalewicz [107]: *Evolutionary Computation 1: Basic Algorithms and Operators*
Bäck, Fogel, and Michalewicz [108]: *Evolutionary Computation 2: Advanced Algorithms and Operators*
Bentley [181]: *Evolutionary Design by Computers*
De Jong [515]: *Evolutionary Computation: A Unified Approach*
Weicker [2167]: *Evolutionäre Algorithmen*
Gerdes, Klawonn, and Kruse [789]: *Evolutionäre Algorithmen*
Nissen [1535]: *Einführung in evolutionäre Algorithmen: Optimierung nach dem Vorbild der Evolution*
Yao [2284]: *Evolutionary Computation: Theory and Applications*
Yu, Davis, Baydar, and Roy [2299]: *Evolutionary Computation in Practice*
Yang, Ong, and Jin [2280]: *Evolutionary Computation in Dynamic and Uncertain Environments*
Morrison [1464]: *Designing Evolutionary Algorithms for Dynamic Environments*
Branke [280]: *Evolutionary Optimization in Dynamic Environments*
Nedjah, Alba, and Mourelle [1512]: *Parallel Evolutionary Computations*
Kosiński [1177]: *Advances in Evolutionary Algorithms*
Rothlauf [1765]: *Representations for Genetic and Evolutionary Algorithms*
Banzhaf and Eeckman [137]: *Evolution and Biocomputation – Computational Models of Evolution*

Fogel and Corne [704]: *Evolutionary Computation in Bioinformatics*
Johnston [1061]: *Applications of Evolutionary Computation in Chemistry*
Clark [411]: *Evolutionary Algorithms in Molecular Design*
Chen [388]: *Evolutionary Computation in Economics and Finance*
Ghosh and Jain [797]: *Evolutionary Computation in Data Mining*
Miettinen, Mäkelä, Neittaanmäki, and Periaux [1412]: *Evolutionary Algorithms in Engineering and Computer Science*
Fogel [698]: *Evolutionary Computation: Principles and Practice for Signal Processing*
Ashlock [85]: *Evolutionary Computation for Modeling and Optimization*
Watanabe and Hashem [2158]: *Evolutionary Computations – New Algorithms and their Applications to Evolutionary Robots*
Cagnoni, Lutton, and Olague [322]: *Genetic and Evolutionary Computation for Image Processing and Analysis*
Kramer [1214]: *Self-Adaptive Heuristics for Evolutionary Computation*
Lobo, Lima, and Michalewicz [1299]: *Parameter Setting in Evolutionary Algorithms*
Spears [1925]: *Evolutionary Algorithms – The Role of Mutation and Recombination*
Eiben and Michalewicz [621]: *Evolutionary Computation*
Jin [1055]: *Knowledge Incorporation in Evolutionary Computation*
Grosan, Abraham, and Ishibuchi [862]: *Hybrid Evolutionary Algorithms*
Abraham, Jain, and Goldberg [9]: *Evolutionary Multiobjective Optimization*
Kallel, Naudts, and Rogers [1083]: *Theoretical Aspects of Evolutionary Computing*
Ghosh and Tsutsui [798]: *Advances in Evolutionary Computing – Theory and Applications*
Yang, Shan, and Bui [2279]: *Success in Evolutionary Computation*
Pereira and Tavares [1635]: *Bio-inspired Algorithms for the Vehicle Routing Problem*

2.3 Fitness Assignment

2.3.1 Introduction

With concept of Pareto domination and prevalence comparisons introduced in Section 1.2.2 on page 27 we define a partial order on the elements in the problem space $\mathbb{X}$. In multi-objective optimization, each solution candidate $p.x$ is characterized by a vector of objective values $F(p.x)$. Many selection algorithms however cannot work with such vectors and need scalar *fitness* values instead. By assigning a single real number $v(p.x)$ (the fitness) to each solution candidate $p.x$, also a total order is defined on them.

The fitness assigned to an individual may not just reflect its rank in the population, but can also incorporate density/niching information. This way, not only the quality of a solution candidate is considered, but also the overall diversity of the population. This can improve the chance of finding the global optima as well as the performance of the optimization algorithm significantly. If many individuals in the population occupy the same rank or do not dominate each other, for instance, such information will be very helpful.

The fitness $v(p.x)$ thus may not only depend on the solution candidate $p.x$ itself, but on the whole population *Pop* of the evolutionary algorithm (and on the archive *Arc* of optimal elements, if available). In practical realizations, the fitness values are often stored in a special member variable in the individual records. Therefore, $v(p.x)$ can be considered as a mapping that returns the value of such a variable which has previously been stored there by a fitness assignment process "assignFitness".

Definition 2.5 (Fitness Assignment). A fitness assignment process "assignFitness" creates a function $v : \mathbb{X} \mapsto \mathbb{R}^+$ which relates a scalar fitness value to each solution candidate in the population *Pop* Equation 2.1 (and archive *Arc*, if an archive is available Equation 2.2).

$$v = \text{assignFitness}(Pop, \text{cmp}_F) \Rightarrow v(p.x) \in \mathbb{V} \subseteq \mathbb{R}^+ \; \forall p \in Pop \tag{2.1}$$

$$v = \text{assignFitness}(Pop, Arc, \text{cmp}_F) \Rightarrow v(p.x) \in \mathbb{V} \subseteq \mathbb{R}^+ \; \forall p \in Pop \cup Arc \tag{2.2}$$

In the context of this book, we generally minimize fitness values, i. e., the lower the fitness of a solution candidate the better. Therefore, many of the fitness assignment processes based on the prevalence relation will obey to Equation 2.3. This equation represents a general relation – sometimes it is useful to violate it for some individuals in the population, especially when crowding information is incorporated.

$$p_1.x \succ p_2.x \Rightarrow v(p_1.x) < v(p_2.x) \;\; \forall p_1, p_2 \in Pop \cup Arc \tag{2.3}$$

2.3.2 Weighted Sum Fitness Assignment

The most primitive fitness assignment strategy would be assigning a weighted sum of the objective values. This approach is very static and comes with the same problems as weighted sum-based approach for defining what an optimum is introduced in Section 1.2.2 on page 29. It makes no use of the prevalence relation. For computing the weighted sum of the different objective values of a solution candidate, we reuse Equation 1.4 on page 29 from the weighted sum optimum definition. The weights have to be chosen in a way that ensures that $v(p.x) \in \mathbb{R}^+$ holds for all individuals p.

$$v(p.x) = \text{assignFitnessWeightedSum}(Pop) \Leftrightarrow \;\; \forall p \in Pop \Rightarrow v(p.x) = g(p.x) \tag{2.4}$$

2.3.3 Pareto Ranking

Another very simple method for computing fitness values is to let them directly reflect the Pareto domination (or prevalence) relation. Figure 2.4 and Table 2.1 illustrate the Pareto relations in a population of 15 individuals and their corresponding objective values f_1 and f_2, both subject to minimization. There are two ways for doing this: First, to each individual,

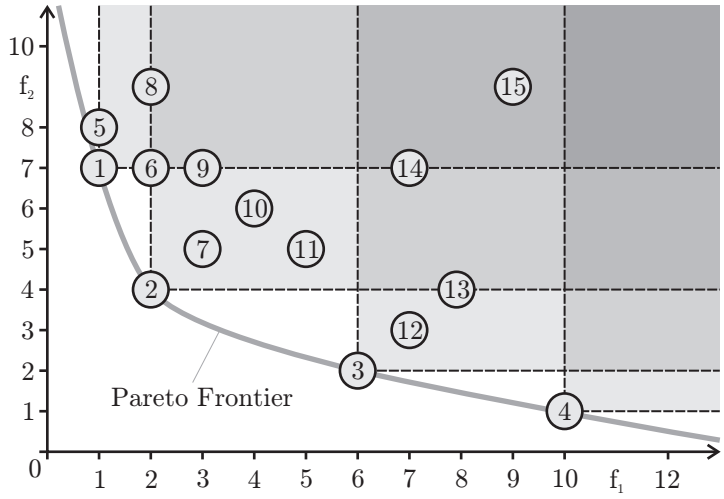


Figure 2.4: An example scenario for Pareto ranking.

we can assign a value inversely proportional to the number of other individuals it prevails, like $v(p_1.x) \equiv \frac{1}{|\forall p_2 \in Pop : p_1.x \succ p_2.x| + 1}$. We have written such fitness values in the column "Ap. 1" of Table 2.1 for Pareto optimization, i. e., the special case where the Pareto dominance

x	prevails	is prevailed by	Ap. 1	Ap. 2
1	$\{5, 6, 8, 9, 14, 15\}$	$\emptyset$	$1/7$	0
2	$\{6, 7, 8, 9, 10, 11, 13, 14, 15\}$	$\emptyset$	$1/10$	0
3	$\{12, 13, 14, 15\}$	$\emptyset$	$1/5$	0
4	$\emptyset$	$\emptyset$	1	0
5	$\{8, 15\}$	$\{1\}$	$1/3$	1
6	$\{8, 9, 14, 15\}$	$\{1, 2\}$	$1/5$	2
7	$\{9, 10, 11, 14, 15\}$	$\{2\}$	$1/6$	1
8	$\{15\}$	$\{1, 2, 5, 6\}$	$1/2$	4
9	$\{14, 15\}$	$\{1, 2, 6, 7\}$	$1/3$	4
10	$\{14, 15\}$	$\{2, 7\}$	$1/3$	2
11	$\{14, 15\}$	$\{2, 7\}$	$1/3$	2
12	$\{13, 14, 15\}$	$\{3\}$	$1/4$	1
13	$\{15\}$	$\{2, 3, 12\}$	$1/2$	3
14	$\{15\}$	$\{1, 2, 3, 6, 7, 9, 10, 11, 12\}$	$1/2$	9
15	$\emptyset$	$\{1, 2, 3, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14\}$	1	13

Table 2.1: The Pareto domination relation of the individuals illustrated in Figure 2.4.

relation is used to define prevalence. Individuals that dominate many others will here receive a lower fitness value than those which are prevailed by many. When taking a look at these values, the disadvantage of this approach becomes clear: It promotes individuals that reside in crowded region of the problem space and underrates those in sparsely explored areas.

By doing so, the fitness assignment process achieves exactly the opposite of what we want. Instead of exploring the problem space and delivering a wide scan of the frontier of best possible solution candidates, it will focus all effort on a small set of individuals. We will only obtain a subset of the best solutions and it is even possible that this fitness assignment method leads to premature convergence to a local optimum. A good example for this problem are the four non-prevailed individuals $\{1, 2, 3, 4\}$ from the Pareto frontier. The best fitness is assigned to the element 2, followed by individual 1. Although individual 7 is dominated (by 1), its fitness is better than the fitness of the non-dominated element 3.

The solution candidate 4 gets the worst possible fitness 1, since it prevails no other element. Its chances for reproduction are similarly low than those of individual 15 which is dominated by all other elements except 4. Hence, both solution candidates will most probably be not selected and vanish in the next generation. The loss of solution candidate 4 will greatly decrease the diversity and even increase the focus on the crowded area near 1 and 2.

A much better second approach for fitness assignment is directly based on the domination (or prevalence) relation and has first been proposed by Goldberg [821]. Here, the idea is to assign the number of individuals it is prevailed by to each solution candidate [1315, 253, 255, 851]. This way, the previously mentioned negative effects will not occur. The column "Ap 2" in Table 2.1 shows that all four non-prevailed individuals now have the best possible fitness 0. Hence, the exploration pressure is applied to a much wider area of the Pareto frontier. This so-called *Pareto ranking* can be performed by first removing all non-prevailed individuals from the population and assigning the rank 0 to them. Then, the same is performed with the rest of the population. The individuals only dominated by those on rank 0 (now non-dominated) will be removed and get the rank 1. This is repeated until all solution candidates have a proper fitness assigned to them. Algorithm 2.3 outlines another simple way to perform Pareto ranking. Since we follow the idea of the freer prevalence comparators instead of Pareto dominance relations, we will synonymously refer to this approach as *Prevalence ranking*.

As already mentioned, the fitness values of all non-prevailed elements in our example Figure 2.4 and Table 2.1 are equally 0. However, the region around the individuals 1 and 2 has probably already extensively been explored, whereas the surrounding of solution candi-

Algorithm 2.3: $v \longleftarrow \text{assignFitnessParetoRank}(Pop, \text{cmp}_F)$

Input: Pop: the population to assign fitness values to
Input: cmp_F: the prevalence comparator defining the prevalence relation
Data: i, j, cnt: the counter variables
Output: v: a fitness function reflecting the Prevalence ranking

```
1 begin
2   for i ⟵ len(Pop) − 1 down to 0 do
3     cnt ⟵ 0
4     p ⟵ Pop[i]
5     for j ⟵ len(Pop) − 1 down to 0 do
        // Check whether cmp_F(Pop[j].x, p.x) < 0
6       if (j ≠ i) ∧ (Pop[j].x ≻ p.x) then cnt ⟵ cnt + 1
7     v(p.x) ⟵ cnt
8   return v
9 end
```

date 4 is rather unknown. A better approach of fitness assignment should incorporate such information and put a bit more pressure into the direction of individual 4, in order to make the evolutionary algorithm investigate this area more thoroughly.

2.3.4 Sharing Functions

Previously, we have mentioned that the drawback of Pareto ranking is that it does not incorporate any information about whether the solution candidates in the population reside closely to each other or in regions of the problem space which are only sparsely covered by individuals. Sharing, as a method for including such diversity information into the fitness assignment process, was introduced by Holland [940] and later refined by Deb [532], Goldberg and Richardson [824], and Deb and Goldberg [539]. [1801, 1417, 1558]

Definition 2.6 (Sharing Function). A sharing function $\text{Sh} : \mathbb{R}^+ \mapsto \mathbb{R}^+$ is a function used to relate two individuals p_1 and p_2 to a value that decreases with their distance[14] $d = \text{dist}(p_1, p_2)$ in a way that it is 1 for $d = 0$ and 0 if the distance exceeds a specified constant σ.

$$\text{Sh}_\sigma(d = \text{dist}(p_1, p_2)) = \begin{cases} 1 & \text{if } d \leq 0 \\ \text{Sh}_\sigma(d) \in [0,1] & \text{if } 0 < d < \sigma \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

Sharing functions can be employed in many different ways and are used by a variety of fitness assignment processes [824, 532]. Typically, the simple triangular function Sh_tri [959] or one of its either convex (Sh_cvexp) or concave (Sh_ccavp) pendants with the power $p \in \mathbb{R}^+, p > 0$ are applied. Besides using different powers of the distance-σ-ratio, another approach is the exponential sharing method Sh_exp.

[14] The concept of distance and a set of different distance measures is defined in Section 29.1 on page 537.

$$\text{Sh\_tri}_\sigma(\sigma)\, d = \begin{cases} 1 - \frac{d}{\sigma} & \text{if } 0 \leq d < \sigma \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

$$\text{Sh\_cvex}_{\sigma,p}(d) = \begin{cases} \left(1 - \frac{d}{\sigma}\right)^p & \text{if } 0 \leq d < \sigma \\ 0 & \text{otherwise} \end{cases} \tag{2.7}$$

$$\text{Sh\_ccav}_{\sigma,p}(d) = \begin{cases} 1 - \left(\frac{d}{\sigma}\right)^p & \text{if } 0 \leq d < \sigma \\ 0 & \text{otherwise} \end{cases} \tag{2.8}$$

$$\text{Sh\_exp}_{\sigma,p}(d) = \begin{cases} 1 & \text{if } d \leq 0 \\ 0 & \text{if } d \geq \sigma \\ \frac{e^{-\frac{pd}{\sigma}} - e^{-p}}{1 - e^{-p}} & \text{otherwise} \end{cases} \tag{2.9}$$

For sharing, the distance of the individuals in the search space $\mathbb{G}$ as well as their distance in the problem space $\mathbb{X}$ or the objective space $\mathbb{Y}$ may be used. If the solution candidates are real vectors in the $\mathbb{R}^n$, we could use the Euclidean distance of the phenotypes of the individuals directly, i.e., compute $\text{dist}_{eucl}(p_1.x, p_2.x)$. In genetic algorithms, where the search space is the set of all bit strings $\mathbb{G} = \mathbb{B}^n$ of the length n, another suitable approach would be to use the Hamming distance[15] $\text{dist}_{Ham}(p_1.g, p_2.g)$ of the genotypes. The work of Deb [532], however, indicates that phenotypical sharing will often be superior to genotypical sharing.

Definition 2.7 (Niche Count). The niche count $\text{m}(p, P)$ [535, 1417] of an individual p is the sum its sharing values with all individual in a list P.

$$\forall p \in P \Rightarrow \text{m}(p, P) = \sum_{i=0}^{\text{len}(P)-1} \text{Sh}_\sigma(\text{dist}(p, P_{[i]})) \tag{2.10}$$

The niche count m is always greater than zero, since $p \in P$ and, hence, $\text{Sh}_\sigma(\text{dist}(p, p)) = 1$ is computed and added up at least once. The original sharing approach was developed for single-objective optimization where only one objective function f was subject to maximization. In this case, its value was simply divided by the niche count, punishing solutions in crowded regions [1417]. The goal of sharing was to distribute the population over a number of different peaks in the fitness landscape, with each peak receiving a fraction of the population proportional to its height [959]. The results of dividing the fitness by the niche counts strongly depends on the height differences of the peaks and thus, on the complexity class[16] of f. On $f_1 \in \mathbf{O}(x)$, for instance, the influence of m is much bigger than on a $f_2 \in \mathbf{O}(e^x)$.

By multiplying the niche count m to predetermined fitness values v', we can use this approach for fitness minimization in conjunction with a variety of other different fitness assignment processes, but also inherit its shortcomings:

$$v(p.x) = v'(p.x) * \text{m}(p, Pop)\,,\ v' \equiv \text{assignFitness}(Pop, \text{cmp}_F) \tag{2.11}$$

Sharing was traditionally combined with fitness proportionate, i. e., roulette wheel selection[17]. Oei et al. [1558] have shown that if the sharing function is computed using the parental individuals of the "old" population and then naïvely combined with the more sophisticated tournament selection[18], the resulting behavior of the evolutionary algorithm may be chaotic. They suggested to use the partially filled "new" population to circumvent this problem. The layout of evolutionary algorithms, as defined in this book, bases the fitness computation on the whole set of "new" individuals and assumes that their objective values have already been completely determined. In other words, such issues simply do not exist in multi-objective evolutionary algorithms as introduced here and the chaotic behavior does occur.

[15] See Definition 29.6 on page 537 for more information on the Hamming distance.
[16] See Section 30.1.3 on page 550 for a detailed introduction into complexity and the **O**-notation.
[17] Roulette wheel selection is discussed in Section 2.4.3 on page 124.
[18] You can find an outline of tournament selection in Section 2.4.4 on page 127.

For computing the niche count m, $\mathbf{O}(n^2)$ comparisons are needed. According to Goldberg et al. [827], sampling the population can be sufficient to approximate min order to avoid this quadratic complexity.

2.3.5 Variety Preserving Ranking

Using sharing and the niche counts naïvely leads to more or less unpredictable effects. Of course, it promotes solutions located in sparsely populated niches but how much their fitness will be improved is rather unclear. Using distance measures which are not normalized can lead to strange effects, too. Imagine two objective functions f_1 and f_2. If the values of f_1 span from 0 to 1 for the individuals in the population whereas those of f_2 range from 0 to 10 000, the components of f_1 will most often be negligible in the Euclidian distance of two individuals in the objective space $\mathbb{Y}$. Another problem is that the effect of simple sharing on the pressure into the direction of the Pareto frontier is not obvious either or depends on the sharing approach applied. Some methods simply add a niche count to the Pareto rank, which may cause non-dominated individuals having worse fitness than any others in the population. Other approaches scale the niche count into the interval $[0, 1)$ before adding it which not only ensures that non-dominated individuals have the best fitness but also leave the relation between individuals at different ranks intact, which does not further variety very much.

Variety Preserving Ranking is a fitness assignment approach based on Pareto ranking using prevalence comparators and sharing. We have developed it in order to mitigate all these previously mentioned side effects and balance the evolutionary pressure between optimizing the objective functions and maximizing the variety inside the population. In the following, we will describe the process of Variety Preserving Ranking-based fitness assignment which is defined in Algorithm 2.4.

Before this fitness assignment process can begin, it is required that all individuals with infinite objective values must be removed from the population *Pop*. If such a solution candidate is optimal, i. e., if it has negative infinitely large objectives in a minimization process, for instance, it should receive fitness zero, since fitness is subject to minimization. If the individual is infeasible, on the other hand, its fitness should be set to $\text{len}(Pop) + \sqrt{\text{len}(Pop)} + 1$, which is one larger than every other fitness values that may be assigned by Algorithm 2.4.

In lines 2 to 9, we create a list *ranks* which we use to efficiently compute the Pareto rank of every solution candidate in the population. By the way, the word *prevalence rank* would be more precise in this case, since we use prevalence comparisons as introduced in Section 1.2.4. Therefore, Variety Preserving Ranking is not limited to Pareto optimization but may also incorporate External Decision Makers (Section 1.2.4) or the method of inequalities (Section 1.2.3). The highest rank encountered in the population is stored in the variable *maxRank*. This value may be zero if the population contains only non-prevailed elements. The lowest rank will always be zero since the prevalence comparators cmp_F define order relations which are non-circular by definition.[19]. We will use *maxRank* to determine the maximum penalty for solutions in an overly crowded region of the search space later on.

From line 10 to 18, we determine the maximum and the minimum values that each objective function takes on when applied to the individuals in the population. These values are used to store the inverse of their ranges in the array *rangeScales*, which we will use to scale all distances in each dimension (objective) of the individuals into the interval $[0, 1]$. There are $|F|$ objective functions in F and, hence, the maximum Euclidian distance between two solution candidates in the (scaled) objective space becomes $\sqrt{|F|}$. It occurs if all the distances in the single dimensions are 1.

The most complicated part of the Variety Preserving Ranking algorithm is between line 19 and 33. Here we computed the scaled distance from every individual to each other

[19] In all order relations imposed on finite sets there is always at least one "smallest" element. See Section 27.7.2 on page 463 for more information.

Algorithm 2.4: $v \longleftarrow$ assignFitnessVarietyPreserving(Pop, cmp_F)

Input: Pop: the population
Input: cmp_F: the comparator function
Input: [implicit] F: the set of objective functions
Data: ...: sorry, no space here, we'll discuss this in the text
Output: v: the fitness function

```
1  begin
       /* If needed: Remove all elements with infinite objective values from Pop
          and assign fitness 0 or len(Pop) + sqrt(len(Pop)) + 1 to them. Then compute the
          prevalence ranks.                                                          */
2      ranks ⟵ createList(len(Pop), 0)
3      maxRank ⟵ 0
4      for i ⟵ len(Pop) − 1 down to 0 do
5          for j ⟵ i − 1 down to 0 do
6              k ⟵ cmp_F(Pop[i].x, Pop[j].x)
7              if k < 0 then ranks[j] ⟵ ranks[j] + 1
8              else if k > 0 then ranks[i] ⟵ ranks[i] + 1
9          if ranks[i] > maxRank then maxRank ⟵ ranks[i]
       // determine the ranges of the objectives
10     mins ⟵ createList(|F|, +∞)
11     maxs ⟵ createList(|F|, −∞)
12     foreach p ∈ Pop do
13         for i ⟵ |F| down to 1 do
14             if f_i(p.x) < mins[i−1] then mins[i−1] ⟵ f_i(p.x)
15             if f_i(p.x) > maxs[i−1] then maxs[i−1] ⟵ f_i(p.x)
16     rangeScales ⟵ createList(|F|, 1)
17     for i ⟵ |F| − 1 down to 0 do
18         if maxs[i] > mins[i] then rangeScales[i] ⟵ 1/(maxs[i] − mins[i])
       // Base a sharing value on the scaled Euclidean distance of all elements
19     shares ⟵ createList(len(Pop), 0)
20     minShare ⟵ +∞
21     maxShare ⟵ −∞
22     for i ⟵ len(Pop) − 1 down to 0 do
23         curShare ⟵ shares[i]
24         for j ⟵ i − 1 down to 0 do
25             dist ⟵ 0
26             for k ⟵ |F| down to 1 do
27                 dist ⟵ dist + [(f_k(Pop[i].x) − f_k(Pop[j].x)) * rangeScales[k−1]]^2
28             s ⟵ Sh_exp_{sqrt(|F|),16}(sqrt(dist))
29             curShare ⟵ curShare + s
30             shares[j] ⟵ shares[j] + s
31         shares[i] ⟵ curShare
32         if curShare < minShare then minShare ⟵ curShare
33         if curShare > maxShare then maxShare ⟵ curShare
       // Finally, compute the fitness values
34     scale ⟵ { 1/(maxShare − minShare)  if maxShare > minShare
               { 1                        otherwise
35     for i ⟵ len(Pop) − 1 down to 0 do
36         if ranks[i] > 0 then
37             v(Pop[i].x) ⟵ ranks[i] + sqrt(maxRank) * scale * (shares[i] − minShare)
38         else v(Pop[i].x) ⟵ scale * (shares[i] − minShare)
39 end
```

solution candidate in the objective space and use this distance to aggregate share values (in the array *shares*). Therefore, again two nested loops are needed (lines 22 and 24). The distance components of two individuals $Pop_{[i]}$ and $Pop_{[j]}$ are scaled and summarized in a variable *dist* in line 27. The Euclidian distance between them is $\sqrt{dist}$ which we use to determine a sharing value in 28. We therefore have decided for exponential sharing with power 16 and $\sigma = \sqrt{|F|}$, as introduced in Equation 2.9 on page 115. For every individual, we sum up all the shares (see line 30). While doing so, we also determine the minimum and maximum such total share in the variables *minShare* and *maxShare* in lines 32 and 33.

We will use these variables to scale all sharing values again into the interval $[0, 1]$ (line 34), so the individual in the most crowded region always has a total share of 1 and the most remote individual always has a share of 0. So basically, we now know two things about the individuals in *Pop*:

1. their Pareto ranks, stored in the array *ranks*, giving information about their relative quality according to the objective values and
2. their sharing values, held in *shares*, denoting how densely crowded the area around them is.

With this information, we determine the final fitness values of an individual p as follows: If p is non-prevailed, i. e., its rank is zero, its fitness is its scaled total share (line 38). Otherwise, we multiply the square root of the maximum rank, $\sqrt{maxRank}$, with the scaled share and add it to its rank (line 37). By doing so, we preserve the supremacy of non-prevailed individuals in the population but allow them to compete with each other based on the crowdedness of their location in the objective space. All other solution candidates may degenerate in rank, but at most by the square root of the worst rank.

Example

Let us now apply Variety Preserving Ranking to the examples for Pareto ranking from Section 2.3.3. In Table 2.2, we again list all the solution candidates from Figure 2.4 on page 112, this time with their objective values obtained with f_1 and f_2 corresponding to their coordinates in the diagram. In the third column, you can find the Pareto ranks of the individuals as it has been listed in Table 2.1 on page 113. The columns *share/u* and *share/s* correspond to the total sharing sums of the individuals, unscaled and scaled into $[0, 1]$.

x	f_1	f_2	rank	share/u	share/s	$v(x)$
1	1	7	0	0.71	0.779	0.779
2	2	4	0	0.239	0.246	0.246
3	6	2	0	0.201	0.202	0.202
4	10	1	0	0.022	0	0
5	1	8	1	0.622	0.679	3.446
6	2	7	2	0.906	1	5.606
7	3	5	1	0.531	0.576	3.077
8	2	9	4	0.314	0.33	5.191
9	3	7	4	0.719	0.789	6.845
10	4	6	2	0.592	0.645	4.325
11	5	5	2	0.363	0.386	3.39
12	7	3	1	0.346	0.366	2.321
13	8	4	3	0.217	0.221	3.797
14	7	7	9	0.094	0.081	9.292
15	9	9	13	0.025	0.004	13.01

Table 2.2: An example for Variety Preserving Ranking based on Figure 2.4.

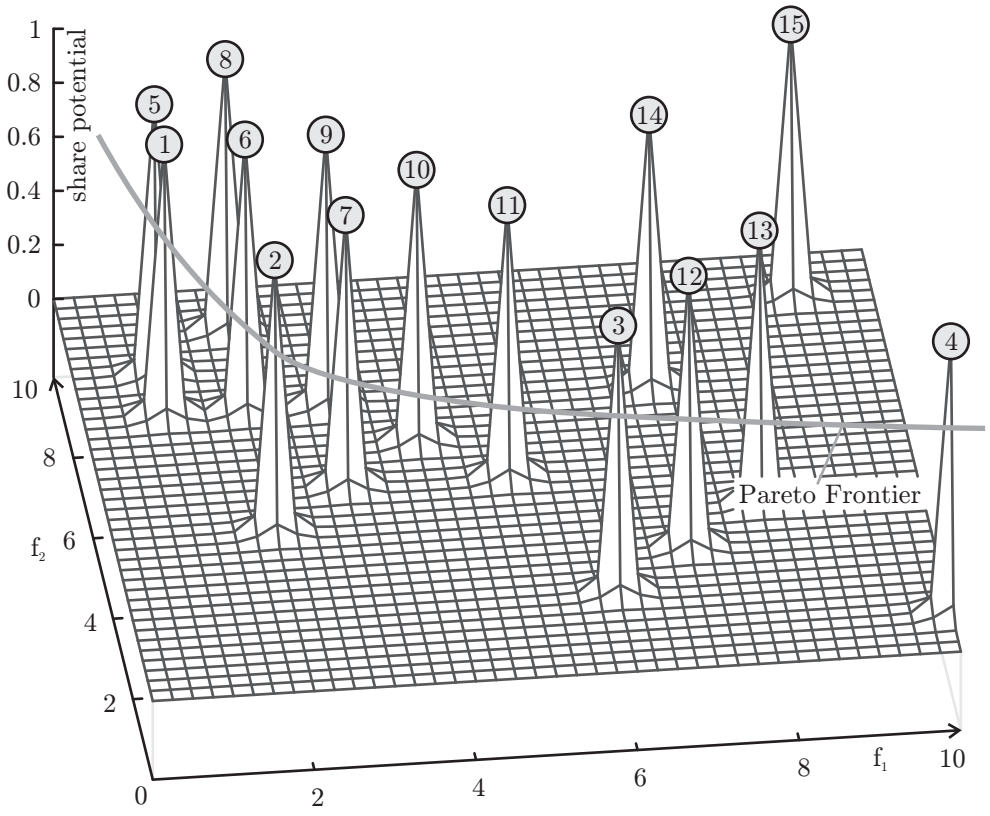


Figure 2.5: The sharing potential in the Variety Preserving Ranking example

But first things first; as already mentioned, we know the Pareto ranks of the solution candidates from Table 2.1, so the next step is to determine the ranges of values the objective functions take on for the example population. These can again easily be found out from Figure 2.4. f_1 spans from 1 to 10, which leads to $rangeScale_{[0]} = 1/9$. $rangeScale_{[1]} = 1/8$ since the maximum of f_2 is 9 and its minimum is 1. With this, we now can compute the (dimensionally scaled) distances amongst the solution candidates in the objective space, the values of $\sqrt{dist}$ in algorithm Algorithm 2.4, as well as the corresponding values of the sharing function $\text{Sh\_exp}_{\sqrt{|F|},16}\left(\sqrt{dist}\right)$. We have noted these in Table 2.3, using the upper triangle of the table for the distances and the lower triangle for the shares.

The value of the sharing function can be imagined as a scalar field, as illustrated in Figure 2.5. In this case, each individual in the population can be considered as an electron that will build an electrical field around it resulting in a potential. If two electrons come close, repulsing forces occur, which is pretty much the same what we want to do with Variety Preserving Ranking. Unlike the electrical field, the power of the sharing potential falls exponentially, resulting in relatively steep spikes in Figure 2.5 which gives proximity and density a heavier influence. Electrons in atoms on planets are limited in their movement by other influences like gravity or nuclear forces, which are often stronger than the electromagnetic force. In Variety Preserving Ranking, the prevalence rank plays this role – as you can see in Table 2.2, its influence on the fitness is often dominant.

By summing up the single sharing potentials for each individual in the example, we obtain the fifth column of Table 2.3, the unscaled share values. Their minimum is around 0.022 and the maximum is 0.94. Therefore, we must subtract 0.022 from each of these values and multiply the result with 1.131. By doing so, we build the column *shares/s*. Finally, we can compute the fitness values $v(x)$ according to lines 38 and 37 in Algorithm 2.4.

Upper triangle: distances. Lower triangle: corresponding share values.															
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**
1		0.391	0.836	1.25	0.125	0.111	0.334	0.274	0.222	0.356	0.51	0.833	0.863	0.667	0.923
2	0.012		0.51	0.965	0.512	0.375	0.167	0.625	0.391	0.334	0.356	0.569	0.667	0.67	0.998
3	7.7E-5	0.003		0.462	0.933	0.767	0.502	0.981	0.708	0.547	0.391	0.167	0.334	0.635	0.936
4	6.1E-7	1.8E-5	0.005		1.329	1.163	0.925	1.338	1.08	0.914	0.747	0.417	0.436	0.821	1.006
5	0.243	0.003	2.6E-5	1.8E-7		0.167	0.436	0.167	0.255	0.417	0.582	0.914	0.925	0.678	0.898
6	0.284	0.014	1.7E-4	1.8E-6	0.151		0.274	0.25	0.111	0.255	0.417	0.747	0.765	0.556	0.817
7	0.023	0.151	0.003	2.9E-5	0.007	0.045		0.512	0.25	0.167	0.222	0.51	0.569	0.51	0.833
8	0.045	0.001	1.5E-5	1.5E-7	0.151	0.059	0.003		0.274	0.436	0.601	0.933	0.914	0.609	0.778
9	0.081	0.012	3.3E-4	4.8E-6	0.056	0.284	0.059	0.045		0.167	0.334	0.669	0.67	0.444	0.712
10	0.018	0.023	0.002	3.2E-5	0.009	0.056	0.151	0.007	0.151		0.167	0.502	0.51	0.356	0.67
11	0.003	0.018	0.012	2.1E-4	0.001	0.009	0.081	0.001	0.023	0.151		0.334	0.356	0.334	0.669
12	8E-5	0.002	0.151	0.009	3.2E-5	2.1E-4	0.003	2.6E-5	0.001	0.003	0.023		0.167	0.5	0.782
13	5.7E-5	0.001	0.023	0.007	2.9E-5	1.7E-4	0.002	3.2E-5	0.001	0.003	0.018	0.151		0.391	0.635
14	0.001	0.001	0.001	9.3E-5	4.6E-4	0.002	0.003	0.001	0.007	0.018	0.023	0.003	0.012		0.334
15	2.9E-5	1.2E-5	2.5E-5	1.1E-5	3.9E-5	9.7E-5	8E-5	1.5E-4	3.1E-4	0.001	0.001	1.4E-4	0.001	0.023	

Table 2.3: The distance and sharing matrix of the example from Table 2.2.

The last column of Table 2.2 lists these results. All non-prevailed individuals have retained a fitness value less than one, lower than those of any other solution candidate in the population. However, amongst these best individuals, solution candidate 4 is strongly preferred, since it is located in a very remote location of the objective space. Individual 1 is the least interesting non-dominated one, because it has the densest neighborhood in Figure 2.4. In this neighborhood, the individuals 5 and 6 with the Pareto ranks 1 and 2 are located. They are strongly penalized by the sharing process and receive the fitness values $v(5) = 3.446$ and $v(6) = 5.606$. In other words, individual 5 becomes less interesting than solution candidate 7 which has a worse Pareto rank. 6 now is even worse than individual 8 which would have a fitness better by two if strict Pareto ranking was applied.

Based on these fitness values, algorithms like Tournament selection (see Section 2.4.2) or fitness proportionate approaches (discussed in Section 2.4.3) will pick elements in a way that preserves the pressure into the direction of the Pareto frontier but also leads to a balanced and sustainable variety in the population. The benefits of this approach have been shown, for instance, in [1650, 2188].

2.3.6 Tournament Fitness Assignment

In tournament fitness assignment, which is a generalization of the q-level binary tournament selection introduced by Weicker [2167], the fitness of each individual is computed by letting it compete q times against r other individuals (with $r = 1$ as default) and counting the number of competitions it loses. For a better understanding of the tournament metaphor see Section 2.4.4 on page 127, where the tournament selection scheme is discussed. Anyway, the number of losses will approximate its Pareto rank, but are a bit more randomized that that. If we would count the number of tournaments won instead of the losses, we would encounter the same problems than in the first idea of Pareto ranking.

TODO add remaining fitness assignment methods

Algorithm 2.5: $v \longleftarrow \text{assignFitnessTournament}_{q,r}(Pop, \text{cmp}_F)$

Input: q: the number of tournaments per individuals
Input: r: the number of other contestants per tournament, normally 1
Input: Pop: the population to assign fitness values to
Input: cmp_F: the comparator function providing the prevalence relation
Data: i, j, k, z: counter variables
Data: b: a Boolean variable being `true` as long as a tournament isn't lost
Data: p: the individual currently examined
Output: v: the fitness function

```
1  begin
2    for i ⟵ len(Pop) − 1 down to 0 do
3      z ⟵ q
4      p ⟵ Pop[i]
5      for j ⟵ q down to 1 do
6        b ⟵ true
7        k ⟵ r
8        while (k > 0) ∧ b do
9          b ⟵ Pop[⌊random_u(0,len(Pop))⌋].x ≻ p.x
10         k ⟵ k − 1
11       if b then z ⟵ z − 1
12     v(p.x) ⟵ z
13   return v
14 end
```

2.4 Selection

2.4.1 Introduction

Definition 2.8 (Selection). In evolutionary algorithms, the selection[20] operation $Mate = \text{select}(Pop, v, ms)$ chooses ms individuals according to their fitness values v from the population Pop and places them into the mating pool $Mate$ [99, 1242, 232, 1431].

$$\begin{aligned} Mate = \text{select}(Pop, v, ms) \Rightarrow\ & \forall p \in Mate \Rightarrow p \in Pop \\ & \forall p \in Pop \Rightarrow p \in \mathbb{G} \times \mathbb{X} \\ & v(p.x) \in \mathbb{R}^+ \ \forall p \in Pop \\ & (\text{len}(Mate) \geq \min\{\text{len}(Pop), ms\}) \wedge (\text{len}(Mate) \leq ms) \end{aligned} \tag{2.12}$$

On the mating pool, the reproduction operations discussed in Section 2.5 on page 137 will subsequently be applied. Selection may behave in a deterministic or in a randomized manner, depending on the algorithm chosen and its application-dependant implementation. Furthermore, elitist evolutionary algorithms may incorporate an archive Arc in the selection process, as sketched in Algorithm 2.2.

Generally, there are two classes of selection algorithms: such *with replacement* (annotated with a subscript $_r$) and such *without replacement* (annotated with a subscript $_w$, see Equation 2.13) [1809]. In a selection algorithm *without* replacement, each individual from the population Pop is taken into consideration for reproduction at most once and therefore

[20] http://en.wikipedia.org/wiki/Selection_%28genetic_algorithm%29 [accessed 2007-07-03]

also will occur in the mating pool *Mate* one time at most. The mating pool returned by algorithms *with* replacement can contain the same individual multiple times. Like in nature, one individual may thus have multiple offspring. Normally, selection algorithms are used in a variant with replacement. One of the reasons therefore is the number of elements to be placed into the mating pool (corresponding to the parameter ms). If $\text{len}(Pop) < ms$, the mating pool returned by a method without replacement contains less than ms individuals since it can at most consist of the whole population.

$$Mate = \text{select}_w(Pop, v, ms) \Rightarrow \text{countOccurences}(p, Mate) = 1 \; \forall p \in Mate \tag{2.13}$$

The selection algorithms have major impact on the performance of evolutionary algorithms. Their behavior has thus been subject to several detailed studies, conducted by, for instance, Goldberg and Deb [823], Blickle and Thiele [232], and Zhong et al. [2318], just to name a few.

Usually, fitness assignment processes are carried out before selection and the selection algorithms base their decisions solely on the fitness v of the individuals. It is possible to rely on the prevalence relation, i. e., to write $\text{select}(Pop, \text{cmp}_F, ms)$ instead of $\text{select}(Pop, v, ms)$, thus saving the costs of the fitness assignment process. However, this will lead to the same problems that occurred in the first approach of prevalence-proportional fitness assignment (see Section 2.3.3 on page 112) and we will therefore not discuss such techniques in this book.

Many selection algorithms only work with scalar fitness and thus need to rely on a fitness assignment process in multi-objective optimization. Selection algorithms can be chained – the resulting mating pool of the first selection may then be used as input for the next one, maybe even with a secondary fitness assignment process in between. In some applications, an environmental selection that reduces the number of individuals is performed first and then a mating selection follows which extracts the individuals which should be used for reproduction.

Visualization

In the following sections, we will discuss multiple selection algorithms. In order to ease understanding them, we will visualize the expected number of times $S(p)$ that an individual p will reach the mating pool *Mate* for some of the algorithms.

$$S(p) = E[\text{countOccurences}(p, Mate)] \tag{2.14}$$

Therefore, we will use the special case where we have a population Pop of $\text{len}(Pop) = 1000$ individuals, $p_0..p_{999}$ and also a target mating pool size $ms = 1000$. Each individual p_i has the fitness value $v(p_i.x)$, and fitness is subject to minimization. For this fitness, we consider two cases:

1. As sketched in Fig. 2.6.a, the individual p_i has fitness i, i. e., $v_1(p_0.x) = 0, v_1(p_1.x) = 1, \ldots, v_1(p_{999}.x) = 999$.
2. Individual p_i has fitness $(i+1)^3$, i. e., $v_2(p_0.x) = 1, v_2(p_1.x) = 3, \ldots, v_2(p_{999}.x) = 1\,000\,000\,000$, as illustrated in Fig. 2.6.b.

2.4.2 Truncation Selection

Truncation selection[21], also called deterministic selection or threshold selection, returns the $k < ms$ best elements from the list Pop. These elements are copied as often as needed until the mating pool size ms reached. For k, normally values like $\text{len}(Pop)/2$ or $\text{len}(Pop)/3$ are

[21] http://en.wikipedia.org/wiki/Truncation_selection [accessed 2007-07-03]

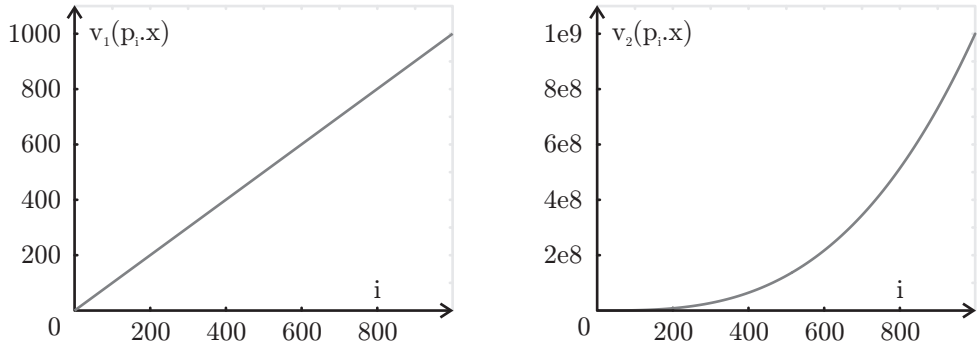


Fig. 2.6.a: Case 1: $v(p_i.x) = i$

Fig. 2.6.b: Case 2: $v(p_i.x) = (i+1)^3$

Figure 2.6: The two example fitness cases.

used. Algorithm 2.6 realizes this scheme by first sorting the population in ascending order according to the fitness v. Then, it iterates from 0 to ms and inserts only the elements with indices from 0 to $k-1$ into the mating pool.

Algorithm 2.6: $Mate \longleftarrow \text{truncationSelect}_k(Pop, v, ms)$

Input: Pop: the list of individuals to select from
Input: v: the fitness values
Input: ms: the number of individuals to be placed into the mating pool $Mate$
Input: k: cut-off value
Data: i: counter variables
Output: $Mate$: the winners of the tournaments which now form the mating pool

```
1 begin
2     Mate ⟵ ()
3     k ⟵ min{k, len(Pop)}
4     Pop ⟵ sortList_a(Pop, v)
5     for i ⟵ 0 up to ms − 1 do
6         Mate ⟵ addListItem(Mate, Pop[i mod k])
7     return Mate
8 end
```

Truncation selection is usually used in Evolution Strategies with $(\mu+\lambda)$ and (μ, λ) strategies. In general evolutionary algorithms, it should be combined with a fitness assignment process that incorporates diversity information in order to prevent premature convergence. Recently, Lässig et al. [1260] have proved that truncation selection is the optimal selection strategy for crossover, provided that the right value of k is used. In practical applications, this value is normally not known.

In Figure 2.7, we sketch the expected number of offspring for the individuals from our examples specified in Section 2.4.1. In this selection scheme, the diagram will look exactly the same regardless whether we use fitness configuration 1 or 2, since it is solely based on the order of individuals and not on the numerical relation of their fitness. If we set $k = ms = \text{len}(Pop)$, each individual will have one offspring in average. If $k = \frac{1}{2}ms$, the top-50% individuals will have two offspring and the others none. For $k = \frac{1}{10}ms$, only the best 100 from the 1000 solution candidates will reach the mating pool but reproduce 10 times in average.

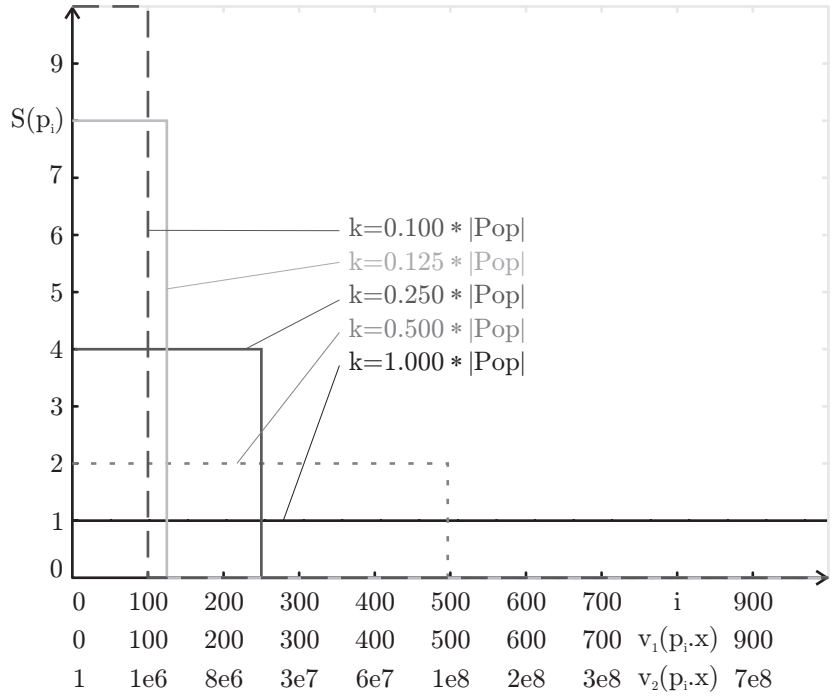


Figure 2.7: The number of expected offspring in truncation selection.

2.4.3 Fitness Proportionate Selection

Fitness proportionate selection[22] has already been applied in the original genetic algorithms as introduced by Holland [940] and therefore is one of the oldest selection schemes. In fitness proportionate selection, the probability $P(p_1)$ of an individual $p_1 \in Pop$ to enter the mating pool is proportional to its fitness $v(p.x)$ (subject to maximization) compared to the sum of the fitness of all individuals. This relation in its original form is defined in Equation 2.15 below.

$$P(p_1) = \frac{v(p_1.x)}{\sum_{\forall p_2 \in Pop} v(p_2.x)} \tag{2.15}$$

There exists a variety of approaches which realize such probability distributions [823], like stochastic remainder selection (Brindle [289], Booker [248]) and stochastic universal selection (Baker [121], Greffenstette and Baker [858]). The most commonly known method is the Monte Carlo *roulette wheel selection* by De Jong [512], where we imagine the individuals of a population to be placed on a roulette[23] wheel as sketched in Fig. 2.8.a. The size of the area on the wheel standing for a solution candidate is proportional to its fitness. The wheel is spun, and the individual where it stops is placed into the mating pool *Mate*. This procedure is repeated until ms individuals have been selected.

In the context of this book, fitness is subject to minimization. Here, higher fitness values $v(p.x)$ indicate unfit solution candidates $p.x$ whereas lower fitness denotes high utility. Furthermore, the fitness values are normalized into a range of $[0, sum]$, because otherwise, fitness proportionate selection will handle the set of fitness values $\{0, 1, 2\}$ in a different way than $\{10, 11, 12\}$. Equation 2.19 defines the framework for such a (normalized) fitness proportionate selection "rouletteWheelSelect". It is illustrated exemplarily in Fig. 2.8.b and realized in Algorithm 2.7 as a variant with and in Algorithm 2.8 without replacement. Amongst

[22] http://en.wikipedia.org/wiki/Fitness_proportionate_selection [accessed 2008-03-19]
[23] http://en.wikipedia.org/wiki/Roulette [accessed 2008-03-20]

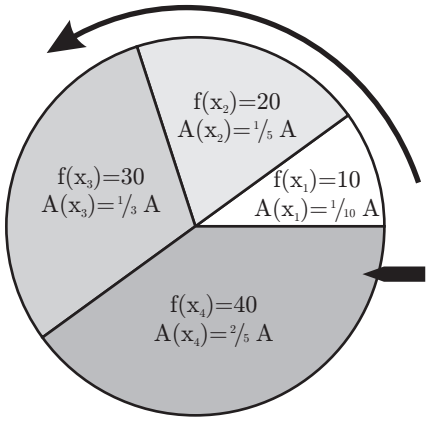


Fig. 2.8.a: Example for fitness maximization.

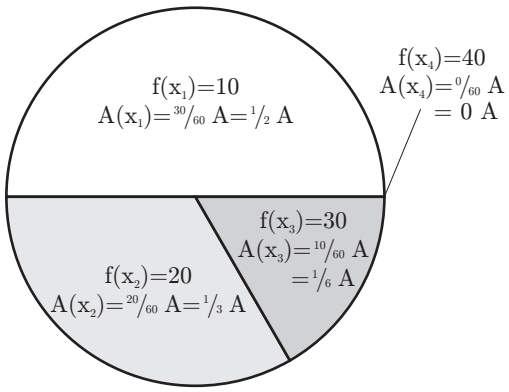


Fig. 2.8.b: Example for normalized fitness minimization.

Figure 2.8: Examples for the idea of roulette wheel selection.

others, Whitley [2211] points out that even fitness normalization as performed here cannot overcome the drawbacks of fitness proportional selection methods.

$$minV = \min\{v(p.x) \ \forall p \in Pop\} \tag{2.16}$$

$$maxV = \max\{v(p.x) \ \forall p \in Pop\} \tag{2.17}$$

$$normV(p.x) = \frac{maxV - v(p.x)}{maxV - minV} \tag{2.18}$$

$$P(p_1) = \frac{normV(p_1.x)}{\sum_{\forall p_2 \in Pop} normV(p_2.x)} \tag{2.19}$$

But what are the drawbacks of fitness proportionate selection methods? Let us therefore visualize the expected results of roulette wheel selection applied to the special cases stated in Section 2.4.1. Figure 2.9 illustrates the number of expected occurrences $S(p_i)$ of an individual p_i if roulette wheel selection was applied. Since $ms = 1000$, we draw one thousand times a single individual from the population Pop. Each single choice is based on the proportion of the individual fitness in the total fitness of all individuals, as defined in Equation 2.15 and Equation 2.19. Thus, in scenario 1 with the fitness sum $\frac{999*998}{2} = 498501$, the relation $S(p_i) = ms * \frac{i}{498501}$ holds for fitness maximization and $S(p_i) = ms\frac{999-i}{498501}$ for minimization. As result (sketched in Fig. 2.9.a), the fittest individuals produce (on average) two offspring, whereas the worst solution candidates will always vanish in this example. For the 2^{nd} scenario with $v_2(p_i.x) = (i+1)^3$, the total fitness sum is approximately $2.51 \cdot 10^{11}$ and $S(p_i) = ms\frac{(i+1)^3}{2.52 \cdot 10^{11}}$ holds for maximization. The resulting expected values depicted in Fig. 2.9.b are significantly different from those in Fig. 2.9.a. The meaning of this is that the design of the objective functions (or the fitness assignment process) has a much stronger influence on the convergence behavior of the evolutionary algorithm. This selection method only works well if the fitness of an individual is indeed something like a proportional measure for the probability that it will produce better offspring.

Thus, roulette wheel selection has a bad performance compared to other schemes like tournament selection [823, 231] or ranking selection [823, 232]. It is mainly included here for the sake of completeness and because it is easy to understand and suitable for educational purposes.

Algorithm 2.7: $Mate \longleftarrow \text{rouletteWheelSelect}_r(Pop, v, ms)$

Input: Pop: the list of individuals to select from
Input: v: the fitness values
Input: ms: the number of individuals to be placed into the mating pool $Mate$
Data: i: a counter variable
Data: a: a temporary store for a numerical value
Data: A: the array of fitness values
Data: min, max, sum: the minimum, maximum, and sum of the fitness values
Output: $Mate$: the mating pool

```
1  begin
2      A ⟵ createList(len(Pop), 0)
3      min ⟵ ∞
4      max ⟵ −∞
5      for i ⟵ 0 up to len(Pop) − 1 do
6          a ⟵ v(Pop[i].x)
7          A[i] ⟵ a
8          if a < min then min ⟵ a
9          if a > max then max ⟵ a
10     if max = min then
11         max ⟵ max + 1
12         min ⟵ min − 1
13     sum ⟵ 0
14     for i ⟵ 0 up to len(Pop) − 1 do
15         sum ⟵ (max − A[i]) / (max − min)
16         A[i] ⟵ sum
17     for i ⟵ 0 up to ms − 1 do
18         a ⟵ searchItem_as(random_u(0, sum), A)
19         if a < 0 then a ⟵ −a − 1
20         Mate ⟵ addListItem(Mate, Pop[a])
21     return Mate
22 end
```

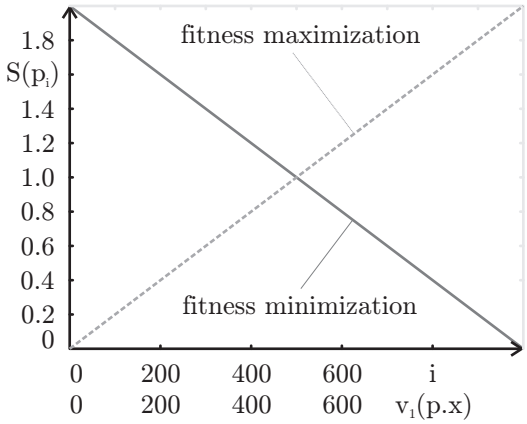


Fig. 2.9.a: $v_1(p_i.x) = i$

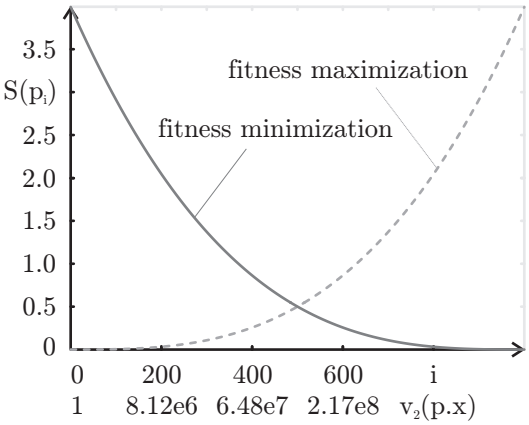


Fig. 2.9.b: $v_1(p_i.x) = (i+1)^3$

Figure 2.9: The number of expected offspring in roulette wheel selection.

Algorithm 2.8: $Mate \longleftarrow \text{rouletteWheelSelect}_w(Pop, v, ms)$

Input: Pop: the list of individuals to select from
Input: v: the fitness values
Input: ms: the number of individuals to be placed into the mating pool $Mate$
Data: i: a counter variable
Data: a, b: temporary stores for numerical values
Data: A: the array of fitness values
Data: min, max, sum: the minimum, maximum, and sum of the fitness values
Output: $Mate$: the mating pool

```
1  begin
2      A ⟵ createList(len(Pop), 0)
3      min ⟵ ∞
4      max ⟵ −∞
5      for i ⟵ 0 up to len(Pop) − 1 do
6          a ⟵ v(Pop[i], x)
7          A[i] ⟵ a
8          if a < min then min ⟵ a
9          if a > max then max ⟵ a
10     if max = min then
11         max ⟵ max + 1
12         min ⟵ min − 1
13     sum ⟵ 0
14     for i ⟵ 0 up to len(Pop) − 1 do
15         sum ⟵ (max − A[i]) / (max − min)
16         A[i] ⟵ sum
17     for i ⟵ 0 up to min{ms, len(Pop)} − 1 do
18         a ⟵ searchItem_as(random_u(0, sum), A)
19         if a < 0 then a ⟵ −a − 1
20         if a = 0 then b ⟵ 0
21         else b ⟵ A[a−1]
22         b ⟵ A[a] − b
23         for j ⟵ a + 1 up to len(A) − 1 do
24             A[j] ⟵ A[j] − b
25         sum ⟵ sum − b
26         Mate ⟵ addListItem(Mate, Pop[a])
27         Pop ⟵ deleteListItem(Pop, a)
28         A ⟵ deleteListItem(A, a)
29     return Mate
30 end
```

2.4.4 Tournament Selection

Tournament selection[24], proposed by Wetzel [2198] and studied by Brindle [289], is one of the most popular and effective selection schemes. Its features are well-known and have been analyzed by a variety of researchers such as Blickle and Thiele [231, 232], Miller and Goldberg [1416], Lee et al. [1269], Sastry and Goldberg [1809], and Oei et al. [1558]. In tournament selection, k elements are picked from the population Pop and compared with each other in a tournament. The winner of this competition will then enter mating pool $Mate$. Although being a simple selection strategy, it is very powerful and therefore used in many practical applications [55, 316, 1403, 46].

As example, consider a tournament selection (with replacement) with a tournament size of two [2208]. For each single tournament, the contestants are chosen randomly according to

[24] http://en.wikipedia.org/wiki/Tournament_selection [accessed 2007-07-03]

a uniform distribution and the winners will be allowed to enter the mating pool. If we assume that the mating pool will contain about as same as many individuals as the population, each individual will, on average, participate in two tournaments. The best solution candidate of the population will win all the contests it takes part in and thus, again on average, contributes approximately two copies to the mating pool. The median individual of the population is better than 50% of its challengers but will also loose against 50%. Therefore, it will enter the mating pool roughly one time on average. The worst individual in the population will lose all its challenges to other solution candidates and can only score even if competing against itself, which will happen with probability $(1/ms)^2$. It will not be able to reproduce in the average case because $ms * (1/ms)^2 = 1/ms < 1 \; \forall ms > 1$.

For visualization purposes, let us go back to our examples from Section 2.4.1 with a population of 1000 individuals $p_0..p_{999}$ and $ms = 1000$. Again, we assume that each individual has an unique fitness value of $v_1(p_i.x) = i$ or $v_2(p_i.x) = (i+1)^3$, respectively. If we apply tournament selection with replacement in this special scenario, the expected number of occurrences $S(p_i)$ of an individual p_i in the mating pool can be computed according to Blickle and Thiele [232] as

$$S(p_i) = ms * \left(\left(\frac{1000 - i}{1000} \right)^k - \left(\frac{1000 - i - 1}{1000} \right)^k \right) \tag{2.20}$$

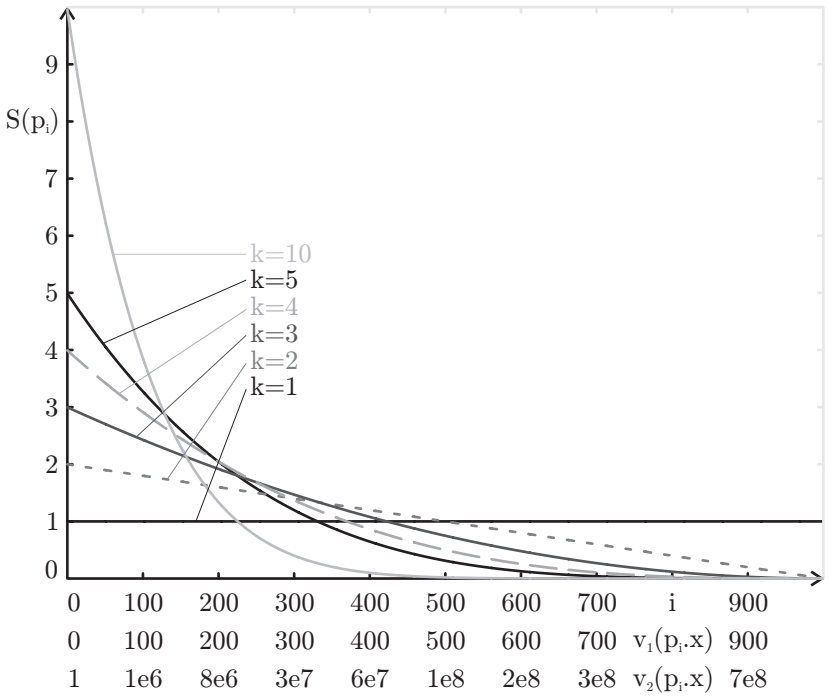


Figure 2.10: The number of expected offspring in tournament selection.

The absolute values of the fitness play no role. The only thing that matters is whether or not the fitness of one individual is higher as the fitness of another one, not fitness difference itself. The expected numbers of offspring for the two example cases 1 and 2 from Section 2.4.1 are the same. Tournament selection thus gets rid of the problems of fitness proportionate methods. Figure 2.10 depicts these numbers for different tournament sizes $k = \{1, 2, 3, 4, 5, 10\}$. If $k = 1$, tournament selection degenerates to randomly picking individuals and each solution candidate will occur one time in the mating pool on average. With

rising k, the selection pressure increases: individuals with good fitness values create more and more offspring whereas the chance of worse solution candidates to reproduce decreases.

Tournament selection with replacement (TSR) is presented in Algorithm 2.9. Tournament selection without replacement (TSoR) [1269, 18] can be defined in two forms. In the first variant specified as Algorithm 2.10, a solution candidate cannot compete against itself. This method is defined in. In Algorithm 2.11, on the other hand, an individual may enter the mating pool at most once.

Algorithm 2.9: $Mate \longleftarrow \text{tournamentSelect}_{r,k}(Pop, v, ms)$

Input: Pop: the list of individuals to select from
Input: v: the fitness values
Input: ms: the number of individuals to be placed into the mating pool $Mate$
Input: [implicit] k: the tournament size
Data: a: the index of the tournament winner
Data: i, j: counter variables
Output: $Mate$: the winners of the tournaments which now form the mating pool

```
1  begin
2      Mate ⟵ ()
3      Pop ⟵ sortList_a(Pop, v)
4      for i ⟵ 0 up to ms − 1 do
5          a ⟵ ⌊random_u(0, len(Pop))⌋
6          for j ⟵ 1 up to k − 1 do
7              a ⟵ min{a, ⌊random_u(0, len(Pop))⌋}
8          Mate ⟵ addListItem(Mate, Pop[a])
9      return Mate
10 end
```

Algorithm 2.10: $Mate \longleftarrow \text{tournamentSelect}_{w1,k}(Pop, v, ms)$

Input: Pop: the list of individuals to select from
Input: v: the fitness values
Input: ms: the number of individuals to be placed into the mating pool $Mate$
Input: [implicit] k: the tournament size
Data: a: the index of the tournament winner
Data: i, j: counter variables
Output: $Mate$: the winners of the tournaments which now form the mating pool

```
1  begin
2      Mate ⟵ ()
3      Pop ⟵ sortList_a(Pop, v)
4      for i ⟵ 0 up to min{len(Pop), ms} − 1 do
5          a ⟵ ⌊random_u(0, len(Pop))⌋
6          for j ⟵ 1 up to min{len(Pop), k} − 1 do
7              a ⟵ min{a, ⌊random_u(0, len(Pop))⌋}
8          Mate ⟵ addListItem(Mate, Pop[a])
9          Pop ⟵ deleteListItem(Pop, a)
10     return Mate
11 end
```

The algorithms specified here should more precisely be entitled as *deterministic* tournament selection algorithms since the winner of the k contestants that take part in each

Algorithm 2.11: $Mate \longleftarrow \text{tournamentSelect}_{w2,k}(Pop, v, ms)$

Input: Pop: the list of individuals to select from
Input: v: the fitness values
Input: ms: the number of individuals to be placed into the mating pool $Mate$
Input: [implicit] k: the tournament size
Data: A: the list of contestants per tournament
Data: a: the tournament winner
Data: i, j: counter variables
Output: $Mate$: the winners of the tournaments which now form the mating pool

```
1  begin
2      Mate ⟵ ()
3      Pop ⟵ sortList_a(Pop, v)
4      for i ⟵ 0 up to ms − 1 do
5          A ⟵ ()
6          for j ⟵ 1 up to min{k, len(Pop)} do
7              repeat
8                  a ⟵ ⌊random_u(0, len(Pop))⌋
9              until searchItem_u(a, A) < 0
10             A ⟵ addListItem(A, a)
11         a ⟵ min A
12         Mate ⟵ addListItem(Mate, Pop_[a])
13     return Mate
14 end
```

tournament enters the mating pool. In the non-deterministic variant this is not necessarily the case. There, a probability p is defined. The best individual in the tournament is selected with probability p, the second best with probability $p(1-p)$, the third best with probability $p(1-p)^2$ and so on. The ith best individual in a tournament enters the mating pool with probability $p(1-p)^i$. Algorithm 2.12 on the facing page realizes this behavior for a tournament selection with replacement. Notice that it becomes equivalent to Algorithm 2.9 on the previous page if p is set to 1. Besides the algorithms discussed here, a set of additional tournament-based selection methods has been introduced by Lee et al. [1269].

Algorithm 2.12: $Mate \longleftarrow \text{tournamentSelect}^{p}_{r,k}(Pop, v, ms)$

Input: Pop: the list of individuals to select from
Input: v: the fitness values
Input: ms: the number of individuals to be placed into the mating pool $Mate$
Input: [implicit] p: the selection probability, $p \in [0, 1]$
Input: [implicit] k: the tournament size
Data: A: the set of tournament contestants
Data: i, j: counter variables
Output: $Mate$: the winners of the tournaments which now form the mating pool

```
1  begin
2      Mate ⟵ ()
3      Pop ⟵ sortList_a(Pop, v)
4      for i ⟵ 0 up to ms − 1 do
5          A ⟵ ()
6          for j ⟵ 0 up to k − 1 do
7              A ⟵ addListItem(A, ⌊random_u(0, len(Pop))⌋)
8          A ⟵ sortList_a(A, cmp(a_1, a_2) ≡ (a_1 − a_2))
9          for j ⟵ 0 up to len(A) − 1 do
10             if (random_u() ≤ p) ∨ (j ≥ len(A) − 1) then
11                 Mate ⟵ addListItem(Mate, Pop[A[j]])
12                 j ⟵ ∞
13     return Mate
14 end
```

2.4.5 Ordered Selection

Ordered selection is another approach for circumventing the problems of fitness proportionate selection methods. Here, the probability of an individual to be selected is proportional to (a power of) its position (rank) in the sorted list of all individuals in the population. The implicit parameter $k \in \mathbb{R}^+$ of the ordered selection algorithm determines the selection pressure. It equals to the number of expected offspring of the best individual and is thus much similar to the parameter k of tournament selection. The bigger k gets, the higher is the probability that individuals which are non-prevailed i. e., have good objective values will be selected.

Algorithm 2.13 demonstrates how ordered selection with replacement works and the variant without replacement is described in Algorithm 2.14. Basically, it first converts the parameter k to a power q to which the uniformly drawn random numbers are raised that are used for indexing the sorted individual list. This can be achieved with Equation 2.21.

$$q = \frac{1}{1 - \frac{\log k}{\log ms}} \tag{2.21}$$

Figure 2.11 illustrates the expected offspring in the application of ordered selection with $k \in \{1, 2, 3, 4, 5\}$. Like tournament selection, a value of $k = 1$ leads degenerates the evolutionary algorithm to a parallel random walk. Another close similarity to tournament selection occurs when comparing the exact formulas computing the expected offspring for our examples:

$$S(p_i) = ms * \left(\left(\frac{i+1}{1000}\right)^q - \left(\frac{i}{1000}\right)^q\right) \tag{2.22}$$

Equation 2.22 looks pretty much like Equation 2.20. The differences between the two selection methods become obvious when comparing the diagrams Figure 2.11 and Figure 2.10 which both are independent of the actual fitness values. Tournament selection creates many

Algorithm 2.13: $Mate \longleftarrow \text{orderedSelect}_r^p(Pop, v, ms)$

Input: Pop: the list of individuals to select from
Input: v: the fitness values
Input: ms: the number of individuals to be placed into the mating pool $Mate$
Input: [implicit] k: the parameter of the ordering selection
Data: q: the power value to be used for ordering
Data: i: a counter variable
Output: $Mate$: the mating pool

```
1 begin
2     q ⟵ 1 / (1 − log k / log ms)
3     Mate ⟵ ()
4     Pop ⟵ sortList_a(Pop, v)
5     for i ⟵ 0 up to ms − 1 do
6         Mate ⟵ addListItem(Mate, Pop[⌊random_u()^p * len(Pop)⌋])
7     return Mate
8 end
```

Algorithm 2.14: $Mate \longleftarrow \text{orderedSelect}_w^p(Pop, v, ms)$

Input: Pop: the list of individuals to select from
Input: v: the fitness values
Input: ms: the number of individuals to be placed into the mating pool $Mate$
Input: [implicit] k: the parameter of the ordering selection
Data: q: the power value to be used for ordering
Data: i, j: counter variables
Output: $Mate$: the mating pool

```
1  begin
2      q ⟵ 1 / (1 − log k / log ms)
3      Mate ⟵ ()
4      Pop ⟵ sortList_a(Pop, v)
5      for i ⟵ 0 up to min{ms, len(Pop)} − 1 do
6          j ⟵ ⌊random_u()^p * len(Pop)⌋
7          Mate ⟵ addListItem(Mate, Pop[j])
8          Pop ⟵ deleteListItem(Pop, j)
9      return Mate
10 end
```

copies of the better fraction of the population and almost none of the others. Ordered selection focuses on an even smaller group of the fittest individuals but also even the worst solution candidates still have a survival probability not too far from one. In other words, while tournament selection reproduces a larger group of good individuals and kills most of the others, ordered selection assigns very high fertility to very few individuals but preservers also the less fitter ones.

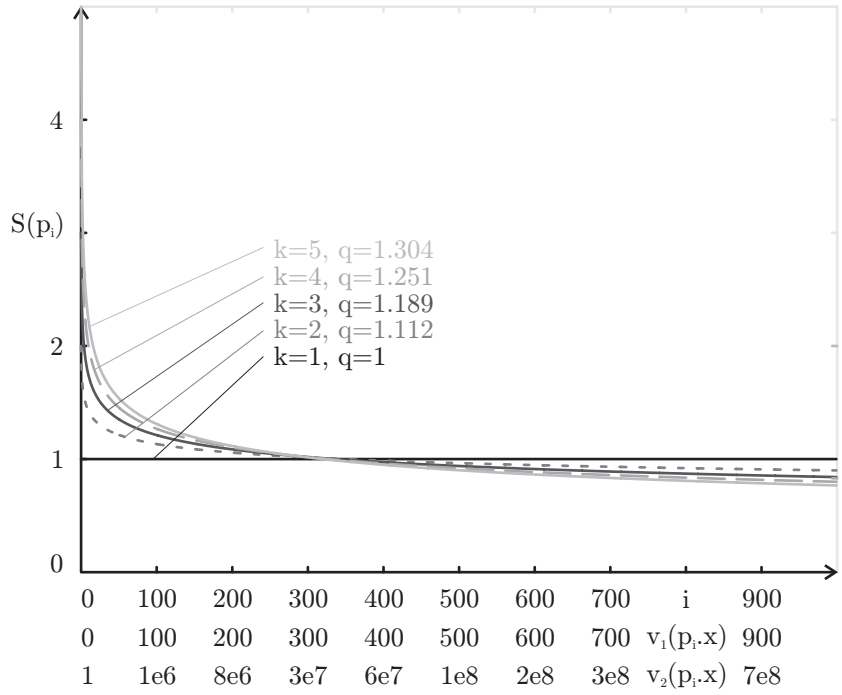


Figure 2.11: The number of expected offspring in ordered selection.

2.4.6 Ranking Selection

Ranking selection, introduced by Baker [120] and more thoroughly discussed by Whitley [2211], Blickle and Thiele [232, 230], and Goldberg and Deb [823] is another approach for circumventing the problems of fitness proportionate selection methods. In ranking selection [120, 2211, 858], the probability of an individual to be selected is proportional to its position (rank) in the sorted list of all individuals in the population. Using the rank smoothes out larger differences of the objective values and emphasizes small ones. Generally, we can the conventional ranking selection method as the application of a fitness assignment process setting the rank as fitness (which can be achieved with Pareto ranking) and a subsequent fitness proportional selection.

2.4.7 VEGA Selection

The *Vector Evaluated Genetic Algorithm* by Schaffer [1821, 1822] applies a special selection algorithm which does not incorporate any preceding fitness assignment process but works on the objective values directly. For each of the objective functions $f_i \in F$, it selects a subset of the mating pool *Mate* of the size $^{ms}/_{|F|}$. Therefore it applies fitness proportionate selection which is based on f_i instead of a fitness assignment "assignFitness". The mating pool is then a mixture of these sub-selections. Richardson et al. [1728] show in [1820] that this selection scheme is approximately the same as if computing a weighted sum of the fitness values. As pointed out by Fonseca and Fleming [714], in the general case, this selection method will sample non-prevailed solution candidates at different frequencies. Schaffer also anticipated that the population of his GA may split into different species, each particularly strong in one objective, if the Pareto frontier is concave.

Algorithm 2.15: $Mate \longleftarrow \text{vegaSelect}(Pop, F, ms)$

Input: Pop: the list of individuals to select from
Input: F: the objective functions
Input: ms: the number of individuals to be placed into the mating pool $Mate$
Data: i: a counter variable
Data: j: the size of the current subset of the mating pool
Data: A: a temporary mating pool
Output: $Mate$: the individuals selected

```
1 begin
2     Mate ⟵ ()
3     for i ⟵ 1 up to |F| do
4         j ⟵ ms/|F|
5         if i = 1 then j ⟵ j + ms mod |F|
6         A ⟵ rouletteWheelSelect_r(Pop, v ≡ f_i, j)
7         Mate ⟵ appendList(Mate, A)
8     return Mate
9 end
```

2.4.8 Clearing and Simple Convergence Prevention (SCP)

In our experiments (especially in Genetic Programming and problems with discrete objective functions) we often use a very simple mechanism to prevent premature convergence (see Section 1.4.2) which we outline in Algorithm 2.17. In our opinion, this *SCP* method is neither a fitness nor a selection algorithm, but we think it fits best into this section.

The idea is simple: the more similar individuals we have in the population, the more likely are we converged. We do not know whether we have converged to a global optimum or to a local one. If we got stuck at a local optimum, we should maybe limit the fraction of the population which resides at this spot. In case we have found the global optimum, this approach does not hurt, because in the end, one single point on this optimum suffices.

Clearing

The first one to apply such an explicit limitation method was Pétrowski [1638, 1639] whose *clearing* approach is applied in each generation and works as specified in Algorithm 2.16 where fitness is subject to minimization. Basically, clearing divides the population of an EA into several sub-populations according to a distance measure dist applied in the genotypic ($\mathbb{G}$) or phenotypic space ($\mathbb{X}$) in each generation. The individuals of each sub-population have at most the distance σ to the fittest individual in this niche. Then, the fitness of all but the k best individuals in such a sub-population is set to the worst possible value. This effectively prevents that a niche can get too crowded. Sareni and Krähenbühl [1801] showed that this method is very promising. Singh and Deb [1892] suggest a modified clearing approach which shifts individuals that would be cleared farther away and reevaluates their fitness.

SCP

We modified this approach in two respects: We measure similarity not in form of a distance in $\mathbb{G}$ or $\mathbb{X}$, but in the objective space $\mathbb{Y} \subseteq \mathbb{R}^{|F|}$. All individuals are compared with each other. If two have exactly the same objective values[25], one of them is thrown away with

[25] The *exactly-the-same*-criterion makes sense in combinatorial optimization and many Genetic Programming problems but may easily be replaced with a limit imposed on the Euclidian distance in real-valued optimization problems, for instance.

Algorithm 2.16: $Pop' \longleftarrow \text{clearing}(Pop, \sigma, k)$

Input: Pop: the list of individuals to apply clearing to
Input: σ: the clearing radius
Input: k: the nieche capacity
Input: [implicit] v: the fitness values
Input: [implicit] dist: a distance measure in the genome or phenome
Data: n: the current number of winners
Data: i, j: counter variables
Output: Pop': the pruned population

```
1  begin
2      Pop' ⟵ sortList_a(Pop, v)
3      for i ⟵ 0 up to len(Pop') − 1 do
4          if v(Pop'[i].x) < ∞ then
5              n ⟵ 1
6              for j ⟵ i + 1 up to len(Pop') − 1 do
7                  if (v(Pop'[j].x) < ∞) ∧ (dist(Pop'[i], Pop'[j]) < σ) then
8                      if n < k then n ⟵ n + 1
9                      else v(Pop'[j].x) ⟵ ∞
10 end
```

probability[26] $cp \in [0, 1]$ and does not take part in any further comparisons. This way, we weed out similar individuals without making any assumptions about $\mathbb{G}$ or $\mathbb{X}$ and make room in the population and mating pool for a wider diversity of solution candidates. For $cp = 0$, this prevention mechanism is turned off, for $cp = 1$, all remaining individuals will have different objective values.

Although this approach is very simple, the results of our experiments were often significantly better with this convergence prevention method turned on than without it [1650, 2188]. Additionally, in none of our experiments, the outcomes were influenced negatively by this filter, which makes it even more robust than other methods for convergence prevention like sharing or variety preserving. Algorithm 2.17, which has to be applied *after* the evaluation of the objective values of the individuals in the population and *before* any fitness assignment or selection takes place, specifies how our simple mechanism works.

If an individual p occurs n times in the population or if there are n individuals with exactly the same objective values, Algorithm 2.17 cuts down the expected number of their occurrences $S(p)$ to

$$S(p) = \sum_{i=1}^{n} (1 - cp)^{i-1} = \sum_{i=0}^{n-1} (1 - cp)^{i} = \frac{(1 - cp)^n - 1}{-cp} = \frac{1 - (1 - cp)^n}{cp} \tag{2.23}$$

In Figure 2.12, we sketch the expected number of remaining instances of the individual p after this pruning process if it occurred n times in the population before Algorithm 2.17 was applied.

From Equation 2.23 follows that even a population of infinite size which has fully converged to one single value will probably not contain more than $\frac{1}{cp}$ copies of this individual after the simple convergence prevention has been applied. This threshold is also visible in Figure 2.12.

$$\lim_{n \to \infty} S(p) = \lim_{n \to \infty} \frac{1 - (1 - cp)^n}{cp} = \frac{1 - 0}{cp} = \frac{1}{cp} \tag{2.24}$$

In Pétrowski's clearing approach [1638], the maximum number of individuals which can survive in a niche was a fixed constant k and, if less than k individuals resided in a niche,

[26] instead of defining a fixed threshold k

Algorithm 2.17: $Pop' \longleftarrow$ convergencePreventionSCP(Pop, cp)

Input: Pop: the list of individuals to apply convergence prevention to
Input: cp: the convergence prevention probability, $cp \in [0, 1]$
Input: [implicit] F: the set of objective functions
Data: i, j: counter variables
Data: p: the individual checked in this generation
Output: Pop': the pruned population

```
1  begin
2      Pop' ⟵ ()
3      for i ⟵ 0 up to len(Pop) − 1 do
4          p ⟵ Pop[i]
5          for j ⟵ len(Pop') − 1 down to 0 do
6              if f(p.x) = f(Pop'[j].x) ∀f ∈ F then
7                  if random_u() < cp then
8                      Pop' ⟵ deleteListItem(Pop', j)
9          Pop' ⟵ addListItem(Pop', p)
10     return Pop'
11 end
```

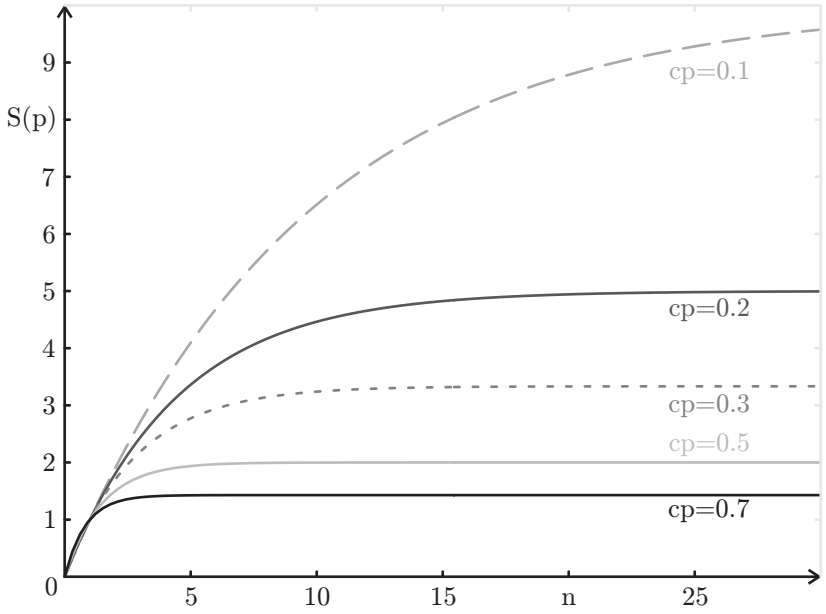


Figure 2.12: The expected numbers of occurences for different values of n and cp.

none of them would be affected. Different from that, an expected value of the number of individuals allowed in a niche is specified with the probability cp and may be both, exceeded or undercut. Another difference of the approaches arises from the space in which the distance is computed.

Discussion

Whereas clearing prevents the EA from concentrating too much on a certain area in the search or problem space, SCP stops it from keeping too many individuals with equal utility. The former approach works against premature convergence to a certain solution structure

while the latter forces the EA to "keep track" of a trail to solution candidates with worse fitness which may later evolve to good individuals with traits different from the currently exploited ones.

Which of the two approaches is better has not yet been tested with comparative experiments and is part of our future work. At the present moment, we assume that in real-valued search or problem spaces, clearing should be more suitable whereas we know from experiments using our approach only that *SCP* performs very good in combinatorial problems [1650, 2188] Genetic Programming (see Section 21.3.2, for instance).

TODO add remaining selection algorithms

2.5 Reproduction

An optimization algorithm uses the information gathered up to step t for creating the solution candidates to be evaluated in step $t+1$. There exist different methods to do so. In evolutionary algorithms, the aggregated information corresponds to the population *Pop* and the set of best individuals *Arc* if such an archive is maintained. The search operations searchOp $\in$ *Op* in used in the evolutionary algorithm family are called *reproduction* operation, inspired by the biological procreation mechanisms[27] of mother nature [1730]. There are four basic operations:

1. *Creation* has no direct natural paragon; it simple creates a new genotype without any ancestors or heritage. Hence, it roughly can be compared with the occurrence of the first living cells from out a soup of certain chemicals[28].
2. *Duplication* resembles the cell division[29], resulting in two individuals similar to one parent.
3. *Mutation* in evolutionary algorithms corresponds to small, random variations in the genotype of an individual, exactly like its natural counterpart[30].
4. Like in sexual reproduction, *recombination*[31] combines two parental genotypes to a new genotype including traits from both elders.

In the following, we will discuss these operations in detail and provide general definitions form them.

Definition 2.9 (Creation). The creation operation "create" is used to produce a new genotype $g \in \mathbb{G}$ with a random configuration.

$$g = \text{create}() \Rightarrow g \in \mathbb{G} \tag{2.25}$$

When an evolutionary algorithm starts, no information about the search space has been gathered yet. Hence, we cannot use existing solution candidates to derive new ones and search operations with an arity higher than zero cannot be applied. Creation is thus used to fill the initial population $Pop(t=0)$.

Definition 2.10 (Duplication). The duplication operation duplicate : $\mathbb{G} \mapsto \mathbb{G}$ is used to create an exact copy of an existing genotype $g \in \mathbb{G}$.

$$g = \text{duplicate}(g) \;\; \forall g \in \mathbb{G} \tag{2.26}$$

[27] http://en.wikipedia.org/wiki/Reproduction [accessed 2007-07-03]
[28] http://en.wikipedia.org/wiki/Abiogenesis [accessed 2008-03-17]
[29] http://en.wikipedia.org/wiki/Cell_division [accessed 2008-03-17]
[30] http://en.wikipedia.org/wiki/Mutation [accessed 2007-07-03]
[31] http://en.wikipedia.org/wiki/Sexual_reproduction [accessed 2008-03-17]

Duplication is just a placeholder for copying an element of the search space, i. e., it is what occurs when neither mutation nor recombination are applied. It is useful to increase the share of a given type of individual in a population.

Definition 2.11 (Mutation). The mutation operation mutate : $\mathbb{G} \mapsto \mathbb{G}$ is used to create a new genotype $g_n \in \mathbb{G}$ by modifying an existing one. The way this modification is performed is application-dependent. It may happen in a randomized or in a deterministic fashion.

$$g_n = \text{mutate}(g) : g \in \mathbb{G} \Rightarrow g_n \in \mathbb{G} \tag{2.27}$$

Definition 2.12 (Recombination). The recombination (or crossover[32]) operation recombine : $\mathbb{G} \times \mathbb{G} \mapsto \mathbb{G}$ is used to create a new genotype $g_n \in \mathbb{G}$ by combining the features of two existing ones. Depending on the application, this modification may happen in a randomized or in a deterministic fashion.

$$g_n = \text{recombine}(g_a, g_b) : g_a, g_b \in \mathbb{G} \Rightarrow g_n \in \mathbb{G} \tag{2.28}$$

Notice that the term *recombination* is more general than *crossover* since it stands for arbitrary search operations that combines the traits of two individuals. Crossover, however, is only used if the elements search space $\mathbb{G}$ are linear representations. Then, it stands for exchanging parts of these so-called strings.

Now we can define the set Op_{EA} of search operations most commonly applied in evolutionary algorithms as

$$Op_{EA} = \{\text{create}, \text{duplicate}, \text{mutate}, \text{recombine}\} \tag{2.29}$$

All of them can be combined arbitrarily. It is, for instance, not unusual to mutate the results of a recombination operation, i. e., to perform mutate(recombine(g_1, g_2)).

The four operators are altogether used to reproduce whole populations of individuals.

Definition 2.13 (reproducePop**).** The population reproduction operation $Pop = \text{reproducePop}(Mate)$ is used to create a new population Pop by applying the reproduction operations to the mating pool $Mate$.

$$\begin{aligned} Pop = \text{reproducePop}(Mate) \Rightarrow\ & \forall p \in Mate \Rightarrow p \in P,\ \forall p \in Pop \Rightarrow p \in P,\ \text{len}(Pop) = \text{len}(Mate) \\ & \forall p \in Pop \Rightarrow\ p.g = \text{create}() \vee \\ & \qquad p.g = \text{duplicate}(p_{old}.g) : p_{old} \in Mate \vee \\ & \qquad p.g = \text{mutate}(p_{old}.g) : p_{old} \in Mate \vee \\ & \qquad p.g = \text{recombine}(p_{old1}.g, p_{old2}.g) : \\ & \qquad\qquad p_{old1}, p_{old2} \in Mate \end{aligned} \tag{2.30}$$

For creating an initial population of the size s, we furthermore define the function createPop(s) in Algorithm 2.18.

2.5.1 NCGA Reproduction

The Neighborhood Cultivation Genetic Algorithm by Watanabe et al. [2160] discussed in **??** uses a special reproduction method. Recombination is performed only on neighboring individuals, which leads to child genotypes close to their parents. This so-called neighborhood cultivation shifts the recombination-operator more into the direction exploitation, i. e., NCGA uses crossover for investigating the close surrounding of known solution candidates. The idea is that parents that do not differ much from each other are more likely to be compatible in order to produce functional offspring than parents that have nothing in common.

[32] http://en.wikipedia.org/wiki/Recombination [accessed 2007-07-03]

Algorithm 2.18: $Pop \longleftarrow \text{createPop}(s)$

Input: s: the number of individuals in the new population
Input: [implicit] create: the creation operator
Data: i: a counter variable
Output: Pop: the new population of randomly created individuals ($\text{len}(Pop) = s$)

```
1 begin
2     Pop ⟵ ()
3     for i ⟵ 0 up to s − 1 do
4         Pop ⟵ addListItem(Pop, create())
5     return Pop
6 end
```

Neighborhood cultivation is achieved in Algorithm 2.19 by sorting the mating pool along one *focused* objective. Then, the elements situated directly besides each other are recombined. The focus on the objective rotates in a way that in a three-objective optimization the first objective is focused at the beginning, then the second, then the third and after that again the first. The algorithm shown here receives the additional parameter *foc* which denotes the focused objective. Both, recombination and mutation are performed with an implicitly defined probability (r and m, respectively).

Algorithm 2.19: $Pop \longleftarrow \text{ncgaReproducePop}_{foc}(Mate)$

Input: $Mate$: the mating pool
Input: foc: the objective currently focused
Input: [implicit] recombine, mutate: the recombination and mutation routines
Input: [implicit] r, m: the probabilities of recombination and mutation
Data: i: a counter variable
Output: Pop: the new population with $\text{len}(Pop) = \text{len}(Mate)$

```
1 begin
2     Pop ⟵ sortList_a(Mate, f_foc)
3     for i ⟵ 0 up to len(Pop) − 1 do
4         if (random_u() ≤ r) ∧ (i < len(Pop) − 1) then Pop[i] ⟵ recombine(Pop[i], Pop[i+1])
5         if random_u() ≤ m then Pop[i] ⟵ mutate(Pop[i])
6     return Pop
7 end
```

2.6 Algorithms

Besides the basic evolutionary algorithms introduced in Section 2.1.3 on page 98, there exists a variety of other, more sophisticated approaches. Many of them deal especially with multi-objective optimization which imposes new challenges on fitness assignment and selection. In this section we discuss the most prominent of these evolutionary algorithms.

2.6.1 VEGA

The very first multi-objective genetic algorithm is the *Vector Evaluated Genetic Algorithm* (VEGA) created by Schaffer [1821, 1822] in the mid-1980s. The main difference between VEGA and the basic form of evolutionary algorithms is the modified selection algorithm which you can find discussed in Section 2.4.7 on page 133. This selection algorithm solely

relies on the objective functions F and does not use any preceding fitness assignment process nor can it incorporate a prevalence comparison scheme cmp_F. However, it has severe weaknesses also discussed in Section 2.4.7 and thus cannot be considered as an efficient approach to multi-objective optimization.

Algorithm 2.20: $X^\star \longleftarrow \text{vega}(F, s)$

Input: F: the objective functions
Input: ps: the population size
Data: t: the generation counter
Data: Pop: the population
Data: $Mate$: the mating pool
Data: v: the fitness function resulting from the fitness assigning process
Output: $X^\star$: the set of the best elements found

```
1 begin
2     t ⟵ 0
3     Pop ⟵ createPop(ps)
4     while ¬terminationCriterion() do
5         Mate ⟵ vegaSelect(Pop, F, ps)
6         t ⟵ t + 1
7         Pop ⟵ reproducePop(Mate)
8     return extractPhenotypes(extractOptimalSet(Pop))
9 end
```

TODO add remaining EAs

3

Genetic Algorithms

3.1 Introduction

Genetic algorithms[1] (GAs) are a subclass of evolutionary algorithms where the elements of the search space $\mathbb{G}$ are binary strings ($\mathbb{G} = \mathbb{B}^*$) or arrays of other elementary types. As sketched in Figure 3.1, the genotypes are used in the reproduction operations whereas the values of the objective functions $f \in F$ are computed on basis of the phenotypes in the problem space $\mathbb{X}$ which are obtained via the genotype-phenotype mapping "gpm". [821, 940, 916, 2208]

The roots of genetic algorithms go back to the mid-1950s, where biologists like Barricelli [150, 151, 152, 153] and the computer scientist Fraser [742] began to apply computer-aided simulations in order to gain more insight into genetic processes and the natural evolution and selection. Bremermann [287] and Bledsoe [216, 215, 217, 218] used evolutionary approaches based on binary string genomes for solving inequalities, for function optimization, and for determining the weights in neural networks in the early 1960s [219]. At the end of that decade, important research on such search spaces was contributed by Bagley [116] (who introduced the term *genetic algorithm*), Rosenberg [1760], Cavicchio, Jr. [354, 355], and Frantz [741] – all based on the ideas of Holland at the University of Michigan. As a result of Holland's work [937, 939, 940, 938] genetic algorithms as a new approach for problem solving could be formalized finally became widely recognized and popular. Today, there are many applications in science, economy, and research and development [1681] that can be tackled with genetic algorithms. Therefore, various forms of genetic algorithms [423] have been developed to. Some genetic algorithms[2] like the human-based genetic algorithms[3] (HBGA), for instance, even require human beings for evaluating or selecting the solution candidates [1884, 1997, 1998, 1178, 883]

It should further be mentioned that, because of the close relation to biology and since genetic algorithms were originally applied to single-objective optimization, the objective functions f here are often referred to as *fitness functions*. This is a historically grown misnaming which should not be mixed up with the fitness assignment processes discussed in Section 2.3 on page 111 and the fitness values v used in the context of this book.

[1] http://en.wikipedia.org/wiki/Genetic_algorithm [accessed 2007-07-03]
[2] http://en.wikipedia.org/wiki/Interactive_genetic_algorithm [accessed 2007-07-03]
[3] http://en.wikipedia.org/wiki/HBGA [accessed 2007-07-03]

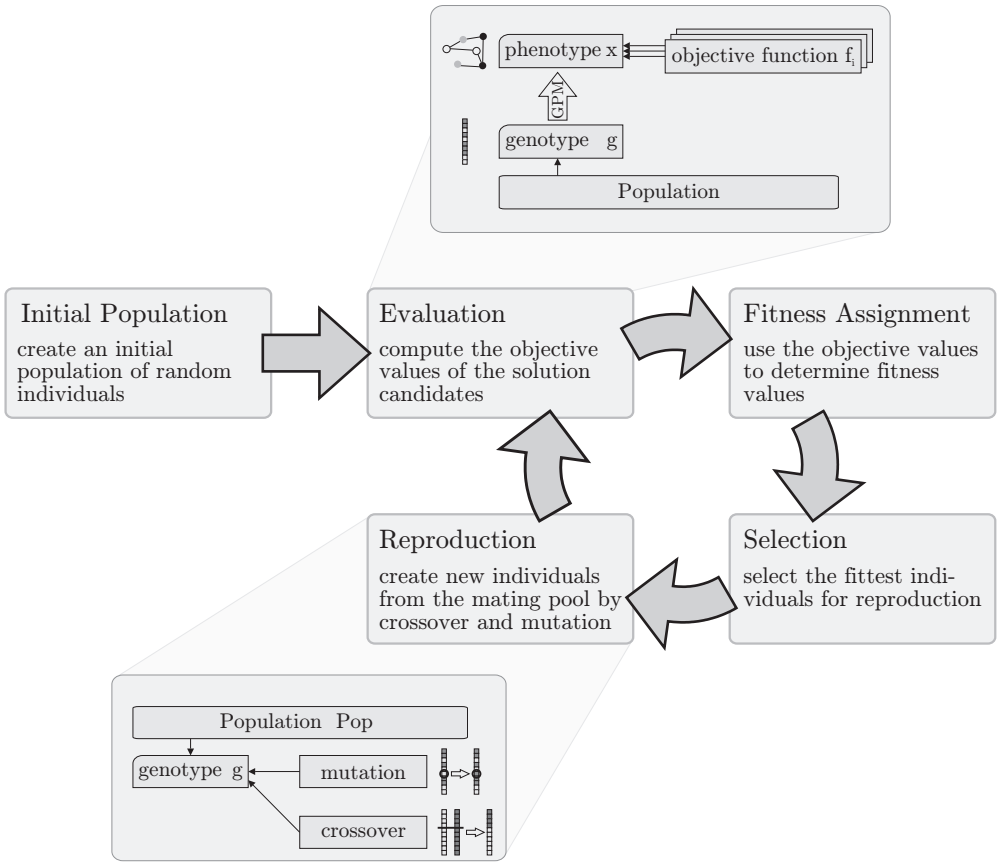


Figure 3.1: The basic cycle of genetic algorithms.

3.2 General Information

3.2.1 Areas Of Application

Some example areas of application of genetic algorithms are:

Application	References
Scheduling	[1275, 417, 1228, 160, 340, 339, 341]
Chemistry, Chemical Engineering	[475, 2269, 474, 476, 531, 2127, 1075, 1401]
Medicine	[319, 1900, 2278, 2117]
Data Mining and Data Analysis	[1424, 1089, 834, 1991, 445]
Geometry and Physics	[366, 367, 966, 1222, 1223]
Economics and Finance	[2302]
Networking and Communication	[628, 1861, 1220, 290, 1164, 2324] see Section 23.2 on page 401
Electrical Engineering and Circuit Design	[1304, 1305, 1306]

Image Processing	[25]
Combinatorial Optimization	[1480, 1134, 1754, 2020, 2323, 32]

3.2.2 Conferences, Workshops, etc.

Some conferences, workshops and such and such on genetic algorithms are:

EUROGEN: Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems
see Section 2.2.2 on page 106

FOGA: Foundations of Genetic Algorithms
http://www.sigevo.org/ [accessed 2007-09-01]
History: 2007: Mexico City, México, see [1960]
2005: Aizu-Wakamatsu City, Japan, see [2259]
2002: Torremolinos, Spain, see [519]
2000: Charlottesville, VA, USA, see [1927]
1998: Madison, WI, USA, see [139]
1996: San Diego, CA, USA, see [172]
1994: Estes Park, Colorado, USA, see [2214]
1992: Vail, Colorado, USA, see [2209]
1990: Bloomington Campus, Indiana, USA, see [1924]

FWGA: Finnish Workshop on Genetic Algorithms and Their Applications
NWGA: Nordic Workshop on Genetic Algorithms
History: 1997: Helsinki, Finland, see [30]
1996: Vaasa, Finland, see [29]
1995: Vaasa, Finland, see [28]
1994: Vaasa, Finland, see [27]
1992: Espoo, Finland, see [26]

GALESIA: International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications
now part of *CEC*, see Section 2.2.2 on page 105
History: 1997: Glasgow, UK, see [990]
1995: Scheffield, UK, see [2309]

GECCO: Genetic and Evolutionary Computation Conference
see Section 2.2.2 on page 107

GEM: International Conference on Genetic and Evolutionary Methods
History: 2008: Las Vegas, Nevada, USA, see [81]
2007: Las Vegas, Nevada, USA, see [80]

ICGA: International Conference on Genetic Algorithms
Now part of GECCO, see Section 2.2.2 on page 107
History: 1997: East Lansing, Michigan, USA, see [98]
1995: Pittsburgh, PA, USA, see [636]
1993: Urbana-Champaign, IL, USA, see [730]
1991: San Diego, CA, USA, see [170]

1989: Fairfax, Virginia, USA, see [1820]
1987: Cambridge, MA, USA, see [857]
1985: Pittsburgh, PA, USA, see [856]

ICANNGA: International Conference on Adaptive and Natural Computing Algorithms
see Section 2.2.2 on page 108

Mendel: International Conference on Soft Computing
see Section 1.6.2 on page 90

3.2.3 Online Resources

Some general, online available ressources on genetic algorithms are:

`http://www.obitko.com/tutorials/genetic-algorithms/` [accessed 2008-05-17]
Last update: 1998
Description: A very thorough introduction to genetic algorithms by Marek Obitko

`http://www.aaai.org/AITopics/html/genalg.html` [accessed 2008-05-17]
Last update: up-to-date
Description: The genetic algorithms and Genetic Programming pages of the AAAI

`http://www.illigal.uiuc.edu/web/` [accessed 2008-05-17]
Last update: up-to-date
Description: The Illinois Genetic Algorithms Laboratory (IlliGAL)

`http://www.cs.cmu.edu/Groups/AI/html/faqs/ai/genetic/top.html` [accessed 2008-05-17]
Last update: 1997-08-10
Description: The Genetic Algorithms FAQ.

`http://www.rennard.org/alife/english/gavintrgb.html` [accessed 2008-05-17]
Last update: 2007-07-10
Description: An introduction to genetic algorithms by Jean-Philippe Rennard.

`http://www.optiwater.com/GAsearch/` [accessed 2008-06-08]
Last update: 2003-11-15
Description: GA-Search – The Genetic Algorithms Search Engine

3.2.4 Books

Some books about (or including significant information about) genetic algorithms are:

Goldberg [821]: *Genetic Algorithms in Search, Optimization and Machine Learning*
Mitchell [1431]: *An Introduction to Genetic Algorithms*
Davis [495]: *Handbook of Genetic Algorithms*
Haupt and Haupt [905]: *Practical Genetic Algorithms*
Gen and Cheng [787]: *Genetic Algorithms and Engineering Design*
Chambers [368]: *Practical Handbook of Genetic Algorithms: Applications*
Chambers [369]: *Practical Handbook of Genetic Algorithms: New Frontiers*
Chambers [370]: *Practical Handbook of Genetic Algorithms: Complex Coding Systems*
Holland [940]: *Adaptation in Natural and Artificial Systems*
Gen and Chen [786]: *Genetic Algorithms (Engineering Design and Automation)*
Cant'u-Paz [330]: *Efficient and Accurate Parallel Genetic Algorithms*
Heistermann [915]: *Genetische Algorithmen. Theorie und Praxis evolutionärer Optimierung*

Schöneburg, Heinzmann, and Feddersen [1831]: *Genetische Algorithmen und Evolutionsstrategien*
Gwiazda [873]: *Crossover for single-objective numerical optimization problems*
Schaefer and Telega [1819]: *Foundations of Global Genetic Optimization*
Karr and Freeman [1093]: *Industrial Applications of Genetic Algorithms*
Bäck [99]: *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*
Davis [494]: *Genetic Algorithms and Simulated Annealing*
Alba and Dorronsoro [33]: *Cellular Genetic Algorithms*

3.3 Genomes in Genetic Algorithms

Most of the terminology which we have defined in Section 1.3 and used throughout this book stems from the GA sector. The search spaces $\mathbb{G}$ of genetic algorithms, for instance, are referred to *genome* and its elements are called genotypes. Genotypes in nature encompass the whole hereditary information of an organism encoded in the DNA[4]. The DNA is a string of base pairs that encodes the phenotypical characteristics of the creature it belongs to. Like their natural prototypes, the genomes in genetic algorithms are strings, linear sequences of certain data types [821, 945, 1431]. Because of the linear structure, these genotypes are also often called *chromosomes*. In genetic algorithms, we most often use chromosomes which are strings of one and the same data type, for example bits or real numbers.

Definition 3.1 (String Chromosome). A string chromosome can either be a fixed-length tuple (Equation 3.1) or a variable-length list (Equation 3.2).

In the first case, the loci i of the genes g_i are constant and, hence, the tuples may contain elements of different types $\mathbb{G}_i$.

$$\mathbb{G} = \{\forall\,(g_{[1]}, g_{[2]}, .., g_{[n]}) : g_{[i]} \in \mathbb{G}_i\ \forall i \in 1..n\} \tag{3.1}$$

This is not given in variable-length string genomes. Here, the positions of the genes may shift when the reproduction operations are applied. Thus, all elements of such genotypes must have the same type $\mathbb{G}_T$.

$$\mathbb{G} = \{\forall \mathit{lists}\ g : g_{[i]} \in \mathbb{G}_T\ \forall 0 \leq i < \text{len}(g)\} \tag{3.2}$$

String chromosomes are normally bit strings, vectors of integer numbers, or vectors of real numbers. Genetic algorithms with numeric vector genomes in their *natural representation*, i. e., where $\mathbb{G} = \mathbb{X} \subseteq \mathbb{R}^n$ are called *real-encoded* [1107]. Today, more sophisticated methods for evolving good strings (vectors) of (real) numbers exist (such as Evolution Strategies, Differential Evolution, or Particle Swarm Optimization) than processing them like binary strings with the standard reproduction operations of GAs.

Bit string genomes are sometimes complemented with the application of gray coding[5] during the genotype-phenotype mapping. This is done in an effort to preserve locality (see Section 1.4.3) and ensure that small changes in the genotype will also lead to small changes in the phenotypes [349]. Collins and Eaton [430] studied different encodings for GAs and found that their *E-code* outperform both gray and direct binary coding in function optimization. Messy genomes (see Section 3.7) where introduced to improve locality by linkage learning.

Genetic algorithms are the original prototype of evolutionary algorithms and therefore, fully adhere to the description given in Section 2.1.2. They provide search operators which closely copy sexual and asexual reproduction schemes from nature. In such "sexual" search

[4] You can find an illustration of the DNA in Figure 1.14 on page 42
[5] http://en.wikipedia.org/wiki/Gray_coding [accessed 2007-07-03]

operations, the genotypes of the two parents genotypes will recombine. In asexual reproduction, mutations are the only changes that occur. It is very common to apply both principles in conjunction, i. e., to first recombine two elements from the search space and subsequently, make them subject to mutation.

In nature, life begins with a single cell which divides[6] time and again until a mature individual is formed[7] after the genetic information has been reproduced. The emergence of a phenotype from its genotypic representation is called embryogenesis in biology and its counterparts in evolutionary search are the genotype-phenotype mapping and artificial embryogeny which we will discuss in Section 3.8 on page 155.

Let us shortly recapitulate the structure of the elements g of the search space $\mathbb{G}$. A gene (see Definition 1.23 on page 43) is the basic informational unit in a genotype g. Depending on the genome, a gene can be a bit, a real number, or any other structure. In biology, a gene is a segment of nucleic acid that contains the information necessary to produce a functional RNA product in a controlled manner. An allele (see Definition 1.24) is a value of specific gene in nature and in EAs alike. The locus (see Definition 1.25) is the position where a specific gene can be found in a chromosome. Besides the functional genes and their alleles, there are also parts of natural genomes which have no (obvious) function [2161, 819]. The American biochemist Gilbert [806] coined the term *intron*[8] for such parts. Similar structures can also be observed in evolutionary algorithms with variable-length encodings.

Definition 3.2 (Intron). Parts of a genotype $g \in \mathbb{G}$ that does not contribute to the phenotype $x = \text{gpm}(g)$ are referred to as introns.

Biological introns have often been thought of as junk DNA or "old code", i. e., parts of the genome that were translated to proteins in evolutionary past, but now are not used anymore. Currently though, many researchers assume that introns are maybe not as useless as initially assumed [467]. Instead, they seem to provide support for efficient splicing, for instance. The role of introns in genetic algorithms is as same as mysterious. They represent a form of redundancy – which is known to have possible as well as negative effects, as outlined in Section 1.4.5 on page 67 and Section 4.10.3.

Figure 3.2 combines Figure 1.15 on page 45 and Figure 1.13 and illustrates the relations between the aforementioned entities in a bit string genome $\mathbb{G} = \mathbb{B}^4$ of the length 4, where two bits encode for one coordinate in a two-dimensional plane. Additional bits could appended to the genotypes because a variable-length representation is used for some strange reason, for instance. Then, these could occur as introns and would not influence the phenotype in the example.

[6] http://en.wikipedia.org/wiki/Cell_division [accessed 2007-07-03]

[7] Matter of fact, cell division will continue until the individual dies. However, this is not important here.

[8] http://en.wikipedia.org/wiki/Intron [accessed 2007-07-05]

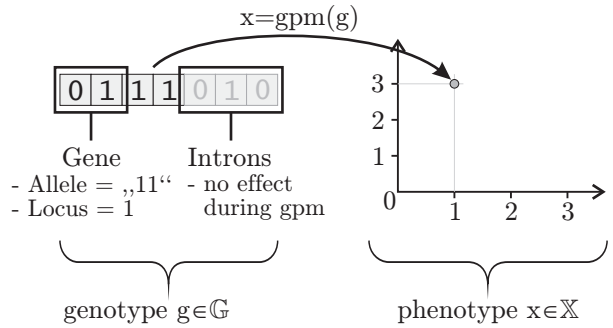


Figure 3.2: A four bit string genome $\mathbb{G}$ and a fictitious phenotype $\mathbb{X}$.

3.4 Fixed-Length String Chromosomes

Especially widespread in genetic algorithms are search spaces based on fixed-length chromosomes. The properties of their crossover and mutation operations are well known and an extensive body of research on them is available [821, 945].

3.4.1 Creation: Nullary Reproduction

Creation of fixed-length string individuals means simple to create a new tuple of the structure defined by the genome and initialize it with random values. In reference to Equation 3.1 on page 145, we could roughly describe this process with Equation 3.3.

$$\text{create}_{fl}() \equiv (g_{[1]}, g_{[2]}, .., g_{[n]}) : g_{[i]} = \mathbb{G}_{i[\lfloor \text{random}_u() * \text{len}(\mathbb{G}_i) \rfloor]} \; \forall i \in 1..n \tag{3.3}$$

3.4.2 Mutation: Unary Reproduction

Mutation is an important method for preserving the diversity of the solution candidates by introducing small, random changes into them. In fixed-length string chromosomes, this can be achieved by randomly modifying the value (allele) of a gene, as illustrated in Fig. 3.3.a. Fig. 3.3.b shows the more general variant of this form of mutation where $0 < n < \text{len}(g)$ locations in the genotype g are changed at once. In binary coded chromosomes, for example, these genes would be bits which can simply be toggled. For real-encoded genomes, modifying an element g_i can be done by replacing it with a number drawn from a normal distribution with expected value g_1, like $g_i^{new} \sim N\left(g_1, \sigma^2\right)$.

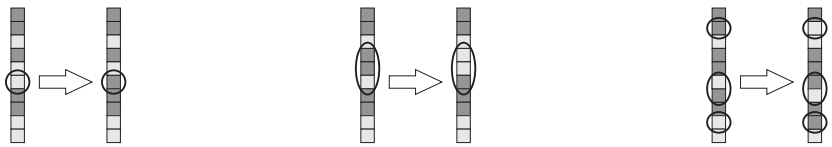


Fig. 3.3.a: Single-gene mutation. Fig. 3.3.b: Multi-gene mutation (a). Fig. 3.3.c: Multi-gene mutation (b).

Figure 3.3: Value-altering mutation of string chromosomes.

3.4.3 Permutation: Unary Reproduction

The permutation operation is an alternative mutation method where the alleles of two genes are exchanged as sketched in Figure 3.4. This, of course, makes only sense if all genes have similar data types. Permutation is, for instance, useful when solving problems that involve finding an optimal sequence of items, like the travelling salesman problem [1263, 78]. Here, a genotype g could encode the sequence in which the cities are visited. Exchanging two alleles then equals of switching two cities in the route.

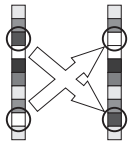


Figure 3.4: Permutation applied to a string chromosome.

3.4.4 Crossover: Binary Reproduction

Amongst all evolutionary algorithms, genetic algorithms have the recombination operation which probably comes closest to the natural paragon. Figure 3.5 outlines the recombination of two string chromosomes, the so-called *crossover*, which is performed by swapping parts of two genotypes.

When performing single-point crossover (SPX[9]), both parental chromosomes are split at a randomly determined *crossover point*. Subsequently, a new child genotype is created by appending the second part of the second parent to the first part of the first parent as illustrated in Fig. 3.5.a. In two-point crossover (TPX, sketched in Fig. 3.5.b), both parental genotypes are split at two points and a new offspring is created by using parts number one and three from the first, and the middle part from the second parent chromosome. Fig. 3.5.c depicts the generalized form of this technique: the n-point crossover operation, also called multi-point crossover (MPX). For fixed-length strings, the crossover points for both parents are always identical.

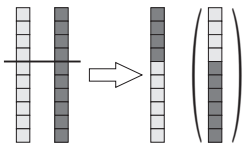


Fig. 3.5.a: Single-point Crossover (SPX).

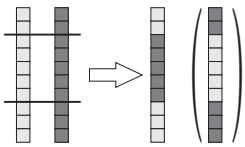


Fig. 3.5.b: Two-point Crossover (TPX).

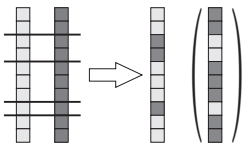


Fig. 3.5.c: Multi-point Crossover (MPX).

Figure 3.5: Crossover (recombination) operators for fixed-length string genomes.

[9] This abbreviation is also used for simplex crossover, see Section 16.4.

3.5 Variable-Length String Chromosomes

Variable-length genomes for genetic algorithms where first proposed by Smith in his PhD thesis [1912]. There, he introduced a new variant of classifier systems[10] with the goal of evolving programs for playing poker [1912, 1688].

3.5.1 Creation: Nullary Reproduction

Variable-length strings can be created by first randomly drawing a length $l > 0$ and then creating a list of that length filled with random elements.

3.5.2 Mutation: Unary Reproduction

If the string chromosomes are of variable length, the set of mutation operations introduced in Section 3.4 can be extended by two additional methods. First, we could insert a couple of genes with randomly chosen alleles at any given position into a chromosome (Fig. 3.6.a). Second, this operation can be reversed by deleting elements from the string (Fig. 3.6.b). It should be noted that both, insertion and deletion, are also implicitly be performed by crossover. Recombining two identical strings with each other can, for example, lead to deletion of genes. The crossover of different strings may turn out as an insertion of new genes into an individual.

Since the reproduction operations can change the length of a genotypes (therefore the name "variable-length"), variable-length strings need to be constructed of elements of the same type. There is no longer a constant relation between locus and type.

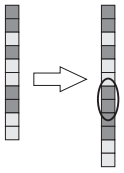


Fig. 3.6.a: Insertion of random genes.

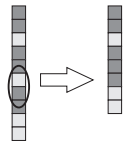


Fig. 3.6.b: Deletion of genes.

Figure 3.6: Search operators for variable-length strings (additional to those from Section 3.4.2 and Section 3.4.3).

3.5.3 Crossover: Binary Reproduction

For variable-length string chromosomes, the same crossover operations are available as for fixed-length strings except that the strings are no longer necessarily split at the same loci. The lengths of the new strings resulting from such a *cut and splice* operation may differ from the lengths of the parents, as sketched in Figure 3.7. A special case of this type of recombination is the homologous crossover, where only genes at the same loci are exchanged. This method is discussed thoroughly in Section 4.6.7 on page 195.

[10] See Chapter 7 for more information on classifier systems.

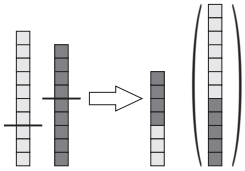


Fig. 3.7.a: Single-Point Crossover

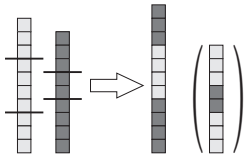


Fig. 3.7.b: Two-Point Crossover

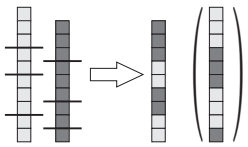


Fig. 3.7.c: Multi-Point Crossover

Figure 3.7: Crossover of variable-length string chromosomes.

3.6 Schema Theorem

The Schema Theorem is a special instance of forma analysis (discussed in Section 1.5.1 on page 80) for genetic algorithms. Matter of fact, it is older than its generalization and was first stated by Holland back in 1975 [940, 512, 945]. Here we will first introduce the basic concepts of schemata, masks, and wildcards before going into detail about the Schema Theorem itself, its criticism, and the related Building Block Hypothesis.

3.6.1 Schemata and Masks

Assume that the genotypes g in the search space $\mathbb{G}$ of genetic algorithms are strings of a fixed-length l over an alphabet[11] Σ, i. e., $\mathbb{G} = \Sigma^l$. Normally, Σ is the binary alphabet $\Sigma = \{\texttt{true}, \texttt{false}\} = \{0, 1\}$. From forma analysis, we know that properties can be defined on the genotypic or the phenotypic space. For fixed-length string genomes, we can consider the values at certain loci as properties of a genotype. There are two basic principles on defining such properties: masks and do not care symbols.

Definition 3.3 (Mask). For a fixed-length string genome $\mathbb{G} = \Sigma^l$, we define the set of all genotypic masks M_l as the power set[12] of the valid loci $M_l = \mathcal{P}(\{1, \ldots, l\})$ [2167]. Every mask $m_i \in M_l$ defines a property ϕ_i and an equivalence relation:

$$g \sim_{\phi_i} h \Leftrightarrow g_{[j]} = h_{[j]} \; \forall j \in m_i \tag{3.4}$$

The order "order(m_i)" of the mask m_i is the number of loci defined by it:

$$\text{order}(m_i) = |m_i| \tag{3.5}$$

The defined length $\delta(m_i)$ of a mask m_i is the maximum distance between two indices in the mask:

$$\delta(m_i) = \max\{|j - k| \;\; \forall j, k \in m_i\} \tag{3.6}$$

A mask contains the indices of all elements in a string that are interesting in terms of the property it defines. Assume we have bit strings of the length $l = 3$ as genotypes ($\mathbb{G} = \mathbb{B}^3$). The set of valid masks M_3 is then $M_3 = \{\{1\}, \{2\}, \{3\}, \{1, 3\}, \{1, 3\}, \{2, 3\}, \{1, 2, 3\}\}$. The mask $m_1 = \{1, 2\}$, for example, specifies that the values at the loci 1 and 2 of a genotype denote the value of a property ϕ_1 and the value of the bit at position 3 is irrelevant. Therefore, it defines four formae $A_{\phi_1=(0,0)} = \{(0, 0, 0), (0, 0, 1)\}$, $A_{\phi_1=(0,1)} = \{(0, 1, 0), (0, 1, 1)\}$, $A_{\phi_1=(1,0)} = \{(1, 0, 0), (1, 0, 0)\}$, and $A_{\phi_1=(1,1)} = \{(1, 1, 0), (1, 1, 1)\}$.

Definition 3.4 (Schema). A forma defined on a string genome concerning the values of the characters at specified loci is called *Schema* [940, 389].

[11] Alphabets and such and such are defined in Section 30.3 on page 561.
[12] The power set you can find described in Definition 27.9 on page 458.

3.6.2 Wildcards

The second method of specifying such schemata is to use *don't care* symbols (wildcards) to create "blueprints" H of their member individuals. Therefore, we place the don't care symbol $*$ at all irrelevant positions and the characterizing values of the property at the others.

$$\forall j \in 1..l \Rightarrow H_{[j]} = \begin{cases} g_{[j]} & \text{if } j \in m_i \\ * & \text{otherwise} \end{cases} \tag{3.7}$$

$$H_{[j]} \in \Sigma \cup \{*\} \ \forall j \in 1..l \tag{3.8}$$

$$\tag{3.9}$$

We now can redefine the aforementioned schemata like: $A_{\phi_1=(0,0)} \equiv H_1 = (0, 0, *)$, $A_{\phi_1=(0,1)} \equiv H_2 = (0, 1, *)$, $A_{\phi_1=(1,0)} \equiv H_3 = (1, 0, *)$, and $A_{\phi_1=(1,1)} \equiv H_4 = (1, 1, *)$. These schemata mark hyperplanes in the search space $\mathbb{G}$, as illustrated in Figure 3.8 for the three bit genome. Schemas correspond to masks and thus, definitions like the *defined length* and *order* can easily be transported into their context.

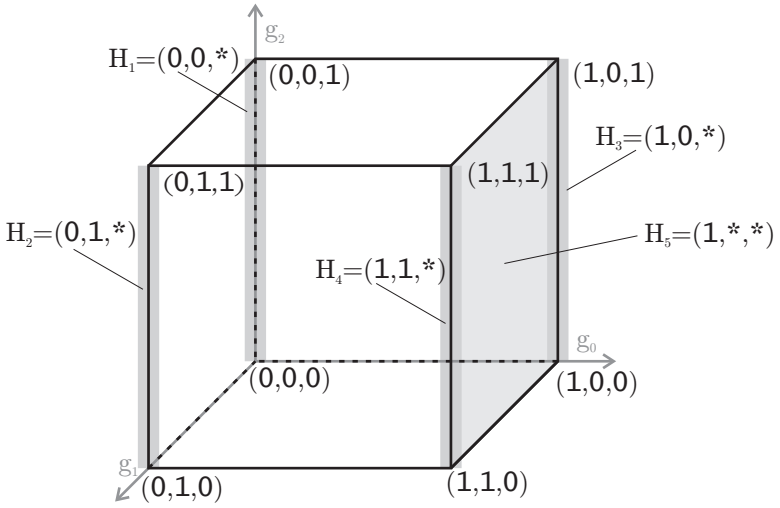


Figure 3.8: An example for schemata in a three bit genome.

3.6.3 Holland's Schema Theorem

The Schema Theorem[13] was defined by Holland [940] for genetic algorithms which use fitness-proportionate selection (see Section 2.4.3 on page 124) where fitness is subject to maximization [512, 945].

$$\text{countOccurences}(H, Pop)_{t+1} \geq \frac{\text{countOccurences}(H, Pop)_t * \overline{v}(H)_t}{\overline{v}_t} (1 - p) \tag{3.10}$$

where

1. $\text{countOccurences}(H, Pop)_t$ is the number of instances of a given schema defined by the blueprint H in the population Pop of generation t,

[13] http://en.wikipedia.org/wiki/Holland%27s_Schema_Theorem [accessed 2007-07-29]

2. $\overline{v}(H)_t$ is the average fitness of the members of this schema (observed in time step t),
3. $\overline{v}_t$ is the average fitness of the population in time step t, and
4. p is the probability that an instance of the schema will be "destroyed" by a reproduction operation, i. e., the probability that the offspring of an instance of the schema is not an instance of the schema.

From this formula can be deduced that genetic algorithms will generate for short, above-average fit schemata an exponentially rising number of samples. This is because they will multiply with a certain factor in each generation and only few of them are destroyed by the reproduction operations. In the special case of single-point crossover (crossover rate cr) and single-bit mutation (mutation rate mr) in a binary genome of the fixed length l $\left(\mathbb{G} = \mathbb{B}^l\right)$, the destruction probability p is noted in Equation 3.11.

$$p = cr\frac{\delta(H)}{l-1} + mr\frac{\text{order}(H)}{l} \tag{3.11}$$

3.6.4 Criticism of the Schema Theorem

The deduction that good schemata will spread exponentially is only a very optimistic assumption and not generally true. If a highly fit schema has many offspring with good fitness, this will also improve the overall fitness of the population. Hence, the probabilities in Equation 3.10 will shift over time. Generally, the Schema Theorem represents a lower bound that will only hold for one generation [2208]. Trying to derive predictions for more than one or two generations using the Schema Theorem as is will lead to deceptive or wrong results [858, 854].

Furthermore, the population of a genetic algorithm only represents a sample of limited size of the search space $\mathbb{G}$. This limits the reproduction of the schemata but also makes statements about probabilities in general more complicated. Since we only have samples of the schemata H and cannot be sure if $\overline{v}(H)_t$ really represents the average fitness of all the members of the schema (that is why we annotate it with $_t$ instead of writing $\overline{v}(H)$). Thus, even reproduction operators which preserve the instances of the schema may lead to a decrease of $\overline{v}(H)_{t+..}$ by time. It is also possible that parts of the population already have converged and other members of a schema will not be explored anymore, so we do not get further information about its real utility.

Additionally, we cannot know if it is really good if one specific schema spreads fast, even it is very fit. Remember that we have already discussed the exploration versus exploitation topic and the importance of diversity in Section 1.4.2 on page 60.

Another issue is that we implicitly assume that most schemata are compatible and can be combined, i. e., that there is low interaction between different genes. This is also not generally valid: Epistatic effects, for instance, can lead to schema incompatibilities. The expressiveness of masks and blueprints even is limited and can be argued that there are properties which we cannot specify with them. Take the set D_3 of numbers divisible by three for example $D_3 = \{3, 6, 9, 12, ..\}$. Representing them as binary strings will lead to $D_3 = \{0011, 0110, 1001, 1100, \dots\}$ if we have a bit-string genome of the length 4. Obviously, we cannot seize these genotypes in a schema using the discussed approach. They may, however, be gathered in a forma. The Schema Theorem, however, cannot hold for such a forma since the probability p of destruction may be different from instance to instance.

3.6.5 The Building Block Hypothesis

According to Harik [896], the substructure of a genotype which allows it to match to a schema is called a *building block*. The Building Block Hypothesis (BBH) proposed by Goldberg [821], Holland [940] is based on two assumptions:

1. When a genetic algorithm solves a problem, there exist some low-order, low-defining length schemata with above-average fitness (the so-called *building blocks*).
2. These schemata are combined step by step by the genetic algorithm in order to form larger and better strings. By using the building blocks instead of testing any possible binary configuration, genetic algorithms efficiently decrease the complexity of the problem. [821]

Although it seems as if the Building Block Hypothesis is supported by the Schema Theorem, this cannot be verified easily. Experiments that originally were intended to proof this theory often did not work out as planned [1432] (and also consider the criticisms of the Schema Theorem mentioned in the previous section). In general, there exists much criticism of the Building Block Hypothesis and, although it is a very nice model, it cannot yet be considered as proven sufficiently.

3.7 The Messy Genetic Algorithm

According to the schema theorem specified in Equation 3.10 and Equation 3.11, a schema is likely to spread in the population if it has above-average fitness, is short (i. e., low defined length) and is of low order [116]. Thus, according to Equation 3.11, from two schemas of the same average fitness and order, the one with the lesser defined length will be propagated to more offspring, since it is less likely to be destroyed by crossover. Therefore, placing dependent genes close to each other would be a search space design approach since it will allow good building blocks to proliferate faster. These building blocks, however, are not known at design time – otherwise the problem would already be solved. Hence, it is not generally possible to devise such a design.

The *messy genetic algorithms* (mGAs) developed by Goldberg et al. [825] use a coding scheme which is intended to allow the genetic algorithm to re-arrange genes at runtime. It can place the genes of a building block spatially close together. This method of *linkage learning* may thus increase the probability that these building blocks, i.e., sets of epistatically linked genes, are preserved during crossover operations, as sketched in Figure 3.9. It thus mitigates the effects of epistasis as discussed in Section 1.4.6.

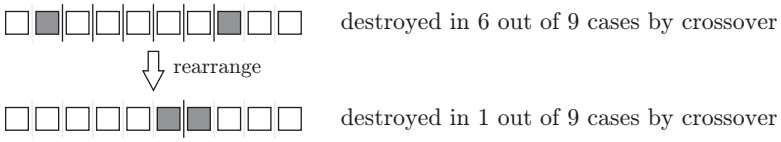


Figure 3.9: Two linked genes and their destruction probability under single-point crossover.

3.7.1 Representation

The idea behind the genomes used in messy GAs goes back to the work Bagley [116] from 1967 who first introduced a representation where the ordering of the genes was not fixed. Instead, for each gene a tuple (ϕ, γ) with its position (locus) ϕ and value (allele) γ was used. For instance, the bit string `000111` can be represented as $g_1 = ((0,0),(1,0),(2,0),(3,1),(4,1),(5,1))$ but as well as $g_2 = ((5,1),(1,0),(3,1),(2,0),(0,0),(4,1))$ where both genotypes map to the same phenotype, i. e., $\text{gpm}(g_1) = \text{gpm}(g_2)$.

3.7.2 Reproduction Operations

Inversion: Unary Reproduction

The *inversion* operator reverses the order of genes between two randomly chosen loci [116, 896]. With this operation, any particular ordering can be produced in a relatively small number of steps. Figure 3.10 illustrates, for example, how the possible building block components $(1, 0)$, $(3, 0)$, $(4, 0)$, and $(6, 0)$ can be brought together in two steps. Nevertheless, the effects of the inversion operation were rather disappointing [116, 741].

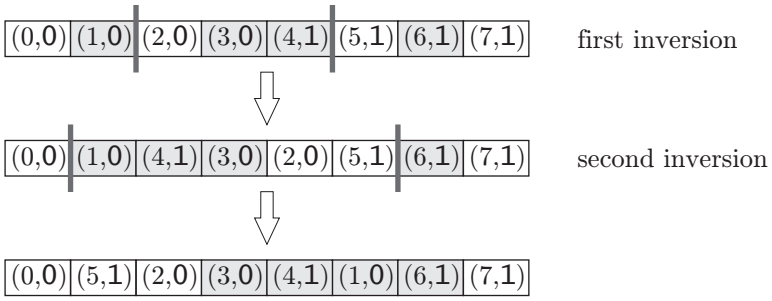


Figure 3.10: An example for two subsequent applications of the inversion operation [896].

Cut: Unary Reproduction

The *cut* operator splits a genotype g into two with the probability $p_c = (\text{len}(g) - 1)\, p_K$ where p_K is a bitwise probability and $\text{len}(g)$ the length of the genotype [1153]. With $p_k = 0.1$, the $g_1 = ((0, 0), (1, 0), (2, 0), (3, 1), (4, 1), (5, 1))$ has a cut probability of $p_c = (6 - 1)*0.1 = 0.5$. A cut at position 4 would lead to $g_3 = ((0, 0), (1, 0), (2, 0), (3, 1))$ and $g_4 = ((4, 1), (5, 1))$.

3.7.3 Splice: Binary Reproduction

The *splice* operator joins two genotypes with a predefined probability p_s by simply attaching one to the other [1153]. Splicing $g_2 = ((5, 1), (1, 0), (3, 1), (2, 0), (0, 0), (4, 1))$ and $g_4 = ((4, 1), (5, 1))$, for instance, leads to $g_5 = ((5, 1), (1, 0), (3, 1), (2, 0), (0, 0), (4, 1), (4, 1), (5, 1))$. In summary, the application of two cut and a subsequent splice operation to two genotypes has roughly the same effect as a single-point crossover operator in variable-length string chromosomes Section 3.5.3.

3.7.4 Overspecification and Underspecification

The genotypes in messy GAs have a variable length and the cut and splice operators can lead to genotypes being over or underspecified. If we assume a three bit genome, the genotype $g_6 = ((2, 0), (0, 0), (2, 1), (1, 0))$ is overspecified since it contains two (in this example, different) alleles for the third gene (at locus 2). $g_7 = ((2, 0), (0, 0))$, in turn, is underspecified since it does not contain any value for the gene in the middle (at locus 1).

Dealing with overspecification is rather simple [1153, 608]: The genes are processed from left to right during the genotype-phenotype mapping, and the first allele found for a specific locus wins. In other words, g_6 from above codes for `000` and the second value for locus 2 is discarded. The loci left open during the interpretation of underspecified genes are filled with values from a template string [1153]. If this string was `000`, g_7 would code for `000`, too.

3.7.5 The Process

In a simple genetic algorithm, building blocks are identified and recombined simultaneously, which leads to a race between recombination and selection [896]. In the messy GA [825, 826], this race is avoided by separating the evolutionary process into two stages:

1. In the *primordial phase*, building blocks are identified. In the original conception of the messy GA, all possible building blocks of a particular order k are generated. Via selection, the best ones are identified and spread in the population.
2. These building blocks are recombined with the cut and splice operators in the subsequent *juxtapositional phase*.

The complexity of the original mGA needed a bootstrap phase in order to identify the order-k building blocks which required to identify the order-$k-1$ blocks first. This bootstrapping was done by applying the primordial and juxtapositional phases for all orders from 1 to $k-1$. This process was later improved by using a probabilistic complete initialization algorithm [828] instead.

3.8 Genotype-Phenotype Mappings and Artificial Embryogeny

As already stated a dozen times by now, genetic algorithms use string genomes to encode the phenotypes x that represent the possible solutions. These phenotypes, however, do not necessarily need to be one-dimensional strings too. Instead, they can be construction plans, circuit layouts, or trees[14]. The process of translating genotypes into corresponding phenotypes is called genotype-phenotype mapping and has been introduced in Definition 1.30 on page 44.

Embryogenesis is the natural process in which the embryo forms and develops[15] and to which the genotype-phenotype mapping in genetic algorithms and Genetic Programming corresponds. Most of even the more sophisticated of these mappings are based on an implicit one-to-one relation in terms of complexity. In the Grammar-guided Genetic Programming approach Gads[16], for example, a single gene encodes (at most) the application of a single grammatical rule, which in turn unfolds a single node in a tree.

Embryogeny in nature is much more complex. Among other things, the DNA, for instance, encodes the structural design information of the human brain. As pointed out by Manos et al. [1358], there are only about 30 thousand active genes in the human genome (2800 million amino acids) for over 100 trillion neural connections in our cerebrum. A huge manifold of information is hence decoded from "data" which is of a much lower magnitude. This is possible because the same genes can be reused in order to repeatedly create the same pattern. The layout of the light receptors in the eye, for example, is always the same – just their wiring changes.

Definition 3.5 (Artificial Embryogeny). We subsume all methods of transforming a genotype into a phenotype of (much) higher complexity under the subject of *artificial embryogeny* [1358, 1957, 192] (also known as computational embryogeny [1221, 259]).

Two different approaches are common in artificial embryogeny: constructing the phenotype by using a grammar to translate the genotype and expanding it step by step until

[14] See for example Section 4.5.6 on page 181

[15] `http://en.wikipedia.org/wiki/Embryogenesis` [accessed 2007-07-03]

[16] See Section 4.5.5 on page 179 for more details.

a terminal state is reached or simulating chemical processes. Both methods may also require subsequent correction steps that ensure that the produced results are correct, which is also common in normal genotype-phenotype mappings [2295]. An example for gene reuse is the genotype-phenotype mapping performed in Grammatical Evolution which is discussed in Section 4.5.6 on page 182.

4

Genetic Programming

4.1 Introduction

The term *Genetic Programming*[1] (GP) [1196, 916] has two possible meanings. First, it is often used to subsume all evolutionary algorithms that have tree data structures as genotypes. Second, it can also be defined as the set of all evolutionary algorithms that breed programs[2], algorithms, and similar constructs. In this chapter, we focus on the latter definition which still includes discussing tree-shaped genomes.

The conventional well-known input-processing-output model[3] from computer science states that a running instance of a program uses its input information to compute and return output data. In Genetic Programming, usually some inputs or situations and corresponding output data samples are known or can be produced or simulated. The goal then is to find a program that connects them or that exhibits some kind of desired behavior according to the specified situations, as sketched in Figure 4.1.

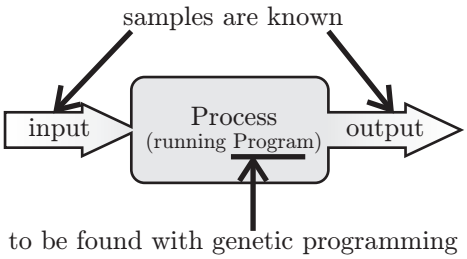


Figure 4.1: Genetic Programming in the context of the IPO model.

4.1.1 History

The history of Genetic Programming [63] goes back to the early days of computer science. In 1957, Friedberg [750] left the first footprints in this area by using a learning algorithm to stepwise improve a program. The program was represented as a sequence of instructions[4] for a theoretical computer called *Herman* [750, 751]. Friedberg did not use an evolutionary, population-based approach for searching the programs. This may be because the idea of

[1] http://en.wikipedia.org/wiki/Genetic_programming [accessed 2007-07-03]
[2] We have extensively discussed the topic of algorithms and programs in Section 30.1.1 on page 547.
[3] see Section 30.1.1 on page 549
[4] Linear Genetic Programming is discussed in Section 4.6 on page 191.

evolutionary algorithms wasn't fully developed yet[5] and also because of the limited computational capacity of the computers of that era.

Around the same time, Samuel applied machine learning to the game of checkers and by doing so, created the world's first self-learning program. In the future development section of his 1959 paper [1795], he suggested that effort could be spent into allowing the (checkers) program to learn scoring polynomials – an activity which would equal symbolic regression. Yet, in his 1967 follow-up work [1797], he could not report any progress in this issue.

The evolutionary programming approach for evolving finite state machines by Fogel et al. [708], discussed in Chapter 6 on page 231, dates back to 1966. In order to build predictors, different forms of mutation (but no crossover) were used for creating offspring from successful individuals.

Fourteen years later, the next generation of scientists began to look for ways to evolve programs. New results were reported by Smith [1912] in his PhD thesis in 1980. Forsyth [733] evolved trees denoting fully bracketed Boolean expressions for classification problems in 1981 [733, 735, 734].

The mid-1980s were a very productive period for the development of Genetic Programming. Cramer [462] applied a genetic algorithm in order to evolve a program written in a subset of the programming language `PL` in 1985.[6] This GA used a string of integers as genome and employed a genotype-phenotype mapping that recursively transformed them into program trees. At the same time, the undergraduate student Schmidhuber [1828] also used a genetic algorithm to evolve programs at the Siemens AG. He re-implemented his approach in `Prolog` at the TU Munich in 1987 [562, 1828]. Hicklin [924] and Fujuki [754] implemented reproduction operations for manipulating the if-then clauses of LISP programs consisting of single COND-statements. With this approach, Fujiko and Dickinson [753] evolved strategies for playing the iterated prisoner's dilemma game. Bickel and Bickel [206] evolved sets of rules which were represented as trees using tree-based mutation crossover operators.

Genetic Programming became fully accepted at the end of this productive decade mainly because of the work of Koza [1183, 1184]. He also studied many benchmark applications of Genetic Programming, such as learning of Boolean functions [1190, 1185], the Artificial Ant problem[7] [1188, 1187, 1196], and symbolic regression[8] [1190, 1196], a method for obtaining mathematical expressions that match given data samples. Koza formalized (and patented [1183, 1194]) the idea of employing genomes purely based on tree data structures rather than string chromosomes as used in genetic algorithms. In symbolic regression, such trees can, for instance, encode `Lisp` S-expressions[9] where a node stands for a mathematical operation and its child nodes are the parameters of the operation. Leaf nodes then are terminal symbols like numbers or variables. This form of Genetic Programming is called *Standard Genetic Programming* or SGP, in short. With it, not only mathematical functions but also more complex programs can be expressed as well.

Generally, a tree can represent a rule set [1389, 1390], a mathematical expressions, a decision tree [1193], or even the blueprint of an electrical circuit [1082]. Trees are very close to the natural structure of algorithms and programs. The syntax of most of the high-level programming languages, for example, leads to a certain hierarchy of modules and alternatives. Not only does this form normally constitute a tree – compilers even use tree representations internally. When reading the source code of a program, they first split it into tokens[10], parse[11] these tokens, and finally create an abstract syntax tree[12] (AST) [1065, 961]. The internal nodes of ASTs are labeled by operators and the leaf nodes contain the operands

[5] Compare with Section 3.1 on page 141.
[6] Cramer's approach is discussed in Section 4.4.1 on page 171.
[7] The Artificial Ant is discussed in Section 21.3.1 on page 354 in this book.
[8] More information on symbolic regression is presented in Section 23.1 on page 397 in this book.
[9] List S-expressions are discussed in Section 30.3.11 on page 571
[10] `http://en.wikipedia.org/wiki/Lexical_analysis` [accessed 2007-07-03]
[11] `http://en.wikipedia.org/wiki/Parse_tree` [accessed 2007-07-03]
[12] `http://en.wikipedia.org/wiki/Abstract_syntax_tree` [accessed 2007-07-03]

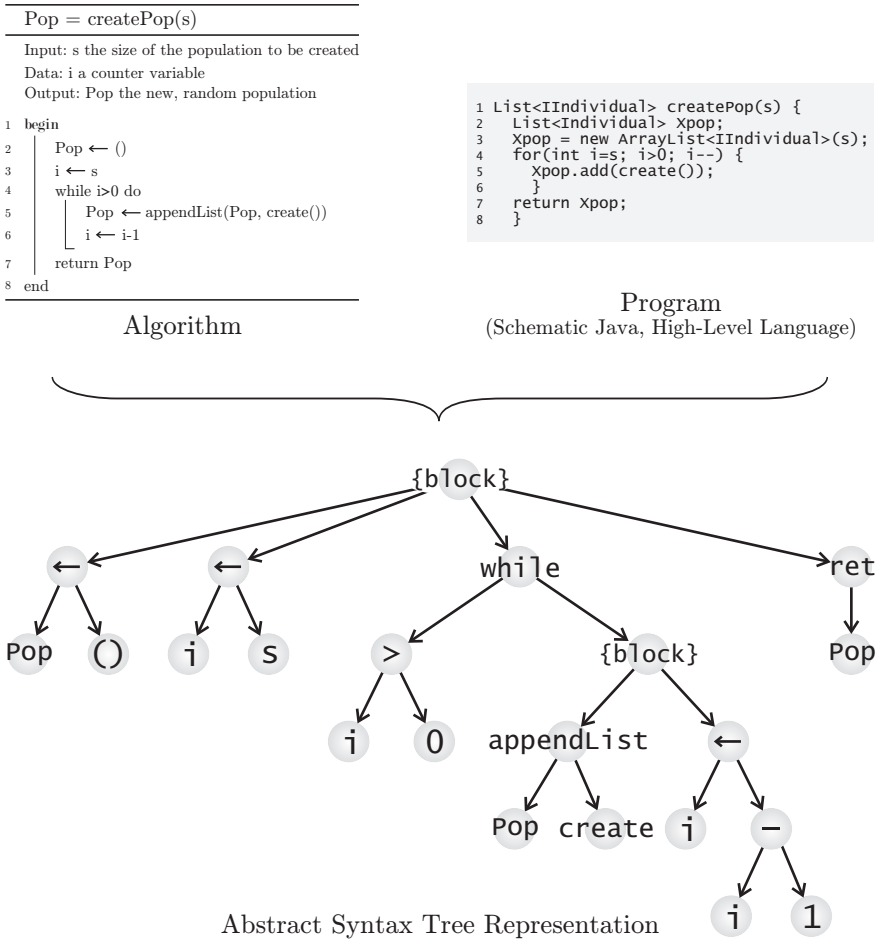


Figure 4.2: The AST representation of algorithms/programs.

of these operators. In principle, we can illustrate almost every[13] program or algorithm as such an AST (see Figure 4.2).

Tree-based Genetic Programming directly evolves individuals in this form, which also provides a very intuitive representation for mathematical functions for which it has initially been used for by Koza. Another interesting aspect of the tree genome is that it has no natural role model. While genetic algorithms match their direct biological metaphor particularly well, Genetic Programming introduces completely new characteristics and traits. Genetic Programming is one of the few techniques that are able to learn solutions of potentially unbound complexity. It can be considered as more general than genetic algorithms, because it makes fewer assumptions about the structure of possible solutions. Furthermore, it often offers white-box solutions that are human-interpretable. Other optimization approaches like artificial neural networks, for example, generate black-box outputs, which are highly complicated if not impossible to fully grasp [1382].

[13] Excluding such algorithms and programs that contain jumps (the infamous "goto") that would produce crossing lines in the flowchart (http://en.wikipedia.org/wiki/Flowchart [accessed 2007-07-03]).

4.2 General Information

4.2.1 Areas Of Application

Some example areas of application of Genetic Programming are:

Application	References
Symbolic Regression and Function Synthesis	[1190, 1196, 87, 2270, 1699, 1196, 17, 528] Section 23.1
Grammar Induction	[1042, 1394, 465, 1174]
Data Mining and Data Analysis	[1186, 744, 1592, 1593, 242, 445, 1193, 2253, 332] Section 22.1.2
Electrical Engineering and Circuit Design	[1082, 1182, 1080, 1206, 1205, 1211, 1669, 506]
Medicine	[2055, 270, 243, 956]
Economics and Finance	[1191, 1513, 1674, 1577]
Geometry and Physics	[1307, 2277]
Cellular Automata and Finite State Machines	[58, 59, 508, 509]
Automated Programming	[140, 1242, 1324, 1325, 1317, 1212]
Robotics	[1201, 1202, 1204, 986, 1317, 57, 1576, 986, 1323]
Networking and Communication	[434, 504, 2180, 1257, 1887, 1888] Section 24.1 on page 413 and Section 23.2 on page 401
Evolving Behaviors, e.g., for Agents or Game Players	[1187, 179, 180, 1688, 1686, 1687, 907, 909, 55, 54, 1933, 67, 1492, 984, 987, 985, 986, 1340, 1341, 1342, 2194, 1323]
Pattern Recognition	[53, 56, 2015, 2014, 2016]
Biochemistry	[1200, 1199]
Machine Learning	[1203, 863]

See also Section 4.4.3 on page 174, Section 4.5.6 on page 184, and Section 4.7.4 on page 201.

4.2.2 Conferences, Workshops, etc.

Some conferences, workshops and such and such on Genetic Programming are:

EuroGP: European Conference on Genetic Programming
http://www.evostar.org/ [accessed 2007-09-05]
Co-located with EvoWorkshops and EvoCOP.
History: 2009: Tübingen, see [2106]
2008: Naples, Italy, see [1579]
2007: Valencia, Spain, see [617]
2006: Budapest, Hungary, see [429]

2005: Lausanne, Switzerland, see [1116]
2004: Coimbra, Portugal, see [1115]
2003: Essex, UK, see [1786]
2002: Kinsale, Ireland, see [737]
2001: Lake Como, Italy, see [1423]
2000: Edinburgh, Scotland, UK, see [1666]
1999: Göteborg, Sweden, see [1664]
1998: Paris, France, see [141, 1663]

GECCO: Genetic and Evolutionary Computation Conference
see Section 2.2.2 on page 107

GP: Annual Genetic Programming Conference
Now part of GECCO, see Section 2.2.2 on page 107
History: 1998: Madison, Wisconsin, USA, see [1209, 1198]
1997: Stanford University, CA, USA, see [1208, 1956]
1996: Stanford University, CA, USA, see [1207, 1197]

GPTP: Genetic Programming Theory Practice Workshop
http://www.cscs.umich.edu/gptp-workshops/ [accessed 2007-09-28]
History: 2007: Ann Arbor, Michigan, USA, see [1945]
2006: Ann Arbor, Michigan, USA, see [1735]
2005: Ann Arbor, Michigan, USA, see [2298]
2004: Ann Arbor, Michigan, USA, see [1583]
2003: Ann Arbor, Michigan, USA, see [1734]

ICANNGA: International Conference on Adaptive and Natural Computing Algorithms
see Section 2.2.2 on page 108

Mendel: International Conference on Soft Computing
see Section 1.6.2 on page 90

4.2.3 Journals

Some journals that deal (at least partially) with Genetic Programming are:

Genetic Programming and Evolvable Machines (GPEM), ISSN: 1389-2576 (Print) 1573-7632 (Online), appears quaterly, editor(s): Wolfgang Banzhaf, publisher: Springer Netherlands, http://springerlink.metapress.com/content/104755/ [accessed 2007-09-28]

4.2.4 Online Resources

Some general, online available ressources on Genetic Programming are:

http://www.genetic-programming.org/ [accessed 2007-09-20] and http://www.genetic-programming.com/ [accessed 2007-09-20]
Last update: up-to-date
Description: Two portal pages on Genetic Programming websites, both maintained by Koza.

http://www.cs.bham.ac.uk/~wbl/biblio/ [accessed 2007-09-16]
Last update: up-to-date
Description: Langdon's large Genetic Programming bibliography.

http://www.lulu.com/items/volume_63/2167000/2167025/2/print/book.pdf [accessed 2008-03-26]

Last update: up-to-date
Description: A Field Guide to Genetic Programming, see [1667]

http://www.aaai.org/AITopics/html/genalg.html [accessed 2008-05-17]
Last update: up-to-date
Description: The genetic algorithms and Genetic Programming pages of the AAAI

http://www.cs.ucl.ac.uk/staff/W.Langdon/www_links.html [accessed 2008-05-18]
Last update: 2007-07-28
Description: William Langdon's Genetic Programming contacts

4.2.5 Books

Some books about (or including significant information about) Genetic Programming are:

Koza [1196]: *Genetic Programming, On the Programming of Computers by Means of Natural Selection*
Poli, Langdon, and McPhee [1667]: *A Field Guide to Genetic Programming*
Koza [1195]: *Genetic Programming II: Automatic Discovery of Reusable Programs: Automatic Discovery of Reusable Programs*
Koza, Bennett III, Andre, and Keane [1210]: *Genetic Programming III: Darwinian Invention and Problem Solving*
Koza, Keane, Streeter, Mydlowec, Yu, and Lanza [1212]: *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*
Langdon and Poli [1242]: *Foundations of Genetic Programming*
Langdon [1238]: *Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming!*
Banzhaf, Nordin, Keller, and Francone [140]: *Genetic Programming: An Introduction – On the Automatic Evolution of Computer Programs and Its Applications*
Kinnear, Jr. [1140]: *Advances in Genetic Programming, Volume 1*
Angeline and Kinnear, Jr [61]: *Advances in Genetic Programming, Volume 2*
Spector, Langdon, O'Reilly, and Angeline [1936]: *Advances in Genetic Programming, Volume 3*
Brameier and Banzhaf [275]: *Linear Genetic Programming*
Wong and Leung [2253]: *Data Mining Using Grammar Based Genetic Programming and Applications*
Geyer-Schulz [795]: *Fuzzy Rule-Based Expert Systems and Genetic Machine Learning*
Spector [1932]: *Automatic Quantum Computer Programming – A Genetic Programming Approach*
Nedjah, Abraham, and de Macedo Mourelle [1511]: *Genetic Systems Programming: Theory and Experiences*

4.3 (Standard) Tree Genomes

Tree-based Genetic Programming (TGP), usually referred to as Standard Genetic Programming, SGP) is the most widespread Genetic Programming variant, both for historical reasons and because of its efficiency in many problem domains. In this section, the well-known reproduction operations applicable to tree genomes are outlined.

4.3.1 Creation: Nullary Reproduction

Before the evolutionary process can begin, we need an initial, randomized population. In genetic algorithms, we therefore simply created a set of random bit strings. For Genetic Programming, we do the same with trees instead of such one-dimensional sequences.

Normally, there is a maximum depth d specified that the tree individuals are not allowed to surpass. Then, the creation operation will return only trees where the path between the root and the most distant leaf node is not longer than d. There are three different ways for realizing the "create()" operation (see Definition 2.9 on page 137) for trees which can be distinguished according to the depth of the produced individuals.

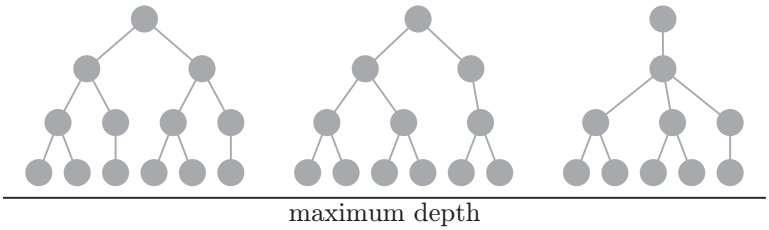


Figure 4.3: Tree creation by the *full* method.

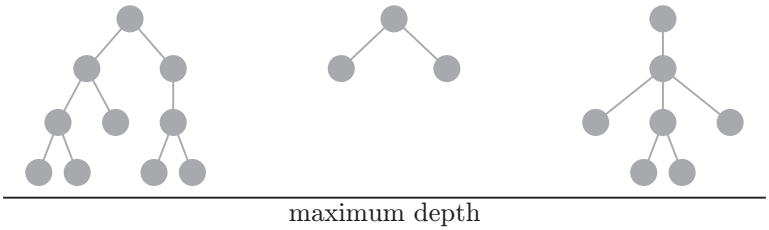


Figure 4.4: Tree creation by the *grow* method.

The *full* method (Figure 4.3) creates trees where each (non-backtracking) path from the root to the leaf nodes has exactly the length d. The *grow* method depicted in Figure 4.4, also creates trees where each (non-backtracking) path from the root to the leaf nodes is not longer than d *but* may be shorter. This is achieved by deciding randomly for each node if it should be a leaf or not when it is attached to the tree. Of course, to nodes of the depth $d-1$, only leaf nodes can be attached to.

Koza [1196] additionally introduced a mixture method called *ramped half-and-half*. For each tree to be created, this algorithm draws a number r uniformly distributed between 2 and d: $(r = \lfloor random 2d + 1 \rfloor)$. Now either *full* or *grow* is chosen to finally create a tree with the maximum depth r (in place of d). This method is often preferred since it produces an especially wide range of different tree depths and shapes and thus provides a great initial diversity.

4.3.2 Mutation: Unary Reproduction

Tree genotypes may undergo small variations during the reproduction process in the evolutionary algorithm. Such a *mutation* is usually defined as the random selection of a node in the tree, removing this node and all of its children, and finally replacing it with another node [1196]. From this idea, three operators can be derived:

1. replacement of existing nodes randomly created ones (Fig. 4.5.a),
2. insertions of new nodes or small trees (Fig. 4.5.b), and
3. the deletion of nodes, as illustrated in Fig. 4.5.c.

The effects of insertion and deletion can also be achieved with replacement.

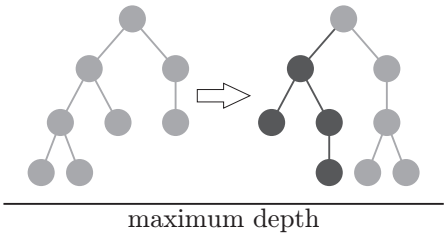


Fig. 4.5.a: Sub-tree replacement.

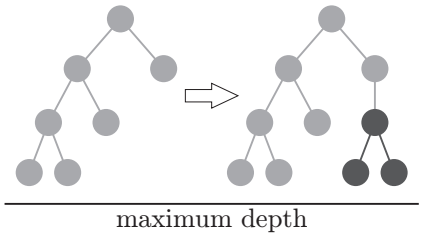


Fig. 4.5.b: Sub-tree insertions.

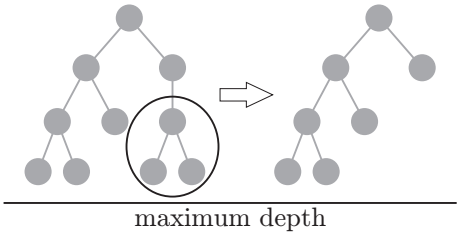


Fig. 4.5.c: Sub-tree deletion.

Figure 4.5: Possible tree mutation operations.

4.3.3 Recombination: Binary Reproduction

The mating process in nature – the recombination of the genotypes of two individuals – is also copied in tree-based Genetic Programming. Applying the default sub-tree exchange recombination operator to two trees means to swap sub-trees between them as illustrated in Figure 4.6. Therefore, one single sub-tree is selected randomly from each of the parents and subsequently are cut out and reinserted in the partner genotype. Notice that, like in genetic algorithms, the effects of insertion and deletion operations can also be achieved by recombination.

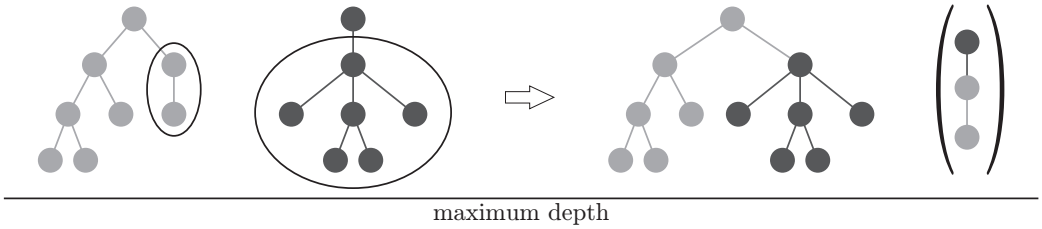


Figure 4.6: Tree crossover by exchanging sub-trees.

If a depth restriction is imposed on the genome, both, the mutation and the crossover operation have to respect them. The new trees they create must not exceed it.

The intent of using the recombination operation in Genetic Programming is the same as in genetic algorithms. Over many generations, successful building blocks – for example a highly fit expression in a mathematical formula – should spread throughout the population and be combined with good genes of different solution candidates. Yet, recombination in Standard Genetic Programming can also have a very destructive effect on the individual fitness [1525, 1544, 140]. Angeline [62] even argues that it performs no better than mutation and causes bloat [65].

Several techniques have been proposed in order to mitigate these effects. In 1994, D'Haeseleer [557] obtained modest improvements with his strong context preserving crossover that permitted only the exchange of sub-trees that occupied the same positions in the parents. Poli and Langdon [1661, 1662] define the similar single-point crossover for tree genomes with the same purpose: increasing the probability of exchanging genetic material which is structural and functional akin and thus decreasing the disruptiveness. A related approach define by Francone et al. [740] for linear Genetic Programming is discussed in Section 4.6.7 on page 195.

4.3.4 Permutation: Unary Reproduction

The tree permutation operation illustrated in Figure 4.7 resembles the permutation operation of string genomes or the inversion used in messy GA (Section 3.7.2, [1196]). Like mutation, it is used to reproduce one single tree. It first selects an internal node of the parental tree. The child nodes attached to that node are then shuffled randomly, i. e., permutated. If the tree represents a mathematical formula and the operation represented by the node is commutative, this has no direct effect. The main goal is to re-arrange the nodes in highly fit sub-trees in order to make them less fragile for other operations such as recombination. The effects of this operation are doubtable and most often it is not applied [1196].

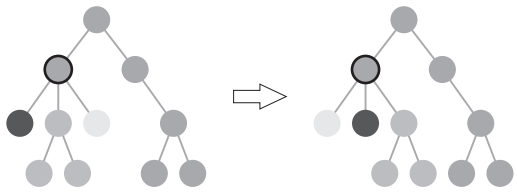


Figure 4.7: Tree permutation – (asexually) shuffling sub-trees.

4.3.5 Editing: Unary Reproduction

Editing trees in Genetic Programming is what simplifying is to mathematical formulas. Take $x = b + (7-4) + (1*a)$ for instance. This expression clearly can be written in a shorter way be replacing $(7-4)$ with 3 and $(1*a)$ with a. By doing so, we improve its readability and also decrease the computational time for concrete values of a and b. Similar measures can often be applied to algorithms and program code. Editing a tree as outlined in Figure 4.8 means to create a new offspring tree which is more efficient but, in terms of functional aspects, equivalent to its parent. It is thus a very domain-specific operation.

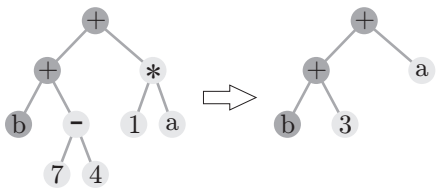


Figure 4.8: Tree editing – (asexual) optimization.

A positive aspect of editing is that it usually reduces the number of nodes in a tree by removing useless expression, for instance. This makes it more easy for recombination

operations to pick "important" building blocks. At the same time, the expression $(7-4)$ is now less likely to be destroyed by the reproduction processes since it is replaced by the single terminal node 3.

On the other hand, editing also reduces the diversity in the genome which could degrade the performance by decreasing the variety of structures available. Another negative aspect would be if (in our example) a fitter expression was $(7-(4*a))$ and a is a variable close to 1. Then, transforming $(7-4)$ into 3 prevents a transition to the fitter expression.

In Koza's experiments, Genetic Programming with and without editing showed equal performance, so this operation is not necessarily needed [1196].

4.3.6 Encapsulation: Unary Reproduction

The idea behind the encapsulation operation is to identify potentially useful sub-trees and to turn them into atomic building block as sketched in Figure 4.9. To put it plain, we create new terminal symbols that (internally hidden) are trees with multiple nodes. This way, they will no longer be subject to potential damage by other reproduction operations. The new terminal may spread throughout the population in the further course of the evolution. According to Koza, this operation has no substantial effect but may be useful in special applications like the evolution of artificial neural networks [1196].

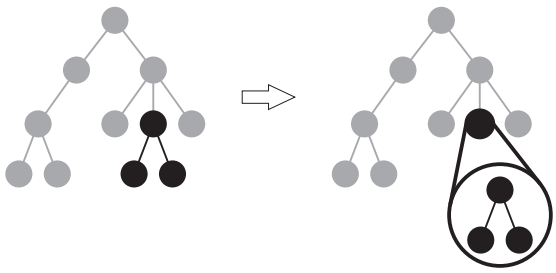


Figure 4.9: An example for tree encapsulation.

4.3.7 Wrapping: Unary Reproduction

Applying the wrapping operation means to first select an arbitrary node n in the tree. Additionally, we create a new non-terminal node m outside of the tree. In m, at least one child node position is left unoccupied. We then cut n (and all its potential child nodes) from the original tree and append it to m by plugging it into the free spot. Now we hang m into the tree position that formerly was occupied by n.

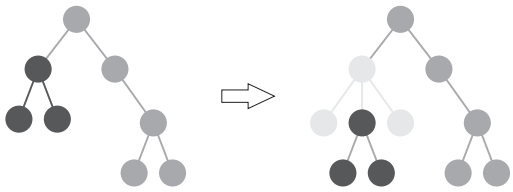


Figure 4.10: An example for tree wrapping.

The purpose of this reproduction method illustrated in Figure 4.10 is to allow modifications of non-terminal nodes that have a high probability of being useful. Simple mutation would, for example, cut n from the tree or replace it with another expression. This will always change the meaning of the whole sub-tree below n dramatically, like for example in $(b{+}3) + a \longrightarrow (b{*}3) + a$. By wrapping however, a more subtle change like $(b{+}3) + a \longrightarrow ((b{+}\mathbf{1}){+}3) + a$ is possible.

The wrapping operation is introduced by the author – at least, I have not seen another source where it is used.

4.3.8 Lifting: Unary Reproduction

While wrapping allows nodes to be inserted in non-terminal positions with small change of the tree's semantic, lifting is able to remove them in the same way. It is the inverse operation to wrapping, which becomes obvious when comparing Figure 4.10 and Figure 4.11.

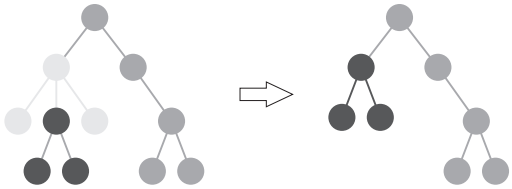


Figure 4.11: An example for tree lifting.

Lifting begins with selecting an arbitrary inner node n of the tree. This node then replaces its parent node. The parent node inclusively all of its child nodes (except n) are removed from the tree. With lifting, a tree that represents the mathematical formula $(b + (1 - a)) * 3$ can be transformed to $b * 3$ in a single step. Lifting is used by the author in his experiments with Genetic Programming (see for example Section 24.1.2 on page 414). I, however, have not yet found other sources using a similar operation.

4.3.9 Automatically Defined Functions

The concept of automatically defined functions (ADFs) introduced by Koza [1196] provides some sort of pre-specified modularity for Genetic Programming. Finding a way to evolve modules and reusable building blocks is one of the key issues in using GP to derive higher-level abstractions and solutions to more complex problems [66, 67, 1195]. If ADFs are used, a certain structure is defined for the genome. The root of the tree usually loses its functional responsibility and now serves only as glue that holds the individual together and has a fixed number n of children, from which $n - 1$ are automatically defined functions and one is the result-generating branch. When evaluating the fitness of an individual, often only this first branch is taken into consideration whereas the root and the ADFs are ignored. The result-generating branch, however, may use any of the automatically defined functions to produce its output.

When ADFs are employed, typically not only their number must be specified beforehand but also the number of arguments of each of them. How this works can maybe best illustrated by using the example given in Figure 4.12. It stems from function approximation[14], since this is the area where many early examples of the idea of ADFs come from.

Assume that the goal of GP is to approximate a function g with the one parameter x and that a genome is used where two functions (f_0 and f_1) are automatically defined. f_0

[14] A very common example for function approximation, Genetic Programming-based symbolic regression, is discussed in Section 23.1 on page 397.

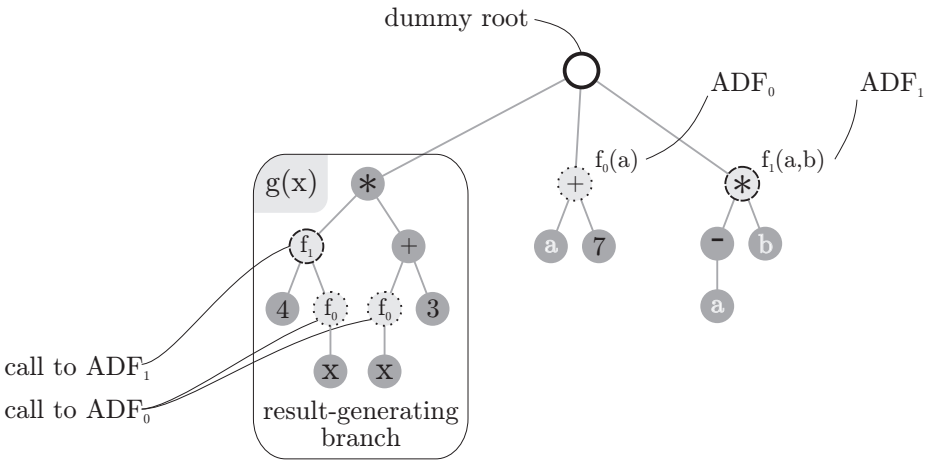


Figure 4.12: A concrete example for automatically defined functions.

has a single formal parameter a and f_1 has two formal parameters a and b. The genotype Figure 4.12 encodes the following mathematical functions:

$$\begin{aligned} g(x) &= f_1(4, f_0(x)) * (f_0(x) + 3) \\ f_0(a) &= a + 7 \\ f_1(a, b) &= (-a) * b \end{aligned}$$

Hence, $g(x) \equiv ((-4) * (x + 7)) * ((x + 7) + 3)$. The number of children of the function calls in the result-generating branch must be equal to the number of the parameters of the corresponding ADF.

Although ADFs were first introduced in symbolic regression by Koza [1196], they can also be applied to a variety of other problems like in the evolution of agent behaviors [1688, 1686, 52, 55], electrical circuit design [1206], or the evolution of robotic behavior [57].

4.3.10 Automatically Defined Macros

Spector's idea of automatically defined macros (ADMs) complements the ADFs of Koza [1928, 1929]. Both concepts are very similar and only differ in the way that their parameters are handled. The parameters in automatically defined functions are always values whereas automatically defined macros work on code expressions. This difference shows up only when side-effects come into play.

In Figure 4.13, we have illustrated the pseudo-code of two programs – one with a function (called `ADF`) and one with a macro (called `ADM`). Each program has a variable `x` which is initially zero. The function `y()` has the side-effect that it increments `x` and returns its new value. Both, the function and the macro, return a sum containing their parameter `a` two times. The parameter of `ADF` is evaluated *before* `ADF` is invoked. Hence, `x` is incremented one time and `1` is passed to `ADF` which then returns `2=1+1`. The parameter of the macro, however, is the invocation of `y()`, not its result. Therefore, the `ADM` resembles to two calls to `y()`, resulting in `x` being incremented two times and in `3=1+2` being returned.

The ideas of automatically defined macros and automatically defined functions are very close to each other. Automatically defined macros are likely to be useful in scenarios where context-sensitive or side-effect-producing operators play important roles [1928, 1929]. In other scenarios, there is no much difference between the application of ADFs and ADMs. Finally, it should be mentioned that the concepts of automatically defined functions and

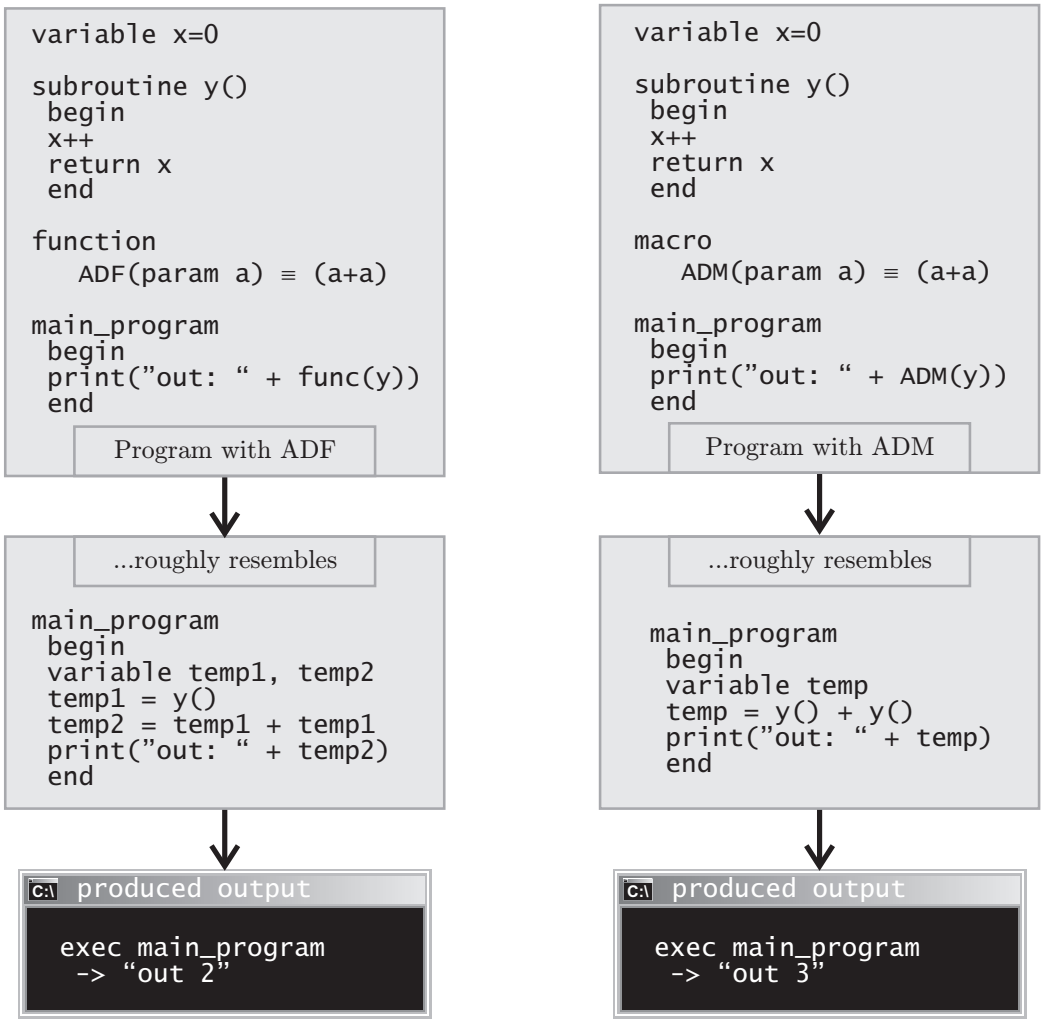


Figure 4.13: Comparison of functions and macros.

macros are not restricted to the standard tree genomes but are also applicable in other forms of Genetic Programming, such as linear Genetic Programming or PADO.[15]

4.3.11 Node Selection

In most of the reproduction operations for tree genomes, in mutation as well as in recombination, certain nodes in the trees need to be selected. In order to apply the mutation, we first need to find the node which is to be altered. For recombination, we need one node in each parent tree. These nodes are then exchanged. The question how to select these nodes seems to be more or less irrelevant but plays an important role in reality. The literature most often speaks of "randomly selecting" a node but does not describe how exactly this should be done.

A good method for doing so could select all nodes c and n in the tree t with exactly the same probability as done by the method "uniformSelectNode", i. e., $P(\text{uniformSelectNode}(t) = c) = P(\text{uniformSelectNode}(t) = n)\ \forall s, n \in t$.

[15] Linear Genetic Programming is discussed in Section 4.6 on page 191 and a summary on PADO can be found in Section 4.7.1 on page 196.

Therefore, we define the weight nodeWeight(n) of a tree node n to be the total number of nodes in the sub-tree with n as root, i. e., itself, its children, grandchildren, grand-grandchildren, etc.

$$\text{nodeWeight}(n) = 1 + \sum_{i=0}^{\text{len}(n.\mathit{children})-1} \text{nodeWeight}(n.\mathit{children}_{[i]}) \tag{4.1}$$

Thus, the nodeWeight of the root of a tree is the number of all nodes in the tree and the nodeWeight of each of the leaves is exactly 1. In uniformSelectNode, the probability for a node of being selected in a tree t is thus $1/\text{nodeWeight}(t)$. We can create such a probability distribution by descending it from the root according to Algorithm 4.1.

Algorithm 4.1: $n \longleftarrow$ uniformSelectNode(t)

Input: t: the (root of the) tree to select a node from
Data: c: the currently investigated node
Data: $c.\mathit{children}$: the list of child nodes of c
Data: b, d: two Boolean variables
Data: r: a value uniformly distributed in $[0, \text{nodeWeight}(c)]$
Data: i: an index
Output: n: the selected node

```
1  begin
2      b ⟵ true
3      c ⟵ t
4      while b do
5          r ⟵ ⌊random_u(0, nodeWeight(c))⌋
6          if r ≥ nodeWeight(c) − 1 then b ⟵ false
7          else
8              i ⟵ len(c.children) − 1
9              while i ≥ 0 do
10                 r ⟵ r − nodeWeight(c.children[i])
11                 if r < 0 then
12                     c ⟵ c.children[i]
13                     i ⟵ −1
14                 else
15                     i ⟵ i − 1
16     return c
17 end
```

A tree descend where with probabilities different from these defined here will lead to unbalanced node selection probability distributions. Then, the reproduction operators will prefer accessing some parts of the trees while very rarely altering the other regions. We could, for example, descend the tree by starting at the root t and would return the current node with probability 0.5 or recursively go to one of its children (also with 50% probability). Then, the root t would have a 50 : 50 chance of being the starting point of reproduction operation. Its direct children have at most probability $0.5^2/\text{len}(t.\mathit{children})$ each, and their children even $0.5^3/\text{len}(t.\mathit{children})\text{len}(t.\mathit{children}_{[i]}.\mathit{children})$ and so on. Hence, the leaves would almost never take actively part in reproduction. We could also choose other probabilities which strongly prefer going down to the children of the tree, but then, the nodes near to the root will most likely be left untouched during reproduction. Often, this approach is favored by selection methods, although leaves in different branches of the tree are not chosen with the same probabilities if the branches differ in depth. When applying Algorithm 4.1 on the other hand, there exist no regions in the trees that have lower selection probabilities than others.

4.4 Genotype-Phenotype Mappings

Genotype-phenotype mappings (GPM, see Section 3.8 on page 155) are used in many different Genetic Programming approaches. Here we give a few examples about them. Many of the Grammar-guided Genetic Programming approaches discussed in Section 4.5 on page 176 are based on similar mappings.

4.4.1 Cramer's Genetic Programming

It is interesting to see that the earliest Genetic Programming approaches were based on a genotype-phenotype mapping. One of them, dating back to 1985, is the method of Cramer [462]. His goal was to evolve programs in a modified subset of the programming language PL. Two simple examples for such programs, obtained from his work, are:

```
;;Set variable V0 to have the value of V1
(:ZERO V0)
(:LOOP V1 (:INC V0))

;;Multiply V3 by V4 and store the result in V5
(:ZERO V5)
(:LOOP V3 (:LOOP V4 (:INC V5)))
```

Listing 4.1: Two examples for the PL dialect used by Cramer for Genetic Programming

On basis of a genetic algorithm working on integer strings, he proposed two ideas on how to convert these strings to valid program trees.

The JB Mapping

The first approach was to divide the integer string into tuples of a fixed length which is large enough to hold the information required to encode an arbitrary instruction. In the case our examples, these are triplets where the first item identifies the operation, and the following two numbers define its parameters. Superfluous information, like a second parameter for a unary operation, is ignored.

```
(0  4 2) → (:BLOCK AS4 AS2)
(1  6 0) → (:LOOP V6 AS0)
(2  1 9) → (:SET V1 V9)
(3 17 8) → (:ZERO V17) ;;the 8 is ignored
(4  0 5) → (:INC V0)   ;;the 5 is ignored
```

Listing 4.2: An example for the JB Mapping

Here, the symbols of the form Vn and ASn represent variables and auxiliary statements, respectively. Cramer distinguishes between input variables providing data to a program and local (body) variables used for computation. Any of them can be chosen as output variable at the end of the execution. The multiplication program used in Listing 4.1 can now be encoded as `(0 0 1 3 5 8 1 3 2 1 4 3 4 5 9 9 2)` which translates to

```
(0 0 1) ;;main statement       → (:BLOCK AS0 AS1)
(3 5 8) ;;auxiliary statement 0 → (:ZERO V5)
(1 3 2) ;;auxiliary statement 1 → (:LOOP V3 AS2)
(1 4 3) ;;auxiliary statement 2 → (:LOOP V4 AS3)
(4 5 9) ;;auxiliary statement 3 → (:INC V5)
```

Listing 4.3: Another example for the JP Mapping

Cramer outlines some of the major problems of this representation, especially the strong positional epistasis[16] – the strong relation of the meaning of an instruction to its position. This epistasis makes it very hard for the genetic operations to work efficiently, i.e., to prevent destruction of the genotypes passed to them.

[16] We come back to positional epistasis in Section 4.8.1 on page 202.

The TB Mapping

The TB mapping is essentially the same as the JB mapping, but reduces these problems a bit. Instead of using the auxiliary statement method as done in JB, the expressions in the TB language are decoded recursively. The string `(0 (3 5)(1 3 (1 4 (4 5))) )`, for instance, expands to the program tree illustrated in Listing 4.3. Furthermore, Cramer restricts mutation to the statements near the fringe of the tree, more specifically, to leaf operators that do not require statements as arguments and to non-leaf operations with leaf statements as arguments. Similar restrictions apply to crossover.

4.4.2 Binary Genetic Programming

With their Binary Genetic Programming (BGP) approach [136], Keller and Banzhaf [1119, 1120, 1121] further explore the utility of explicit genotype-phenotype mappings and neutral variations in the genotypes. They called the genes in their fixed-length binary string genome *codons* analogously to molecular biology where a codon is a triplet of nucleic acids in the DNA[17], encoding one amino acid at most. Each codon corresponds to one symbol in the target language. The translation of the binary string genotype g into a string representing an expression in the target language works as follows:

1. $x \longleftarrow \varepsilon$
2. Take the next gene (codon) `g` from g and translate it to the according symbol s.
3. If s is a valid continuation of x, set $x \longleftarrow x \circ s$ and continue in step 2.
4. Otherwise, compute the set of symbols S that would be valid continuation of x.
5. From this set, extract the set of (valid) symbols S' which have the minimal Hamming distance[18] to the codon `g`.
6. From S' take the symbol s' which has the minimal codon value and append it to x: $x \longleftarrow x \circ s'$.

After this mapping, x can still be an invalid expression since there maybe were not enough genes in g so the phenotype is incomplete, for example $x = 3 * 4 - sin(v*$. These incomplete sequences are fixed by consecutively appending symbols that lead to a quick end of an expression according to some heuristic.

The genotype-phenotype mapping of Binary Genetic Programming represents a $n : 1$ relation: Due to the fact that different codons may be replaced by the same approximation, multiple genotypes have the same phenotypic representation. This also means that there can be genetic variations induced by the mutation operation that do not influence the fitness. Such neutral variations are often considered as a driving force behind (molecular) evolution [1137, 1138, 973] and are discussed in Section 1.4.5 on page 67 in detail.

From the form of the genome we assume the number of corrections needed in the genotype-phenotype mapping(especially for larger grammars) will be high. This, in turn, could lead to very destructive mutation and crossover operations since if one codon is modified, the semantics of many subsequent codons may be influenced wildly. This issue is also discussed in Section 4.8.1 on page 204.

4.4.3 Gene Expression Programming

Gene Expression Programming (GEP) by Ferreira [654, 655, 656, 657, 658] introduces an interesting method for dealing with remaining unsatisfied function arguments at the end of the expression tree building process. Like BGP, Gene Expression Programming uses a genotype-phenotype mapping that translates fixed-length string chromosomes into tree phenotypes representing programs.

[17] See Figure 1.14 on page 42 for more information on the DNA.
[18] see Definition 29.6 on page 537

A gene in GEP is composed of a head and a tail [654] which are further divided into codons, where each codon directly encodes one expression. The codons in the head of a gene can represent arbitrary expressions whereas the codons in the tail can only stand for parameterless terms. This makes the tail a reservoir for unresolved arguments of the expressions in the head.

For each problem, the length h of the head is chosen as a fixed value, and the length of the tail t is defined according to Equation 4.2, where n is the arity (the number of arguments) of the function with the most arguments.

$$t = h(n-1)+1 \tag{4.2}$$

The reason for this formula is that we have h expressions in the head, each of them taking at most n parameters. An upper bound for the total number of arguments is thus $h*n$. From this number, $h-1$ are already satisfied since all expressions in the head (except for the first one) themselves are arguments to expressions instantiated before. This leaves at most $h*n-(h-1) = h*n-h+1 = h(n-1)+1$ unsatisfied parameters. With this simple measure, incomplete expressions that require additional repair operations in BGP and most other approaches simply cannot occur.

For instance, consider the grammar for mathematical expressions with the terminal symbols $\Sigma = \{\sqrt{\cdot}, \texttt{*}, \texttt{/}, \texttt{-}, \texttt{+}, \texttt{a}, \texttt{b}\}$ given as example in [654]. It includes two variables, `a` and `b`, as well as five mathematical functions, $\sqrt{\cdot}$, `*`, `/`, `+`, and `-`. $\sqrt{\cdot}$ has the arity 1 since it takes one argument, the other four have arity 2. Hence, $n = 2$.

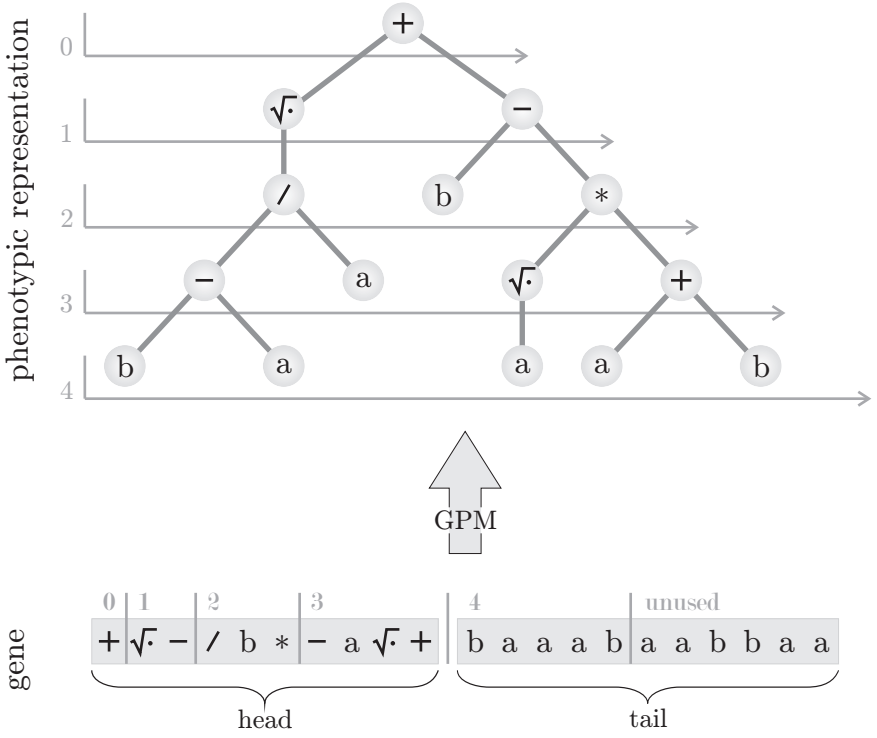


Figure 4.14: A GPM example for Gene Expression Programming.

Figure 4.14 illustrates an example gene (with $h = 10$ and $t = h(2-1)+1 = 11$) and its phenotypic representation of this mathematical expression grammar. A phenotype is built by interpreting the gene as a level-order traversal[19] of the nodes of the expression tree. In

[19] `http://en.wikipedia.org/wiki/Tree_traversal` [accessed 2007-07-15]

other words, the first codon of a gene encodes the root r of expression tree (here +). Then, all nodes in the first level (i. e., the children of r, here $\sqrt{\cdot}$ and -) are stored from left to right, then their children and so on. In the phenotypic representation, we have sketched the traversal order and numbered the levels. These level numbers are annotated to the gene but are neither part of the real phenotype nor the genotype. Furthermore, the division of the gene into head and tail is shown. In the head, the mathematical expressions as well as the variables may occur, while variables are the sole construction element of the tail.

In GEP, multiple genes form one genotype, thus encoding multiple expression trees. These trees may then be combined to one phenotype by predefined statements. It is easy to see that binary or integer strings can be used as genome, because the number of allowed symbols is known in advance.

This fixed mapping is also a disadvantage of Gene Expression Programming in comparison with the methods introduced later which have variable input grammars. On the other hand, there is the advantage that all genotypes can be translated to valid expression trees without requiring any corrections. Another benefit is that it seems to circumvent – at least partially – the problem of low causality from which the string-to-tree-GPM based approaches in often suffer. By modularizing the genotypes, potentially harmful influences of the reproduction operations are confined to single genes while others may stay intact. (See Section 4.8.1 on page 204 for more details.)

General Information

Areas Of Application

Some example areas of application of Gene Expression Programming are:

Application	References
Symbolic Regression and Function Synthesis	[659, 660, 1308]
Data Mining and Data Analysis	[1389, 1390, 2319, 2334, 2320, 1361]
Electrical Engineering and Circuit Design	[337]
Machine Learning	[661, 1278]
Geometry and Physics	[2018, 1127, 364]

Online Resources

Some general, online available ressources on Gene Expression Programming are:

http://www.gene-expression-programming.com/ [accessed 2007-08-19]	
Last update:	up-to-date
Description:	Gene Expression Programming Website. Includes publications, tutorials, and software.

4.4.4 Edge Encoding

Up until now, we only have considered how string genotypes can be transformed to more complex structures like trees. Obviously, genotype-phenotype mappings are not limited to this, but can work on tree genotypes as well. In [1321], Luke and Spector present their *edge encoding approach* where the genotypes are trees (or forests) of expressions from a graph-definition language. During the GPM, these trees are interpreted and construct the phenotypes, arbitrary directed graphs. Edge encoding is closely related to Gruau's cellular encoding [863], which works on nodes instead of edges.

Each functions and terminals in edge encoding work on tuples (a, b) containing two node identifiers. Such a tuple represents a directed edge from node a to node b. The functions edit these tuples, add nodes or edges and thus, successively build the graph. Unlike normal Genetic Programming applications like symbolic regression, for instance, the nodes of trees in edge encoding are "executed" from top to bottom (pre-order) and pass control down to their children (from left to right). After an edge has been processed by a terminal node, it becomes permanent part of the graph constructed. In order to allow the construction of arbitrary graphs, an additional control structure, a stack of node identifiers, is used. Each node in the GP tree may copy this stack, modify this copy, and pass it to all of its children.

In their paper [1321], Luke and Spector give multiple possible function and terminal sets for edge encoding. We provide a set that is sufficient to build arbitrary graphs in Table 4.1. Generally, each node receives an input edge tuple $E = (a, b)$ and a stack s which it can process. The two commands `labelE` and `labelN` in the table are no real functions but just here to demonstrate how nodes and edges can be enriched with labels and other sorts of information.

Operator	Children	Description
`double`	2	1. create a new edge $F = (a, b)$ 2. pass $E = (a, b)$ and the stack s to the first child 3. pass $F = (a, b)$ and the stack s to the second child
`bud`	2	1. create node c 2. create an edge $F = (b, c)$ 3. pass $E = (a, b)$ and the stack s to the first child 4. pass $F = (b, c)$ and the stack s to the second child
`split`	2	1. create a node c 2. change edge $E = (a, b)$ to $E = (a, c)$ 3. change edge $F = (c, b)$ 4. pass $E = (a, c)$ and the stack s to the first child 5. pass $F = (c, b)$ and the stack s to the second child
`loop`	2	1. create a new edge $F = (b, b)$ 2. pass $E = (a, b)$ and the stack s to the first child 3. pass $F = (b, b)$ and the stack s to the second child
`cut`	0	1. eliminate edge $E = (a, b)$
`nop`	0	1. make edge $E = (a, b)$ permanent
`push`	1	1. create a new node c 2. make a copy s' of the stack s 3. push c onto this copy s' 4. pass $E = (a, b)$ and the new stack s' to the child
`attach`	3	1. make a copy s' of the stack s 2. if s' is empty, create a new node and push it onto s' 3. pop the node c from the top of the stack s' 4. create two new edges $F = (a, c)$ and $G = (b, c)$ 5. pass $E = (a, b)$ and the new stack s' to the first child 6. pass $F = (a, c)$ and the new stack s' to the second child 7. pass $G = (b, c)$ and the new stack s' to the third child
`labelN`	1	1. label node b from edge $E = (a, b)$ with something 2. pass $E = (a, b)$ and the stack s to the child
`labelE`	1	1. label the edge $E = (a, b)$ with something 2. pass $E = (a, b)$ and the stack s to the child

Table 4.1: One possible operator set of edge encoding.

In Figure 4.15, an example genotype for edge encoding is given. The nodes of this genotype are annotated with the parameters which are passed to them by their parent. The root receives an initial node tuple and an empty stack. Notice that we have replaced the node

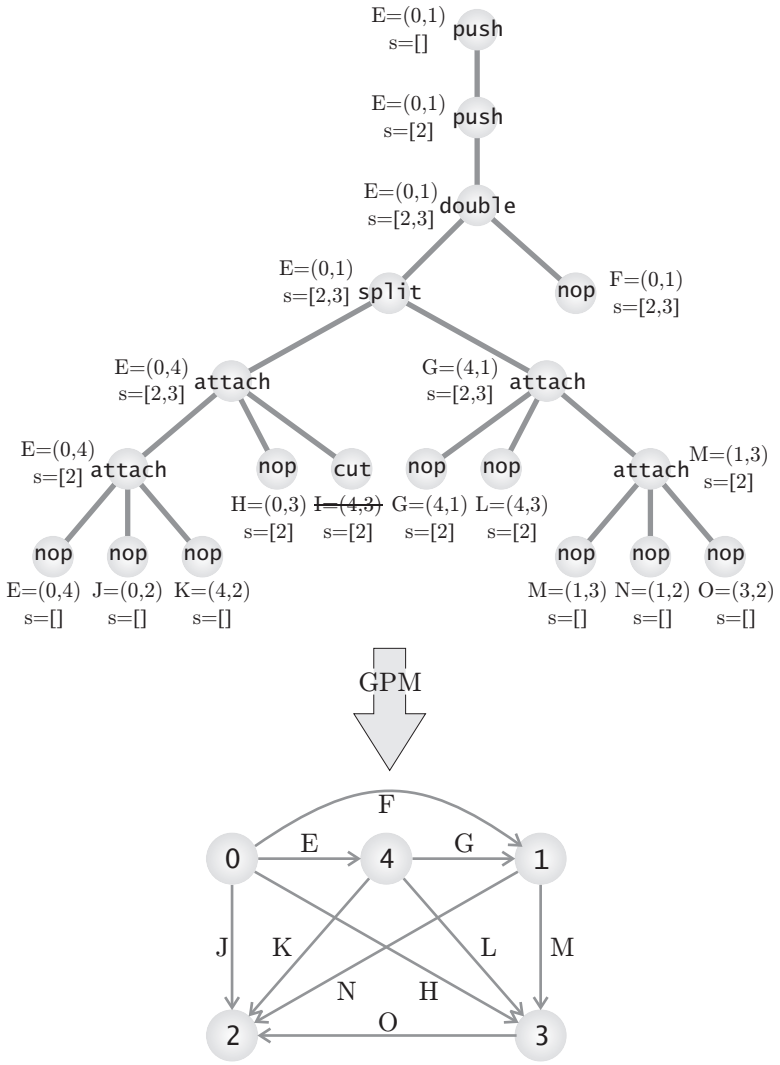


Figure 4.15: An example for edge encoding.

names a, b, c from Table 4.1 with running numbers and that new edges receive automatically a new name. At the bottom of the graphic, you find the result of the interpretation of the genotype by the GPM, a beautiful graph.

Edge encoding can easily be extended with automatically defined functions (as also shown in [1321]) and gave the inspiration for Sinclair's node pair encoding method for evolving network topologies [1887] (discussed in ?? on page ??). Vaguely related to such a graph generating approach are some of the methods for deriving electronic circuits by Lohn et al. [1306] and Koza et al. [1205] which you can find listed in the "Applications" tables in the general information sections.

4.5 Grammars in Genetic Programming

4.5.1 Introduction

We have learned that the most common genotypic and phenotypic representations in Genetic Programming are trees and also have discussed the reproduction operations available for tree-

based genomes. In this discussion, we left one out important point: in many applications, reproduction cannot occur freely. Normally, there are certain restrictions to the structure and shape of the trees that must not be violated. Take our pet-example symbolic regression[20] for instance. If we have a node representing a division operation, it will take two arguments: the dividend and the divisor. One argument is not enough and a third argument is useless, as one can easily see in Figure 4.16.

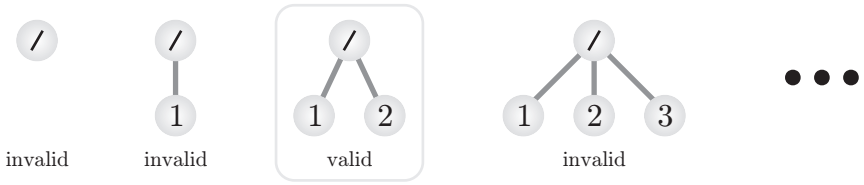


Figure 4.16: Example for valid and invalid trees in symbolic regression.

There are four general methods how to avoid invalid configurations under these limitations:

1. Compensate illegal configurations during the evaluation of the objective functions. This would mean, for example, that a division with no arguments could return 1, a division with only the single argument a could return a, and that superfluous arguments (like c in Figure 4.16) would simply be ignored.
2. A subsequent repair algorithm could correct errors in the tree structure that have been introduced during reproduction.
3. Using additional checking and refined node selection algorithms, we can ensure that only valid trees are created during the reproduction cycle.
4. With special genotype-phenotype mappings, we can prevent the creation of invalid trees from the start.

In this section, we will introduce some general methods of enforcing valid configurations in the phenotypes, mostly regarding the fourth approach. A very natural way to express structural and semantic restrictions of a search space are formal grammars which are elaborated on in Section 30.3 on page 561. Genetic Programming approaches that limit their phenotypes (the trees) to sentences of a formal language are subsumed under the topic of Grammar-guided Genetic Programming (GGGP, G3P) [1382].

4.5.2 Trivial Approach

Standard Genetic Programming as introduced by Koza [1196] already inherently utilizes simple mechanisms to ensure the correctness of the tree structures. These mechanisms are rather trivial, though, and should not be counted to the family of GGGP approaches, but are mentioned here for the sake of completeness.

In Standard Genetic Programming, all expressions have exactly the same type. Applied to symbolic regression, this means that, for instance, all constructs will be real-valued or return real values. If logical functions like multiplexers are grown, all entities will be Boolean-valued, and so on. For each possible tree node type, we just need to specify the exact amount of children. This approach corresponds to a context-free grammar[21] with a single non-terminal symbol which is expanded by multiple rules. Listing 4.4 illustrates such a trivial grammar $G = (N, \Sigma, P, S)$ in Backus-Naur Form (BNF)[22]. Here, the non-terminal symbol

[20] See Section 23.1 on page 397.
[21] see Section 30.3.2 on page 563 for details
[22] The Backus-Naur form is discussed in Section 30.3.4 on page 564.

```
1 <Z> ::= (<Z> + <Z>)
2 <Z> ::= (<Z> - <Z>)
3 <Z> ::= (<Z> * <Z>)
4 <Z> ::= (<Z> / <Z>)
5 <Z> ::= (sin <Z>)
6 <Z> ::= X
```

Listing 4.4: A trivial symbolic regression grammar.

is `Z` ($N = \{\texttt{Z}\}$), the terminal symbols are $\Sigma = \{\texttt{(}, \texttt{)}, \texttt{+}, \texttt{-}, \texttt{*}, \texttt{/}, \texttt{sin}, \texttt{X}\}$, and six different productions are defined. The start symbol is $S = \texttt{Z}$.

Standard Genetic Programming does not utilize such grammars directly. Rather, they are hard-coded in the reproduction operators or are represented in fixed internal data structures.

Here we should mention that illegal configurations can also rise at runtime from semantics. In symbolic regression, a division operation is invalid if the divisor is zero, for instance. The same goes for logarithms, or a tangent of $\left(n + \frac{1}{2}\right)\pi \ \forall n \in \mathbb{Z}$. All four approaches for enforcing a proper tree structure previously introduced cannot prevent such errors from the start. Therefore, the function set (the possible inner nodes of the trees) need to ensure the property of *closure* as defined by Koza [1196].

Definition 4.1 (Closure). If a function set N has the property *closure*, it ensures that all possible values are accepted as parameter by any function.

Closure is especially important in approaches like symbolic regression, and can easily be achieved by redefining the mathematical functions for special cases, like setting $\frac{a}{0} = a \ \forall a \in \mathbb{R}$, for instance. It does, however, not consider the tree structure itself – the number of arguments still needs to be sufficient.

4.5.3 Strongly Typed Genetic Programming

The strongly typed Genetic Programming (STGP) approach developed by Montana [1446, 1447, 1448] is still very close to Standard Genetic Programming. With strongly typed Genetic Programming, it becomes possible to use typed data structures and expressions in Genetic Programming. Hence, the issue of well-typedness arises, as illustrated in Figure 4.17.

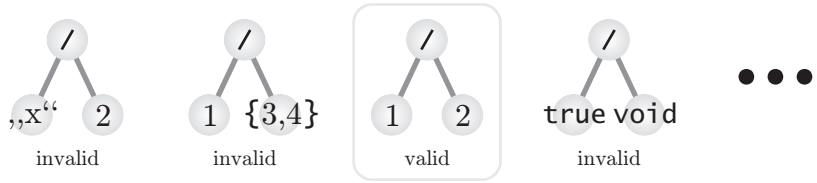


Figure 4.17: Example for valid and invalid trees in typed Genetic Programming.

As already mentioned in Section 4.5.2 on the previous page, in Standard Genetic Programming such errors are circumvented by only using representations that are type-safe per definition. In symbolic regression, for instance, only functions and variables which are real-typed are allowed, and in the evolution of logic functions only Boolean-valued expressions will be admitted. Thus, inconsistencies like in Figure 4.17 are impossible.

In STGP, a tree genome is used which permits different data types that are not assignment-compatible. One should not mistake STGP for a fully grammar-guided approach, since it uses rules still based on an implicit, hard-coded internal grammar which are built

in the bootstrap phase of the GP system. However, it represents clearly a method to shape the individuals according to some validity constraints.

These constraints are realized by modified reproduction operations that use *types possibilities tables* which denote which types for expressions are allowed in which level of a tree (individual). The creation and mutation operators now return valid individuals per default. Recombination still selects the node to be replaced in the first parent randomly, but the sub-tree in the second parent which should replace this node is selected in a way that ensures that the types match. If this is not possible recombination either returns the parents or an empty set.

STGP also introduces interesting new concepts like generic functions and data types, very much like in Ada or C [1448] and hierarchical type systems, comparable to object-oriented programming in their inheritance structure [910]. This way, STGP increases the reusability and modularity in GP which is needed for solving more complex problems [67, 1195].

4.5.4 Early Research in GGGP

Research steps into grammatically driven program evolution can be traced to the early 1990s where Antonisse [73] developed his Grammar-based Genetic Algorithm. As genome, he used character strings representing sentences in a formal language defined by a context-free grammar. Whenever crossover was to be performed, these strings were parsed into the derivation trees[23] of that grammar. Then, recombination was applied in the same way as in tree-based systems. This parsing was the drawback of the approach, leading to two major problems: First, it slows down the whole evolution since it is an expensive operation. Secondly, if the grammar is ambiguous, there may be more than one derivation tree for the same sentence [1382]. Antonisse's early example was succeeded by other researchers like Stefanski [1958], Roston [1763], and Mizoguchi et al. [1439].

In the mid-1990s [1382, 1785], more scientists began to concentrate on this topic. The LOGENPRO system developed by Wong and Leung [2250, 2247, 2248, 2249, 2251, 2252, 2253] used PROLOG Definite Clause Grammars to derive first-order logic programs. A GP system proposed by Whigham [2201, 2202, 2203, 2205, 2204] applied context-free grammars in order to generate populations of derivation trees. This method additionally had the advantage that it allowed the user to bias the evolution into the direction of certain parts of the grammar [2205]. Geyer-Schulz [795] derived a similar approach, differing mainly in the initialization procedure [241, 1382], for learning rules for expert systems. The Genetic Programming Kernel (GPK) by Hörner [960] used tree-genomes where each genotype was a deviation tree generated from a BNF definition.

4.5.5 Gads 1

The Genetic Algorithm for Deriving Software 1 (Gads 1) by Paterson and Livesey [1620, 1621] is one of the basic research projects that paved the way for other, more sophisticated approaches like Grammatical Evolution. Like the Binary Genetic Programming system by Keller and Banzhaf [1119], it uses a clear distinction between the search space $\mathbb{G}$ and the problem space $\mathbb{X}$. The genotypes $g \in \mathbb{G}$ in Gads are fixed-length integer strings which are transformed to character string phenotypes $x \in \mathbb{X}$ (representing program syntax trees) by a genotype-phenotype mapping (see Section 3.8 on page 155). Because of this genome, Gads can use a conventional genetic algorithm engine[24] to evolve the solution candidates.

Gads receives a context-free grammar $G = (N, \Sigma, P, S)$ specified in Backus-Naur form as input. In Binary Genetic Programming, the genome encodes the sequence of terminal symbols of the grammar directly. Here, a genotype specifies the sequence of the productions to be applied to build a sentence of terminal symbols.

[23] An elaboration on derivation trees can be found in Section 30.3.3 on page 563.

[24] Gads 1 uses the genetic algorithm C++ class library GAGS (`http://geneura.ugr.es/GAGS/` [accessed 2007-07-09]) release 0.95.

```
 (0) <expr>   ::= <expr> <op> <expr>
 (1) <expr>   ::= (<expr> <op> <expr>)
 (2) <expr>   ::= <pre-op> (<expr>)
 (3) <expr>   ::= <var>

 (4) <op>     ::= +
 (5) <op>     ::= -
 (6) <op>     ::= /
 (7) <op>     ::= *

 (8) <pre-op> ::= log
 (9) <pre-op> ::= tan
(10) <pre-op> ::= sin
(11) <pre-op> ::= cos

(12) <var>    ::= X

(13) <func>   ::= double func(double x){
                      return <expr>;
                  }
```

Listing 4.5: A simple grammar for C functions that could be used in Gads.

Although Gads was primarily tested with LISP S-expressions, it can evolve sentences in arbitrary BNF grammars. For the sake of coherence with later sections, we use a grammar for simple mathematical functions in C as example. Here, the set of possible terminals is $\Sigma = \{\texttt{sin}, \texttt{cos}, \texttt{tan}, \texttt{log}, \texttt{+}, \texttt{-}, \texttt{*}, \texttt{/}, \texttt{X}, \texttt{()}, \dots\}$ and as non-terminal symbols we use $N = \{\texttt{expr}, \texttt{op}, \texttt{pre-op}, \texttt{func}\}$. The starting symbol is $S = \texttt{func}$ and the set of productions P is illustrated in Listing 4.5.

In the BNF grammar definitions for Gads, the "|" symbol commonly denoting alternatives is not used. Instead, multiple productions can be defined for the same non-terminal symbol.

Every gene in a Gads genotype contains the index of the production in G to be applied next. For now, let us investigate the genotype $g = (2, 0, 12, 5, 5, 13, 10)$ as example. If the predefined start symbol is `func`, we would start with the phenotype string x_1

```
double func(double x){
  return <expr>;
}
```

The first gene in g, 2, leads to the application of rule `(2)` to x_1 and we obtain x_2:

```
double func(double x){
  return <pre-op> (<expr>);
}
```

The next gene is 0, which means that we will use production `(0)`). There is a (non-terminal) `expr` symbol in x_2, so we get x_3 as follows:

```
double func(double x){
  return <pre-op> (<expr> <op> <expr>);
}
```

Now comes the next gene with allele 12[25]. We cannot apply rule `(12)` since no `var` symbol can be found in x_3 – we simple ignore this gene and set $x_3 = x_4$. The following gene with value 5 translates the symbol `op` to `-` and we obtain for x_5:

[25] An allele is a value of specific gene, see Definition 1.24 on page 43.

```
double func(double x){
  return <pre-op> (<expr> - <expr>);
}
```

The next two genes, 5 and 13, must again be ignored ($x_7 = x_6 = x_5$). Finally, the last gene with the allele 10 resolves the non-terminal `pre-op` and we get for x_8:

```
double func(double x){
  return sin (<expr> - <expr>);
}
```

For the remaining two `expr` non-terminal symbols no rule is defined in the genotype g. There are several ways for dealing with such incomplete resolutions. One would be to exclude the individual from evaluation/simulation and to give it the lowest possible objective values directly. Gads instead uses simple default expansion rules. In this example, we could translate all remaining `expr`s to `var`s and these subsequently to `X`. This way we obtain the resulting function below.

```
double func(double x){
  return sin (X - X);
}
```

One of the problems in Gads is the unacceptable large number of introns[26] [1619] caused by the encoding scheme. Many genes will not contribute to the structure of the phenotype since they encode productions that cannot be executed (like allele 12 in the example genotype g) because there are no matching non-terminal symbols. This is especially the case in "real-world" applications where the set of non-terminal symbols N becomes larger.

With the Gads system, Paterson paved the way for many of the advanced techniques described in the following sections.

4.5.6 Grammatical Evolution

Like Gads, Grammatical Evolution[27] (GE), developed by Ryan et al. [1785], creates expressions in a given language by iteratively applying the rules of a grammar specified in the Backus-Naur form [1785, 1565, 1784].

In order to discuss how Grammatical Evolution works, we re-use the example of C-style mathematical functions [1785] from Section 4.5.5. Listing 4.6 specifies the according rules using a format which is more suitable for grammatical evolution.

There are five rules in the set of productions P, labeled from `A` to `E`. Some of the rules have different options (separated by `|`). In each rule, options are numbered started with `0`. When the symbol `<exp>` for example is expanded, for example, there are four possible results (`0-3`). The shape of the sentences produced by the grammar depends on these choices.

Like in Gads, the genotypes in GE are numerical strings. These strings encode the indices of the options instead of the productions themselves. In Gads, each option was treated as a single production because of the absence of the "`|`" operator. The idea of Grammatical Evolution is that it is already determined which rules must be used by the non-terminal symbol to be expanded and we only need to decide which option of this rule is to be applied. Therefore, the number of introns is dramatically reduced compared to Gads.

The variable-length string genotypes of Grammatical Evolution can again be evolved using genetic algorithms [1785, 1783] (like in Gads) or with other techniques, like Particle Swarm Optimization [1578, 1568] or Differential Evolution [1567]. As illustrated in Figure 4.18, a Grammatical Evolution system consists of three components: the problem definition (including the means of evaluating a solution candidate), the grammar that defines the possible shapes of the individuals, and the search algorithm that creates the individuals [1782].

[26] Introns are genes or sequences of genes (in the genotype) that do not contribute to the phenotype or its behavior, see Definition 3.2 on page 146 and Section 4.10.3.

[27] `http://en.wikipedia.org/wiki/Grammatical_evolution` [accessed 2007-07-05]

```
(A) <expr>    ::= <expr> <op> <expr>     (0)
                | (<expr> <op> <expr>)   (1)
                | <pre-op> (<expr>)      (2)
                | <var>                  (3)

(B) <op>      ::= +                      (0)
                | -                      (1)
                | /                      (2)
                | *                      (3)

(C) <pre-op>  ::= log                    (0)
                | tan                    (1)
                | sin                    (2)
                | cos                    (3)

(D) <var>     ::= X                      (0)

(E) <func>    ::= double func(double x){
                      return <expr>;
                  }                      (0)
```

Listing 4.6: A simple grammar for C functions that could be used by GE.

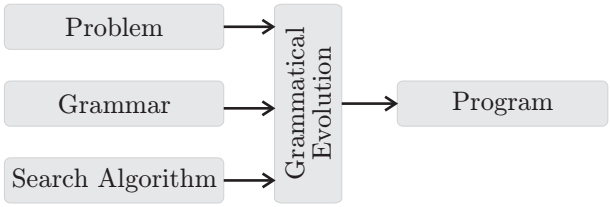


Figure 4.18: The structure of a Grammatical Evolution system [1782].

An Example Individual

We get back to our mathematical C function example grammar in Listing 4.6. As already said, a genotype $g \in \mathbb{G}$ is a variable-length string of numbers that denote the choices to be taken whenever a non-terminal symbol from N is to be expanded and more than one option is available (as in the productions `(A)`, `(B)`, and `(C)`). The start symbol, $S =$ `func` does not need to be encoded since it is predefined. Rules with only one option do not consume information from the genotype. The processing of non-terminal symbols uses a depth-first order [1785], so resolving a non-terminal symbol ultimately to terminal symbols has precedence before applying an expansion to a sibling.

Let us assume we have settled for bytes as genes in the genome. As we may have less than 256 options, we apply modulo arithmetic to get the index of the option. This way, the sequence $g = (193, 47, 51, 6, 251, 88, 63)$ is a valid genotype. According to our grammar, the first symbol to expand is $S =$ `func` (rule `(E)`) where only one option is available. Therefore, all phenotypes will start out like

```
double func(double x){
 return <expr>;
}
```

The next production we have to check is `(A)`, since it expands `expr`. This productions has four options, so taking the first number from the genotype g, we get $193 \bmod 4 = 1$ which means that we use option `(1)` and obtain

```
double func(double x){
 return (<expr> <op> <expr>);
}
```

As `expr` appears again, we have to evaluate rule `(A)` once more. The next number, 47, gives us $47 \bmod 4 = 3$ so option `(3)` is used.

```
double func(double x){
 return (<var> <op> <expr>);
}
```

`var` is expanded by rule `(D)` where only one result is possible:

```
double func(double x){
 return (X <op> <expr>);
}
```

Subsequently, `op` will be evaluated to `*` since $51 \bmod 4 = 3$ (rule `(B)(3)`) and `expr` becomes `pre-op(<expr>)` because $6 \bmod 4 = 2$ (production `(A)(2)`). Rule `(C)(3)` then turns `pre-op` into `cos` since $251 \bmod 4 = 3$. `expr` is expanded to `<expr> <op> <expr>` by `(A)(0)` because $88 \bmod 4 = 0$. The last gene in our genotype is 63, and thus rule `(A)(3)` ($63 \bmod 4 = 3$) transforms `expr` to `<var>` which then becomes `X`.

```
double func(double x){
 return (X * cos(X <op> <expr>));
}
```

By now, the numbers available in g are exhausted and we still have non-terminal symbols left in the program. As already outlined earlier, there are multiple possible approaches how to proceed in such a situation:

1. Mark g as invalid and give it a reasonably bad fitness.
2. Expand the remaining non-terminals using default rules (i. e., we could say the default value for `expr` is `X` and `op` becomes `+`),
3. or wrap around and restart taking numbers from the beginning of g.

The latter method is applied in Grammatical Evolution. It has the disadvantage that it can possible result in an endless loop in the genotype-phenotype translation, so there should be a reasonable maximum for the iteration steps after which we fall back to default rules.

In the example, we will proceed by expanding `op` according to `(B)(1)` since $193 \bmod 4 = 1$ and obtain `-` (minus). The next gene gives us $47 \bmod 4 = 3$ so the last `expr` will become a `<var>` and finally our phenotype is:

```
double func(double x){
 return (X * cos(X - X));
}
```

Note that if the last gene 63 was missing in g, the "restart method" which we have just described would produce an infinite loop, because the first non-terminal to be evaluated whenever we restart taking numbers from the front of the genome then will always be `expr`. In this example, we are lucky and this is not the case since after wrapping at the genotype end, a `pre-op` is to be resolved. The gene 193 thus is an index into rule `A` at its first usage and an index into production `C` in the second application.

Initialization

Grammatical Evolution uses an approach for initialization similar to ramped half-and-half[28], but on basis of derivation trees[29]. Therefore, the numbers of the choices made during a

[28] An initialization method of standard, tree-based Genetic Programming that creates a good mixture of various tree shapes [1196], see Section 4.3.1 on page 163 for more details.

[29] see Section 30.3.3 on page 563

random grammatical rule expansion beginning at the start symbol are recorded. Then, a genotype is built by reversing the modulo operation, i. e., finding a number that produces the same number as recorded when modulo-divided for each gene. The number of clones is subsequently reduced and, optionally, the single-point individuals are deleted.

General Information

Areas Of Application

Some example areas of application of Grammatical Evolution are:

Application	References
Mathematical Problems (vs. Standard Genetic Programming: [1196])	[1783, 1785]
Automated Programming	[1784, 1569, 1566]
Robotics (vs. Standard Genetic Programming: [1317, 1204])	[1576, 1575]
Economics and Finance (vs. Standard Genetic Programming: [1513, 1674])	[1577, 266, 264, 265]

There even exists an approach called "Grammatical Evolution by Grammatical Evolution" ($(GE)^2$, [1571]) where the grammar defining the structure of the solution candidates itself is co-evolved with the individuals.

Conferences, Workshops, etc.

Some conferences, workshops and such and such on Grammatical Evolution are:

GEWS: Grammatical Evolution Workshop
http://www.grammatical-evolution.com/gews.html [accessed 2007-09-10]
History: 2004: Seattle, WA, USA, see [1574]
2003: Chicago, IL, USA, see [1573]
2002: New York, NY, USA, see [1572]

Online Resources

Some general, online available ressources on Grammatical Evolution are:

http://www.grammatical-evolution.com/ [accessed 2007-07-05]
Last update: up-to-date
Description: Grammatical Evolution Website. Includes publications, links, and software.

Books

Some books about (or including significant information about) Grammatical Evolution are:

O'Neill and Ryan [1570]: *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*

4.5.7 Gads 2

In Gads 2, Paterson [1619] uses the experiences from Gads 1 and the methods of the Grammatical Evolution approach to tackle context-sensitive grammars with Genetic Programming. While context-free grammars are sufficient to describe the syntax of a programming language, they are not powerful enough to determine if a given source code is valid. Take for example the C snippet:

```
1 char i;
2 i = 0.5;
3 y = 1;
```

This is obviously not a well-typed program although syntactically correct. Context-sensitive grammars[30] allow productions like $\alpha A\beta \to \alpha\gamma\beta$ where $A \in N$ is a non-terminal symbol, and $\alpha, \beta, \gamma \in V^*$ are concatenations of arbitrary many terminal and non-terminal symbols (with the exception that $\gamma \neq \varepsilon$, i. e., it must not be the empty string). Hence, it is possible to specify that a value assignment to a variable must be of the same type as the variable and that the variable must have previously been declared with a context-sensitive grammar. Paterson argues that the application of existing approaches like two-level grammars and standard attribute grammars[31] in Genetic Programming is infeasible [1619] and introduces an approach based on *reflective attribute grammars*.

Definition 4.2 (Reflective Attribute Grammar). A reflective attribute grammar (*rag*[32]) [1619] is a special form of attribute grammars. When expanding a non-terminal symbol with a *rag* production, the grammar itself is treated as an (inherited) attribute. During the expansion, it can be modified and is finally passed on to the next production step involving the newly created nodes.

The transformation of a genotype $g \in \mathbb{G}$ into a phenotype using a reflective attribute grammar r resembles Grammatical Evolution to some degree. Here we discuss it with the example of the recursive expansion of the symbol s:

1. Write the symbol s to the output.
2. If $s \in \Sigma$, i. e., s is a terminal symbol, nothing else is to do – return.
3. Use the next gene in the genotype g to choose one of the alternative productions that have s on their left hand side. If g is exhausted, choose the default rule.
4. Create the list of the child symbols $s_1 \ldots s_n$ according to the right-hand side of the production.
5. For $i = 1$ to n do
   a) Resolve the symbol i, passing in s_i, r, and g.
   b) If needed, modify the grammar r according to the semantics of s and s_i.

Item 5 is the main difference between Gads 2 and Grammatical Evolution. What happens here depends on the semantics in the *rag*. For example, if a non-terminal symbol that declares a variable x is encountered, a new terminal symbol κ is added to the alphabet Σ that corresponds to the name of x. Additionally, the rule which expands the non-terminal symbol that stands for variables of the same type now is extended by a new option that returns κ. Thus, the new variable becomes available in the subsequent code.

Another difference to Grammatical Evolution is the way the genes are used to select an option in item 3. GE simply uses the modulo operation to make its choice. Assume we have

[30] See Section 30.3.2 on page 563 where we discuss the Chomsky Hierarchy of grammars.
[31] See Section 30.3.6 on page 565 for a discussion of attribute grammars.
[32] Notice that the shortcut of this definition *rag* slightly collides with the one of Recursive Adaptive Grammars (*RAG*) introduced by Shutt [1874] and discussed in Section 30.3.8 on page 568, although their letter cases differ. To the knowledge of the author, *rags* are exclusively used in Gads 2.

genotypes where each gene is a single byte and encounter a production with four options while the next gene has the value 45. In Grammatical Evolution, this means to select the second option since $45 \bmod 4 = 1$ and we number the alternatives beginning with zero. Gads 2, on the other hand, will divide the range of possible alleles into four disjoint intervals of (approximately) equal size $[0, 63]$, $[64, 127]$, $[128, 191]$, $[192, 255]$ where 45 falls clearly into the first one. Thus, Gads 2 will expand the first rule.

The advantage of Gads 2 is that it allows to grow valid sentences according to context-sensitive grammars. It becomes not only possible to generate syntactically correct but also well-typed source code for most conventional programming languages. Its major drawback is that it has not been realized fully. The additional semantics of the production expansion rule 5b have not been specified in the grammar or in an additional language as input for the Genetic Programming system but are only exemplarily realized in a hard-coded manner for the programming language S-Algol [1461]. The experimental results in [1619], although successful, do not provide substantial benefits compared to the simpler Grammatical Evolution approach.

Gads 2 shows properties that we also experienced in the past: Even if constructs like loops, procedure calls, or indexed access to memory are available, the chance that they are actually used in the way in which we would like them to be used is slim. Genetic Programming of real algorithms in a high-level programming language-like syntax exhibits a high affinity to employ rather simple instructions while neglecting more powerful constructs. Good fitness values are often reached with overfitting only.

Like Grammatical Evolution, the Gads 2 idea can be realized with arbitrary genetic algorithm engines. Paterson [1619] uses the Java-based evolutionary computation system ECJ by Luke et al. [1327] as genetic algorithm engine in his experiments.

4.5.8 Christiansen Grammar Evolution

Christiansen Grammar, which you can find described in Section 30.3.9 on page 569, have many similarities to the reflective attribute grammars used in Gads 2. They are both Extended Attribute Grammars[33] and the first attribute of both grammars is an inherited instance of themselves. Christiansen Grammars are formalized and backed by comprehensive research since being developed back in 1985 by Christiansen [402].

Building on their previous work de la Cruz Echeandía et al. [520] place the idea of Gads 2 on the solid foundation of Christiansen Grammars with their Christiansen Grammar Evolution approach (CGE) [521]. They tested their system for finding logic function identities with constraints on the elementary functions to be used. Instead of elaborating on this experiment, let us stick with the example of mathematical functions in C for the sake of simplicity.

In Listing 4.7 we define the productions P of a Christiansen Grammar that extends the examples from before by the ability of creating and using local variables. Three new rules `(F)`, `(G)`, and `(H)` are added, and the existing ones have been extended with attributes.

The non-terminal symbol `expr` now receives the inherited attribute `g` which is the (Christiansen) grammar to be used for its expansion. The ↓ (arrow down) indicates inherited attribute values that are passed down from the parent symbol, whereas ↑`a` (arrow up) identifies an attribute value `a` synthesized during the expansion of a symbol and passed back to the parent symbol.

The start symbol S is still `func`, but the corresponding production `(E)` has been complemented by a reference to the new non-terminal symbol `stmt` (line 19). The symbol `stmt` has two attributes: an inherited (input) grammar `g0` and a synthesized (output) grammar `g2`. We need to keep that in mind when discussing the options possible for its resolution. A `stmt` symbol can either be expanded to two new `stmt`s in option `(0)`, a variable declaration

[33] See Section 30.3.7 on page 567 for more information on such grammars.

```
1  (A) <expr ↓g>      ::= <expr↓g> <op↓g> <expr↓g>        (0)
2                        | (<expr ↓g> <op ↓g> <expr ↓g>)  (1)
3                        | <pre-op ↓g> (<expr ↓g>)        (2)
4                        | <var ↓g>                       (3)
5
6  (B) <op ↓g>        ::= "+"                            (0)
7                        | "-"                            (1)
8                        | "/"                            (2)
9                        | "*"                            (3)
10
11 (C) <pre-op ↓g>    ::= "log"                          (0)
12                       | "tan"                          (1)
13                       | "sin"                          (2)
14                       | "cos"                          (3)
15
16 (D) <var ↓g>        ::= "X"                           (0)
17
18 (E) <func ↓g1>      ::= "double␣func(double␣x){"
19                             <stmt ↓g1 ↑g2>
20                             "return␣" <expr ↓g2> ";"
21                             }                         (0)
22
23 (F) <stmt ↓g0 ↑g2> ::= <stmt ↓g0 ↑g1><stmt ↓g1  ↑g2>  (0)
24                          | <new-var ↓g0 ↑g2>          (1)
25                          | <assign ↓g0 ↑g2>           (2)
26
27 (G) <new-var ↓g ↑g+new-rule>  ::=
28             "double␣" <alpha-list ↓g ↑w> "=0;"        (0)
29                where <new-rule> is <var ↓g> ::= w
30
31 (H) <assign ↓g ↑g> ::= <var ↓g> "=" <expr ↓g> ";"     (0)
```

Listing 4.7: A Christiansen grammar for C functions that that use variables.

represented by the non-terminal symbol `new-var` as option `(1)`, or to a variable assignment (symbol `assign`) in option `(2)`. Most interesting here is option `(1)`, the variable declaration.

The production for `new-var`, labeled `(G)`, receives the grammar `g` as input. The synthesized attribute it generates as output is `g` extended by a new rule `new-rule`. The name of the new variable is a string over the Latin alphabet. In order to create this string, we make use of the non-terminal symbol `alpha-list` defined in Listing 30.12 on page 569. `alpha-list` inherits a grammar as first attribute, generates a character string `w`, and also synthesizes it as output. Production `(G)` uses this value `w` in order to build its output grammar. It creates a new rule (see line 29) which extends the production `(D)` by a new option. `var` can now be resolved to either `X` or to one of the new variables in subsequent expansions of `expr` because the synthesized grammar is passed up to `stmt` and from there to all subsequent statements (see rule `(F)` option `(0)`) and even by the returned expression in line 20. It should be mentioned that this example grammar does not prevent name collisions of the identifiers, since `X`, for instance, is also a valid expansion of `new-var`.

With this grammar, a Christiansen Grammar Evolution system would proceed exactly as done in Section 4.5.6 on page 181.

4.5.9 Tree-Adjoining Grammar-guided Genetic Programming

A different approach to Grammar-guided Genetic Programming has been developed by Nguyen [1525] with his Tree-Adjoining Grammar-guided Genetic Programming (TAG3P)

system [1526, 1529, 1527, 1528, 1530]. Instead of using grammars in the Backus-Naur Form or one of its extensions as done in the aforementioned methods, it bases on tree-adjoining grammars (TAGs) which are introduced in Section 30.3.10 on page 569.

An Example TAG grammar

A tree-adjoining grammar can be defined as quintuple $G = (N, \Sigma, A, I, S)$ where N are the non-terminal, Σ contains the terminal symbols, and S is the start symbol. TAGs support two basic operations: adjunction and substitution. For these operations, blueprint trees are provided in the set of auxiliary and initial trees respectively (A and I). Substitution is quite similar to expansion in BNF, the root of an initial tree replaces a leaf with the same label in another tree. A tree β to be used for adjunction has at least one leaf node ν (usually marked with an asterisk *) with the same label as its root. It is injected into another tree by replacing a node with (again) that label whose children are then attached to ν.

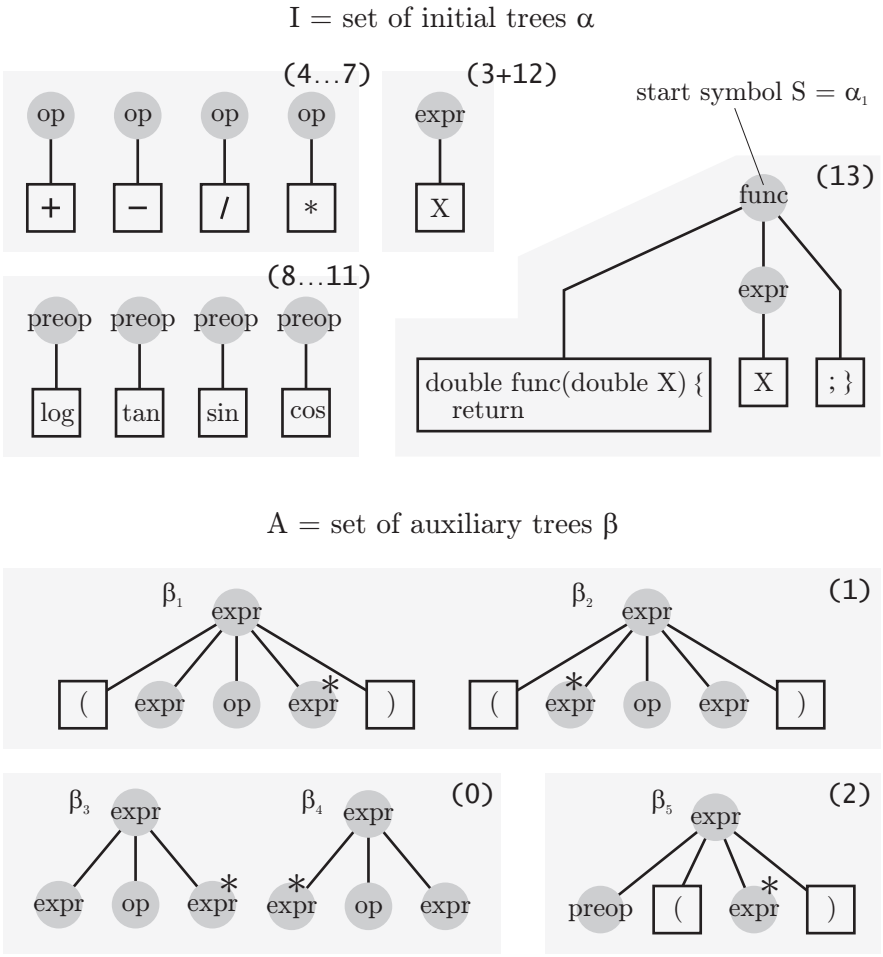


Figure 4.19: An TAG realization of the C-grammar of Listing 4.6.

Let us take a look back on the tree-adjoining representation of our earlier example grammar G in Listing 4.6 on page 182 for mathematical functions in C. Figure 4.19 illustrates one possible realization of G as TAG. The productions are divided into the set of initial trees I, which are used in substitution operations, and the auxiliary trees A needed by the

adjunction operator. Again, the start symbol is `func` – this time however it identifies a tree in I. We additionally have annotated the trees with the index of the corresponding rule in Listing 4.6. It is possible that we need to build multiple TAG trees for one BNF rule, as done with rule `1` which is reflected in the two auxiliary tress β_1 and β_2. The rules `3` and `12` on the other hand have been united into one initial tree for the purpose of simplicity (It could have been done in the BNF in the same way).

Like the other grammar-guided methods, the TAG3P approach uses a genotype-phenotype mapping. The phenotypes are, of course, trees that comply with the input tree-adjoining grammar. The genotypes being evolved are derivation trees that work on this grammar too. Derivation trees illustrate the way the productions of a grammar are applied in order to derive a certain sentence, as discussed in Section 30.3.3 on page 563.

Derivation Trees

For tree-adjoining grammars, there exist different types of derivation trees [1525]. In the method of Weir [2174], they are characterized as object trees where the root is labeled with an S-type initial tree (i. e., the start symbol) and all other trees are labeled with the names of auxiliary trees. Each connection from a parent p to a child node c is labeled with the index of the node in p being the center of the operation. Indices are determined by numbering the non-terminal nodes according to a preorder traversal[34]. The number of adjunctions performed with each node is limited to one. Substitution operations are not possible with Weir's approach. Joshi and Schabes [1074] introduce an extension mitigating this problem. In their notation (not illustrated here) a solid connection between two nodes in the derivation tree stands for adjunction, whereas a broken line denotes a substitution.

In TAG3P, Nguyen [1525] uses a restricted form of such TAG derivation trees where adjunction is not permitted to (initial) trees used for substitution. This essentially means that all adjunctions are performed before any substitutions. With this definition, substitutions become basically in-node operations. We simply attach the nodes substituted into a tree as list of lexemes (here terminal symbols) to the according node of a derivation tree.

Example Mapping: Derivations Tree → Tree

Figure 4.20 outlines some example mappings from derivation trees on the left side to sentences of the target languages (displayed as trees) on the right side. In Figure 4.19, we have annotated some of the elementary trees with α or β and numbers which we will use here. The derivation tree α_1, for example, represents the initial production for the starting symbol. In addition, we have attached the preorder index to each node of the trees α_1, β_3, and β_5. In the next tree we show how the terminal symbols `X` and `+` can be substituted into β_3. In the corresponding derivation tree, they are simply attached as a list of lexemes. A similar substitution can be performed with β_5, where `sin` is attached as terminal symbol.

In the fourth example, the second derivation tree is adjoined to the first one. Since it replaces the node with the preorder index 1, the connection from β_3 to α_1 is labeled with 1. Finally, in the fifth example, the third derivation tree is adjoined. We use the rule for `preop`s to replace the node number 3 (according to preorder) in the second derivation in its adjoined state.

As you can see, all initial trees as well as all trees derived from them are always valid sentences of the grammar. This means that we can remove any of the derivation steps and still get valid phenotypes. Thus, we can evaluate the share of the fitness clubbed by every single modification by evaluating the resulting phenotypes with and without it.

[34] http://en.wikipedia.org/wiki/Tree_traversal [accessed 2007-07-18]

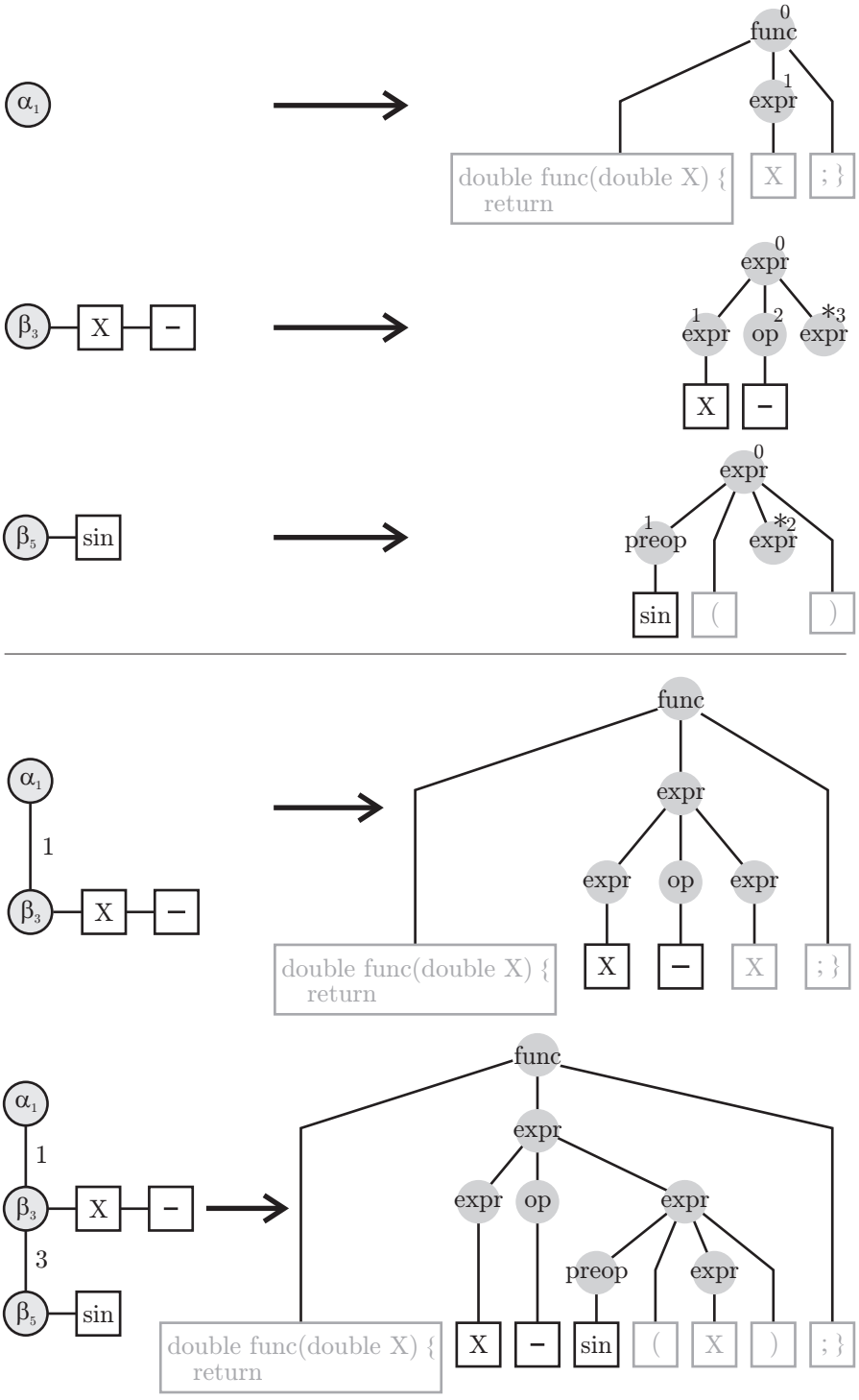


Figure 4.20: One example genotype-phenotype mapping in TAG3P.

Summary

Tree-Adjoining Grammar-guided Genetic Programming is a different approach to Grammar-guided Genetic Programming which has some advantages compared with the other methods. One of them is the increased domain of locality. All nodes of a derivation tree stay accessible for the reproduction operations. This becomes interesting when modifying nodes "without side effects to other regions of the resulting trees". If we, for example, toggle one bit in a Grammatical Evolution-based genotype, chances are that the meaning of all subsequent genes change and the tree resulting from the genotype-phenotype mapping will be totally different from its parent. In TAG3P, this is not the case. All operations can, at most, influence the node they are applied to and its children. Here, the principle of strong causality holds since small changes in the genotype lead to small changes in the phenotype. On the other hand, some of these positive effects may also be reached more easily with the wrapping and lifting operations for Genetic Programming introduced in this book in Section 4.3.7 on page 166 and Section 4.3.8. The reproduction operations of TAG3P become a little bit more complicated. When performing crossover, for instance, we can only exchange *compatible* nodes. We cannot adjoin the tree α_1 in Figure 4.20 with itself, for example.

General Information

Areas Of Application

Some example areas of application of Tree-Adjoining Grammar-guided Genetic Programming are:

Application	References
Symbolic Regression and Function Synthesis	[1529, 1527, 1528]
Mathematical Problems	[1524, 1531]

Online Resources

Some general, online available ressources on Tree-Adjoining Grammar-guided Genetic Programming are:

`http://sc.snu.ac.kr/SCLAB/Research/publications.html` [accessed 2007-09-10]	
Last update:	up-to-date
Description:	Publications of the Structural Complexity Laboratory of the Seoul National University, includes Nguyen's papers about TAG3P

4.6 Linear Genetic Programming

4.6.1 Introduction

In the beginning of this chapter, we have learned that the major goal of Genetic Programming is to find programs that solve a given set of problems. We have seen that tree genomes are suitable to encode such programs and how the genetic operators can be applied to them.

Nevertheless, we have also seen that trees are not the only way for representing programs. Matter of fact, a computer processes them as sequences of instructions instead. These sequences may contain branches in form of jumps to other places in the code. Every possible flowchart describing the behavior of a program can be translated into such a sequence. It is therefore only natural that the first approach to automated program generation developed by Friedberg [750] at the end of the 1950s used a fixed-length instruction sequence genome

[750, 751]. The area of Genetic Programming focused on such instruction string genomes is called *linear Genetic Programming* (LGP).

Linear Genetic Programming can be distinguished from approaches like Grammatical Evolution (see Section 4.5.6 on page 181) by the fact that strings there are just genotypic, intermediate representations that encode the program trees. In LGP, they are the center of the whole evolution and contain the program code directly. Some of the most important early contributions to this field come from [1667]:

1. Banzhaf [135], who used a genotype-phenotype mapping with repair mechanisms to translate a bit string into a sequence of simple arithmetic instructions in 1993,
2. Perkis [1636] (1994), whose stack based GP evaluated arithmetic expressions in Reverse Polish Notation (RPN),
3. Openshaw and Turton [1582] (1994) who also used Perkis's approach but already represented mathematical equations as fixed-length bit string back in the 1980s [1581], and
4. Crepeau [464], who developed a machine code GP system around an emulator for the Z80 processor.

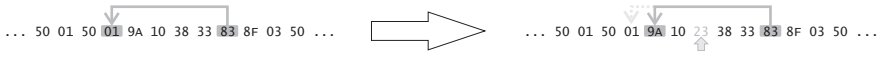
Besides the methods discussed in this section, other interesting approaches to linear Genetic Programming are the LGP variants developed by Eklund [627] and Leung et al. [1273, 380] on specialized hardware, the commercial system by Foster [736], and the MicroGP (μGP) system for test program induction by Corno et al. [451, 1949].

4.6.2 Advantages and Disadvantages

The advantage of linear Genetic Programming lies in the straightforward evaluation of the evolved algorithms. Its structure furthermore eases limiting the runtime in the program evaluation and even simulating parallelism. The drawback is that simply reusing the genetic operators for variable-length string genomes (discussed in Section 3.5 on page 149), which randomly insert, delete, or toggle bits, is not really feasible. In LGP forms that allow arbitrary jumps and call instructions to shape the control flow, this becomes even more eminent because of a high degree of *epistasis* (see Section 1.4.6 and Section 4.8).

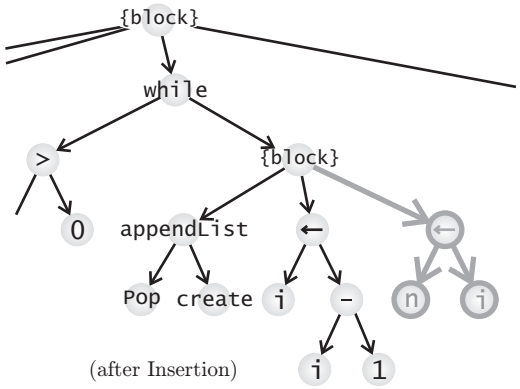
We can visualize, for example, that the alternatives and loops which we know from high-level programming languages are mapped to conditional and unconditional jump instructions in machine code. These jumps target to either absolute or relative addresses inside the program. Let us consider the insertion of a single, new command into the instruction string, maybe as result of a mutation or recombination operation. If we do not perform any further corrections after this insertion, it is well possible that the resulting shift of the absolute addresses of the subsequent instructions in the program invalidates the control flow and renders the whole program useless. This issue is illustrated in Fig. 4.21.a. Nordin et al. [1546, 1546] point out that standard crossover is highly disruptive. Even though the subtree crossover in tree-genomes is shown to be not very efficient either [62], in comparison, tree-based genomes are less vulnerable in this aspect. The loop in Fig. 4.21.b, for instance, stays intact although it is now one useless instruction richer. In LGP, precautions have to be taken in order to mitigate these problems, linear Genetic Programming becomes more competitive to Standard Genetic Programming also in terms of robustness of the recombination operations.

One approach to do so is to create intelligent mutation and crossover operators which preserve the control flow of the program when inserting or deleting instructions. Such operations could, for instance, analyze the program structure and automatically correct jump targets, for instance. Operations which are restricted to have only minimal effect on the control flow from the start can also easily be introduced. In Section 4.6.6, we shortly outline some of the work of Brameier and Banzhaf, who define some interesting approaches to this issue. Section 4.6.7 discusses the homologous crossover operation which represents another method for decreasing the destructive effects of reproduction in LGP.



(before insertion) (after insertion: loop begin shifted)

Fig. 4.21.a: Inserting into an instruction string.



(after Insertion)

Fig. 4.21.b: Inserting in a tree representation.

Figure 4.21: The impact of insertion operations in Genetic Programming

4.6.3 The Compiling Genetic Programming System

Its roots go back to Nordin [1541], who was dissatisfied with the performance of GP systems written in an interpreted language which, in turn, interpret the programs evolved using a tree-shaped genome. In 1994, he published his work on a new *Compiling Genetic Programming System* (CGPS) written in the C programming language[35] [1126] directly manipulating individuals represented as machine code.

Each solution candidate consisted of a prologue for shoveling the input from the stack into registers, a set of instructions for information processing, and an epilogue for terminating the function [1542]. The prologue and epilogue were never modified by the genetic operations. As instructions for the middle part, the Genetic Programming system had arithmetical operations and bit-shift operators at its disposal in [1541], but no control flow manipulation primitives like jumps or procedure calls. These were added in [1543] along with ADFs, making this LGP approach Turing-complete.

Nordin [1541] used the classification of Swedish words as task in the first experiments with this new system. He found that it had approximately the same capability for growing classifiers as artificial neural networks but performed much faster. Another interesting application of his system was the compression of images and audio data [1545].

4.6.4 Automatic Induction of Machine Code by Genetic Programming

CGPS originally evolved code for the Sun Sparc processors, which is a member of the RISC[36] processor class. This had the advantage that all instructions are have the same size. In the *Automatic Induction of Machine Code with GP* system (AIM-GP, AIMGP), the successor of CGPS, the support for multiple other architectures was added by Nordin, Banzhaf, and Francone [1549, 1550], including Java bytecode[37] and CISC[38] CPUs with variable instruction widths such as Intel 80x86 processors. A new interesting application for

[35] http://en.wikipedia.org/wiki/C_(programming_language) [accessed 2008-09-16]
[36] http://de.wikipedia.org/wiki/Reduced_Instruction_Set_Computing [accessed 2008-09-16]
[37] http://en.wikipedia.org/wiki/Bytecode [accessed 2008-09-16]
[38] http://de.wikipedia.org/wiki/Complex_Instruction_Set_Computing [accessed 2008-09-16]

```
1  void ind(double[8] v) {
2  ...
3  v[0] = v[5] + 73;
4  v[7] = v[0] - 59;           (I)
5  if(v[1] > 0)
6  if(v[5] > 23)
7      v[4] = v[2] * v[1];
8  v[2] = v[5] + v[4];         (I)
9  v[6] = v[0] * 25;           (I)
10 v[6] = v[4] - 4;
11 v[1] = sin(v[6]);
12 if(v[0] > v[1])             (I)
13     v[3] = v[5] * v[5];     (I)
14 v[7] = v[6] * 2;
15 v[5] = v[7] + 115;          (I)
16 if(v[1] <= v[6])
17     v[1] = sin(v[7]);
18 }
```

Listing 4.8: A genotype of an individual in Brameier and Banzhaf's LGP system.

linear Genetic Programming tackled with AIMGP is the evolution of robot behavior such as obstacle avoiding and wall following [1548].

4.6.5 Java Bytecode Evolution

Besides AIMGP, there exist numerous other approaches to the evolution of linear Java bytecode functions. The Java Bytecode Genetic Programming system (JBGP, also Java Method Evolver, JME) by Lukschandl et al. [1328, 1329, 1330, 1331] is written in Java. A genotype in JBGP contains the maximum allowed stack depth together with a linear list of instruction descriptors. Each instruction descriptor holds information such as the corresponding bytecode and the branch offset. The genotypes are transformed with the genotype-phenotype mapping into methods of a Java class which then can be loaded into the JVM, executed, and evaluated. [903, 902].

The JAPHET system of Klahold et al. [1147], the user provides an initial Java class at startup. Classes are divided into a static and a dynamic part. The static parts contain things like version information are not affected by the reproduction operations. The dynamic parts, containing the methods, are modified by the genetic operations which add new byte code [903, 902].

Harvey et al. [903, 902] introduce *byte code GP* (bcGP), where the whole population of each generation is represented by one class file. Like in AIMGP, each individual is a linear sequence of Java bytecode and is surrounded by a prologue and epilogue. Furthermore, by adding buffer space, each individual has the same size and, thus, the whole population can be kept inside a byte array of a fixed size, too.

4.6.6 Brameier and Banzhaf: LGP with Implicit Intron removal

In the Genetic Programming system developed by Brameier and Banzhaf [272] based on former experience with AIMGP, an individual is represented as a linear sequence of simple C instructions as outlined in the example Listing 4.8 (a slightly modified version of the example from [272]). Due to reproduction operations like as mutation and crossover, such genotypes may contain introns, i.e., instructions not influencing the result (see Definition 3.2 and Section 4.10.3). Given that the output of the program defined in Listing 4.8 will store its outputs in `v[0]` and `v[1]`, all the lines marked with `(I)` do not contribute to the overall

functional fitness. Brameier and Banzhaf [272] introduce an algorithm which removes these introns during the genotype-phenotype mapping, before the fitness evaluation. This linear Genetic Programming method was successfully tested with several classification tasks [272, 271, 273], function approximation and Boolean function synthesis [274].

In his doctoral dissertation, Brameier [269] elaborates that the control flow of linear Genetic Programming more equals a graph than a tree because of jump and call instructions. In the earlier work of Brameier and Banzhaf [272] mentioned just a few lines ago, introns were only excluded by the genotype-phenotype mapping but preserved in the *genotypes* because they were expected to make the programs robust against variations. In [269], Brameier concludes that such *implicit* introns representing unreachable or ineffective code have no real protective effect but reduce the efficiency of the reproduction operations and, thus, should be avoided or at least minimized by them. Instead, the concept of *explicitly defined introns* (EDIs) proposed by Nordin et al. [1547] is utilized in form of something like `nop` instructions in order to decrease the destructive effect of crossover. Brameier finds that introducing EDIs decreases the proportion of introns arising from unreachable or ineffective code and lead to better results. In comparison with standard tree-based GP, his linear Genetic Programming approach performed better during experiments with classification, regression, and Boolean function evolution benchmarks.

4.6.7 Homologous Crossover: Binary Reproduction

According to Banzhaf et al. [140], natural crossover is very restricted and usually exchanges only genes that express the same functionality and are located at the same positions (loci) on the chromosomes.

Definition 4.3 (Homology). In genetics, *homology*[39] of protein-coding DNA sequences means that they code for the same protein which may indicate common functionality. Homologous chromosomes[40] are either chromosomes in a biological cell that pair during meiosis or non-identical chromosomes which code for the same functional feature by containing similar genes in different allelic states.

In other words, homologous genetic material is very similar and in nature, only such material is exchanged in sexual reproduction. In linear Genetic Programming with default crossover, it is hard for the evolution to establish a clear structure or a map between locus and functionality. Francone et al. [740, 1549] introduce a *sticky crossover* operator which resembles homology by allowing the exchange of instructions between two genotypes (programs) only if they reside at the same loci. It first chooses a sequence of code in the first genotype and then swaps it with the sequence at exactly the same position in the second parent.

4.6.8 Page-based LGP

A similar approach is the Page-based linear Genetic Programming of Heywood and Zincir-Heywood [923], where programs are described as sequences of pages, each including the same number of instructions. Here, crossover exchanges only a single page between the parents and, as a result, becomes less destructive. This approach should be distinguished from the fixed block size approach of Nordin et al. [1549] for CISC architectures which was developed to accommodate variable instruction lengths in AIMGP.

4.7 Graph-based Approaches

In this section, we will discuss some Genetic Programming approaches that are based on graphs rather than on trees or linear sequences of instructions.

[39] `http://en.wikipedia.org/wiki/Homology_(biology)` [accessed 2008-06-17]
[40] `http://en.wikipedia.org/wiki/Homologous_chromosome` [accessed 2008-06-17]

4.7.1 Parallel Algorithm Discovery and Orchestration

Parallel Algorithm Discovery and Orchestration (PADO) is a Genetic Programming method introduced by Teller and Veloso [2011, 2015] in the mid-1990s. In their CEC paper [2014] and their 1996 book chapter [2016], they describe the graph-structure of their approach as sketched in Figure 4.22. A PADO program is a directed graph of up to n nodes, where

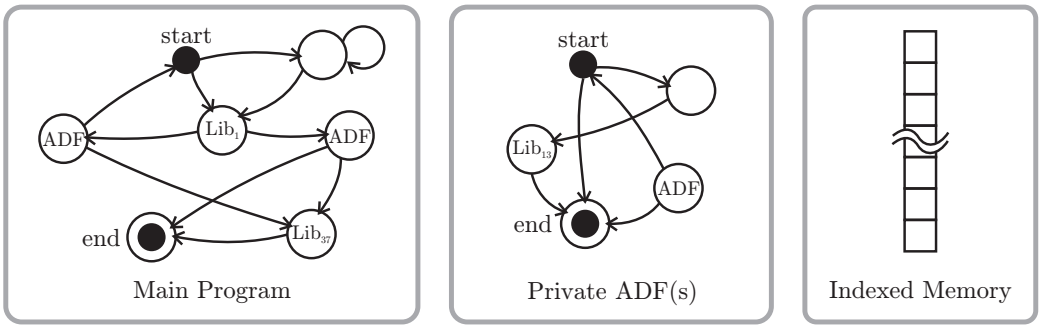


Figure 4.22: The general structure of a Parallel Algorithm Discovery and Orchestration program.

each node may have as many as n outgoing arcs which define the possible control flows. A node consists of two parts: and *action* and a *branching decision*. The programs used indexed memory and an implicitly accessed stack. The actions pop their inputs from the stack and place their outputs onto it. After a node's action has been executed, the branching decision function is used to determine over which of the outgoing arcs the control will be transferred. It can access the stack, the memory, and the action type of the previous node in order to make that decision.

A program in the PADO-syntax has a start node which will initially receive the control token and an end node which terminates the program after its attached action has been performed. Furthermore, the actions may call functions from a library and automatically defined functions (ADFs). These ADFs basically have same structure as the main program and can also invoke themselves recursively.

As actions, PADO provides algebraic primitives like `+`, `-`, `*`, `/`, `NOT`, `MAX`, and `MIN`; the memory instructions `READ` and `WRITE`; branching primitives like `IF-THEN-ELSE` and `PIFTE` (alternative with a randomized condition); as well as constants and some domain-specific instructions. Furthermore, actions may invoke more complex library functions or ADFs. An action takes its arguments from the stack and also pushes its results back onto it. The action `6`, for instance, pushes 6 on the stack whereas the `WRITE` action pops two values, v_1 and v_2, from it and pushes the value of the memory cell indexed by v_2 before storing v_1 at this location.

In PADO, so-called SMART operators are used for mutation and recombination which co-evolve with the main population as described in [2013, 2016].

4.7.2 Parallel Distributed Genetic Programming

Parallel Distributed Genetic Programming (PDGP) is a method for growing programs in the form of graphs that has been developed by Poli [1654, 1655, 1658, 1659] in the mid 1990s. In PDGP, a graph is represented as a fixed-size, n-dimensional grid. The nodes of the grid are labeled with operations, functions, or references to variables. Except for the latter case, they are connected to their inputs with directed links. Both, the labels as well as the connections in the grid are subject to evolution.

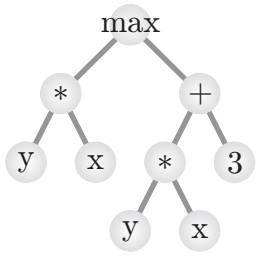


Fig. 4.23.a: tree structure

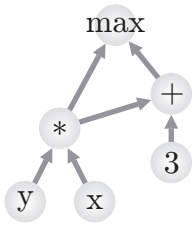


Fig. 4.23.b: graph structure

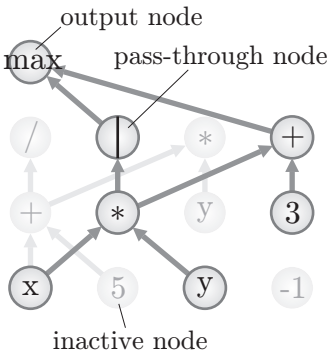


Fig. 4.23.c: PDGP structure

Figure 4.23: The term $\max\{x * y, x * y + 3\}$

In order to illustrate this structure, we use the term $\max\{x * y, x * y + 3\}$ as example. We already have elaborately discussed how we can express mathematical terms as trees. Fig. 4.23.a illustrates such a function tree. Using a directed graph, as outlined in Fig. 4.23.b, we can retrieve a more compact representation of the same term by reusing the expression $x * y$. Evolving such graphs is the goal of PDGP. Therefore, a grid structure needs to be defined first. In Fig. 4.23.c, we have settled for a two dimensional 4*3 grid. Additionally, we add a row at the top containing one cell for each output of the program. We can easily fill the graph from Fig. 4.23.b into this grid. This leaves some nodes unoccupied. If we assume that Fig. 4.23.c represents a solution grown by this Genetic Programming approach, these nodes would be labeled with some unused expressions and would somehow be connected without any influence on the result of the program. Such an arbitrary configuration of *inactive* nodes (or introns and links is sketched in light gray in Fig. 4.23.c. The nodes which have influence on the result of the program, i. e., those which are connected to an output node directly or indirectly, are called *active* nodes.

We may impose restrictions on the connectivity of PDGP graphs. For instance, we can define that each row must only be connected to its predecessor in order to build layered feed-forward networks. We can transform any given parallel distributed program (i. e., any given acyclic graph) into such a layered network if we additionally provide the identity function so pass-through nodes can evolve as shown in Fig. 4.23.c. Furthermore, we could also attach weights to the links between the nodes and make them also subject to evolution. This way, we can also grow artificial neural networks [1657]. However, we can as well do without any form of restrictions for the connectivity and may allow backward connections in the programs, depending on the application.

An interesting part of PDGP is how the programs are executed. Principally, it allows for a great proportion of parallelism. Coming back to the example outlined Fig. 4.23.c, the values of the leaf nodes could be computed in parallel, as well those of the pass-through and the addition node.

Genetic Operations

For this new program representation, novel genetic operations are needed.

Creation

Similar to the *grow* and *full* methods for creating trees in Standard Genetic Programming introduced in Section 4.3.1 on page 162, it is possible to obtain balanced or unbalanced

graphs/trees in PDGP, depending whether we allow variables and constants to occur anywhere in the program or only at a given, predetermined depth.

Crossover: Binary Reproduction

SAAN Crossover The basic recombination operation in PDGP is *Sub-graph Active-Active Node (SAAN) crossover*. The idea of SAAN crossover is that active sub-graphs represent functional units which should be combined in different ways in order to explore new, useful constellations. It proceeds as follows:

1. Select a random active node in each parent, the crossover points.
2. Extract the sub-graph that contains all the (active) nodes that influence the result of the node marking the crossover point in the first parent.
3. Insert this sub-graph at the crossover point in the second parent. If its x-coordinate is incompatible and some nodes of the sub-graph would be outside the grid, wrap it so that these nodes are placed on the other side of the offspring.

Of course, we have to ensure that the depths of the crossover points are compatible and no nodes of the sub-graph would "hang" below the grid in the offspring. This can be achieved by first selecting the crossover point in the first parent and then choosing a compatible crossover point in the second parent.

SSAAN Crossover The *Sub-Sub-Graph Active-Active Node (SSAAN) Crossover* method works essentially the same way, with one exception: it disregards crossover point depth compatibility. It may now happen that we want to insert a sub-graph into an offspring at a point where it does not fit because it is too long. Here we make use of the simple fact that the lowest row in a PDGP graph always is filled with variables and constants only – functions cannot occur there because otherwise, no arguments could be connected to them. Hence, we can cut the overhanging nodes of the sub-graph and connect the now unsatisfied arguments at second-to-last row with the nodes in the last row of the second parent. Of course, we have to pay special attention where to cut the sub-graph: terminal nodes that would be copied to the last row of the offspring can remain in it, functions cannot.

SSIAN Sub-Sub-Graph Inactive-Active Node (SSIAN) Crossover works exactly like SSAAN crossover except that the crossover point in the first parent is chosen amongst both, active and inactive nodes.

Mutation: Unary Reproduction

We can extend the mutation operation from Standard Genetic Programming easily to PDGP by creating new, random graphs and insert them at random points into the offspring. In the context of PDGP, this is called *global mutation* and can be achieved by creating a completely new graph and performing crossover with an existing one.

Furthermore, *link mutation* is introduced as an operation that performs simple local changes on the connection topology of the graphs.

ADLs

Similar to Standard Genetic Programming, we can also introduce automatically defined functions[41] in PDGP by extending the function set with a new symbol which then executes an (also evolved) subprogram when being evaluated. *Automatically Defined Links*, ADLs, work similarly, except that a link is annotated with the subprogram-invoking symbol [1653, 1656].

[41] More information on ADFs can be found in Section 4.3.9 on page 167.

4.7.3 Genetic Network Programming

Genetic Network Programming (GNP) is a Genetic Programming technique introduced by Katagiri et al. [1095] at the 2001 CEC conference in Seoul [1095, 1096, 1097, 931]. In GNP, programs are represented as directed graphs called networks which consist of three types of nodes: the start node, judgment nodes and processing nodes. A processing nodes executes an action from a predefined set of actions P and can have exactly on outgoing connection to a successor node. Judgment nodes may have multiple outgoing connections and have one expression from the set of possible judgment decisions J attached to them with which they make this decision. As in the example illustrated in Figure 4.24, each node in the network is

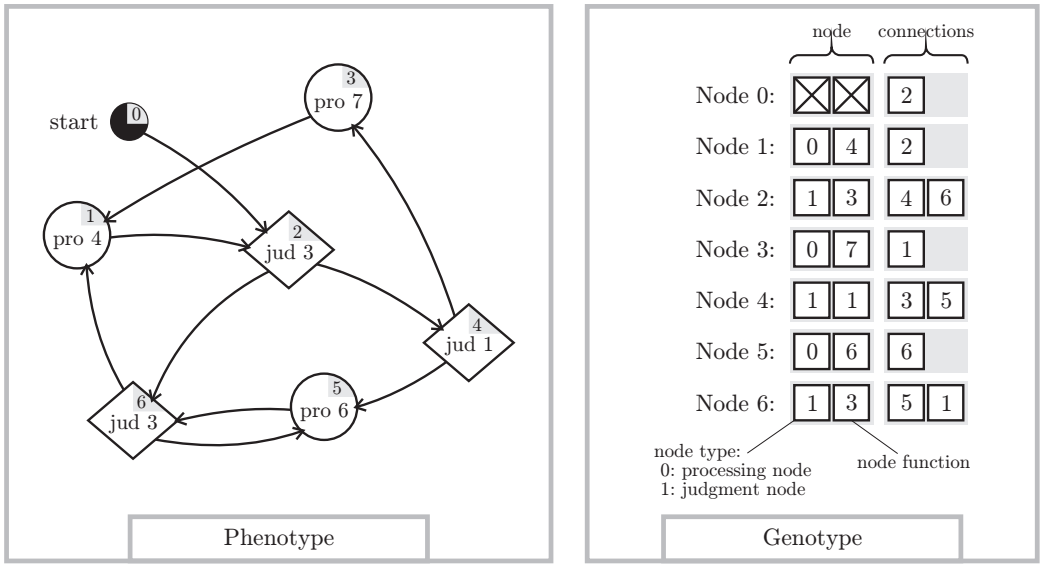


Figure 4.24: An example program in Genetic Network Programming syntax.

represented by two genes, a node gene and a connection gene. The node gene consists of two values, the node type (which is 0 for processing nodes and 1 for judgment nodes) and the function index. For processing nodes, the function index can take on the values from 0 to $|P|-1$ and for judgment nodes, it is in $0..|J|-1$. These values identify the action or decision function to be executed whenever the node receives the control token. In the connection gene, the indices of the other nodes the node is connected to are stored. For processing nodes, this list has exactly one entry, for judgment nodes there always are at least two outgoing connections (in Figure 4.24, there are exactly two). Notice that programs can be interpreted in this representation directly without needing an explicit genotype-phenotype mapping.

Crossover is performed by randomly exchanging notes (and their attached connections) between the parent networks and mutation randomly changes the connections. Murata and Nakamura [1492] extended their approach in order to evolve programs for multi-agent systems where the behavior of agents depends on the group they are assigned groups to. In this Automatically Defined Groups (ADG) model, an individual is defined as a set of GNP programs [1491, 1490]. Genetic Network Programming has also been combined with reinforcement learning by Mabu et al. [1337].

4.7.4 Cartesian Genetic Programming

Cartesian Genetic Programming (CGP) was developed by Miller and Thomson [1421] in order to achieve a higher degree of effectiveness in learning Boolean functions [1418, 1422].

In his 1999 paper, Miller [1418] explains the idea of Cartesian Genetic Programming with the example of a program with $o = 2$ outputs that computes both, the difference and the sum, of the volumes of two boxes $V_1 = X_1X_2X_3$ and $V_2 = Y_1Y_2Y_3$. As illustrated in Figure 4.25, the $i = 6$ input variables $X_1 \dots X_3$ and $Y_1 \dots Y_3$, placed to the left, are numbered from 0

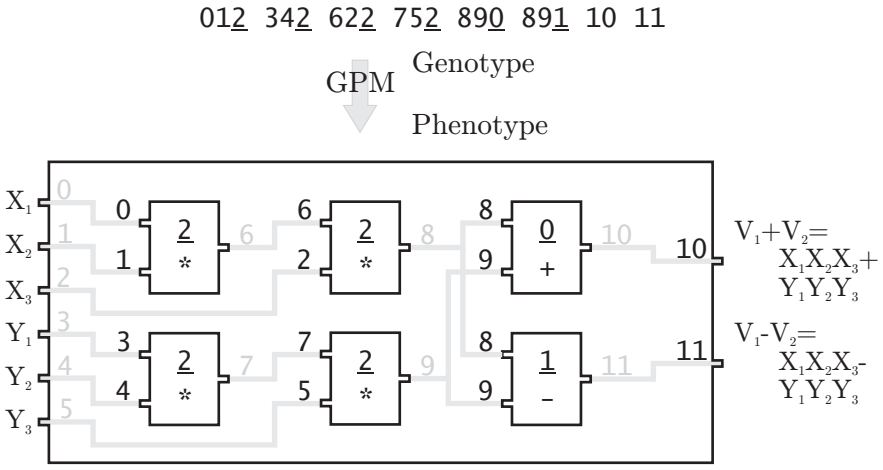


Figure 4.25: An example for the GPM in Cartesian Genetic Programming.

to 5. As function set, we use $\{+ = 0, - = 1, * = 2, / = 3, \vee = 4, \wedge = 5, \oplus = 6, \neg = 7\}$. Like in PDGP, we define a grid of cells before the evolution begins. In our example, this grid is $n = 3$ cells wide and $m = 2$ cells deep. Each of the cells can accommodate an arbitrary function and has a fixed number of inputs and outputs (in the example $i' = 2$ and $o' = 1$, respectively). The outputs of the cells, similarly to the inputs of the program, are numbered in ascending order beginning with i. The output of the cell in the top-left has number 6, the one of the cell below number 7, and so on. This numeration is annotated in gray in Figure 4.25.

Which functions the cells should carry out and how their inputs and outputs are connected will be decided by the optimization algorithm. Therefore, we could use, for instance, a genetic algorithm with or without crossover or a hill climbing approach. The genotypes of Cartesian Genetic Programming are fixed-length integer strings. They consist of $n*m$ genes, each encoding the configuration of one cell. Such a gene starts with i' numbers identifying the incoming data and one number (underlined in Figure 4.25) denoting the function it will carry out. Another gene at the end of the genotype identifies which of the available data are "wired" to the outputs of the program.

By using a fixed-length genotype, the maximum number of expressions in a Cartesian program is also predefined. It may, however, be shorter, since not all internal cells are necessarily connected with the output-producing cells. Furthermore, not all functions need to incorporate all i' inputs into their results. $\neg$, which is also part of the example function set, for instance, uses only the first of its $i' = 2$ input arguments and ignores the second one.

Levels-back, a parameter of CGP, is the number of columns to the left of a given cell whose outputs may be used as inputs of this cell. If levels-back is one, the cell with the output `8` in the example could only use `6` or `7` as inputs. A levels-back value of 2 allows it to be connected with `0-5`. Of course, the reproduction operations have to respect the levels-back value set.

CGP labeled itself a Genetic Programming technique from the beginning. However, most of the work contributed about it did not consider a recombination operation. Hence, one

could regarded it also as an evolutionary programming[42] method. Lately, researchers also begin to focus on efficient crossover techniques for CGP [414].

Neutrality in CGP

Cartesian Genetic Programming explicitly utilizes different forms of neutrality[43] in order to foster the evolutionary progress. Normally, neutrality can have positive as well as negative effects on the evolvability of a system. Yu and Miller [2297, 2296] outline different forms of neutrality in Cartesian Genetic Programming which also apply to other forms of GP or GAs:

1. Inactive genes define cells that are not connected to the outputs in any way and hence cannot influence the output of the program. Mutating these genes therefore has no effect on the fitness and represents *explicit neutrality*.
2. Active genes have direct influence on the results of the program. Neutral mutations here are such modifications that have no influence on the fitness. This *implicit neutrality* is the results of functional redundancy or introns.

Their experiments indicate that neutrality can increase the chance of success of Genetic Programming for needle-in-a-haystack fitness landscapes and in digital circuit evolution [2110].

Embedded Cartesian Genetic Programming

In 2005, Walker and Miller [2135] published their work on Embedded Cartesian Genetic Programming (ECGP), a new type of CGP with a module acquisition [66] method in form of automatic module creation [2135, 2136, 2137]. Therefore, three new operations are introduced:

1. **Compress** randomly selects two points in the genotype and creates a new module containing all the nodes between these points. The module then replaces these nodes with a cell that invokes it. The compress operator has the effect of shortening the genotype of the parent and of making the nodes in the module immune against the standard mutation operation but does not affect its fitness. Modules are more or less treated like functions so cell to which a module number has been assigned now uses that module as "cell function".
2. **Expand** randomly selects a module and replaces it with the nodes inside. Only the cell which initially replaced the module cells due to the Compress operation can be expanded in order to avoid bloat.
3. The new operator **Module Mutation** changes modules by adding or removing inputs and outputs and may also carry out the traditional one-point mutation on the cells of the module.

General Information

Areas Of Application

Some example areas of application of Cartesian Genetic Programming are:

Application	References
Electrical Engineering and Circuit Design	[1420, 2110, 1421, 1419, 1081, 2136]

[42] See Chapter 6 on page 231 for more details on evolutionary programming.
[43] See Section 1.4.5 on page 64 and Section 1.4.5 on page 67 for more information on neutrality and redundancy.

Symbolic Regression and Function Synthesis	[1418, 1422, 2297, 1422]
Robotics	[895, 2138]
Prime Number Prediction	[2139]

Online Resources

Some general, online available ressources on Cartesian Genetic Programming are:

`http://www.cartesiangp.co.uk/` [accessed 2007-11-02]	
Last update:	up-to-date
Description:	The homepage of Cartesian Genetic Programming
`http://www.emoware.org/evolutionary_art.asp` [accessed 2007-11-02]	
Last update:	2006
Description:	A website with art pieces evolved using Cartesian Genetic Programming.

4.8 Epistasis in Genetic Programming

In the previous sections, we have discussed many different Genetic Programming approaches like Standard Genetic Programming and the Grammar-guided Genetic Programming family. We also have elaborated on linear Genetic Programming techniques that encode an algorithm as a stream of instructions, very much like real programs are represented in the memory of a computer.

When we use such methods to evolve "real algorithms", we often find that the fitness landscape is very rugged. To a good part, this ruggedness is rooted in epistasis (see Section 1.4.6 on page 68). In the following section, we want to discuss the different epistatic effects which can be observed in Genetic Programming.

Subsequently, we will introduce some means to mitigate the effects of epistasis. One such approach would be to "learn the linkage" (see Section 1.4.6) between the single instructions. Module acquisition [66] can be regarded as one idea on how to do this. Generally, the linkage between the primitives in GP is far more complicated than in usual genetic algorithm-problems, which is why the author tends to believe that linkage learning will not achieve the same success in the GP than it did in the area of genetic algorithms. Therefore, we consider methods which consider the *representation* of the solution candidates rather than the nature of the search operations applied to the genotypes in order to mitigate or circumvent epistasis more promising. In Section 4.8.2 to Section 4.8.4, we will discuss three such methods.

4.8.1 Forms of Epistasis in Genetic Programming

Semantic Epistasis

In an algorithm, the behavior of each instruction depends on the operations that have been executed before. The result of one instruction will influence the behavior of those executed afterwards. If an instruction is changed, if the arithmetic operation `a = b + c` is swapped with `a = b - c`, for instance, its effects on subsequent instructions will change too [2011]. This obvious fact fully complies with the definition of epistasis and we will refer to it as *semantic* epistasis.

Positional Epistasis

Epistasis also occurs in form of *positional* interdependencies. In order to clarify the role of this facet in the context of Genetic Programming, we begin with some basic assumptions.

Let us consider a program P as some sort of function $P : I \mapsto O$ that connects the possible inputs I of a system to its possible outputs O. Two programs P_1 and P_2 can be considered as equivalent if $P_1(i) = P_2(i)\ \forall i \in I$.[44]

For the sake of simplicity, we further define a program as a sequence of n statements $P = (s_1, s_2, \ldots, s_n)$. For these statements, there are $n!$ possible permutations. We argue that the fraction $\theta(P) = \frac{m}{n!}$ of m permutations that lead to programs equivalent to P is one measure of robustness for a given program representation. More precisely, a low value of θ indicates a high degree of positional epistasis, which means that the loci (the positions) of many different genes in a genome have influence on their functionality [1502]. This reduces, for example, the efficiency of reproduction operations like recombination, since they often change the number and order of instructions in a program.

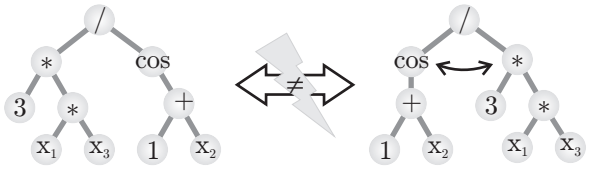


Fig. 4.26.a: in Standard Genetic Programming and symbolic regression

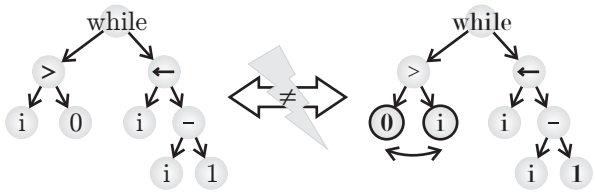


Fig. 4.26.b: in Standard Genetic Programming

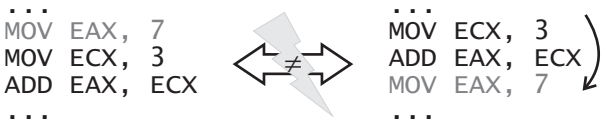


Fig. 4.26.c: in linear Genetic Programming

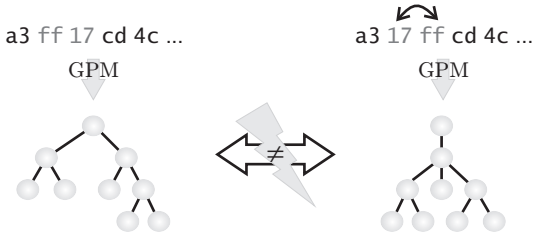


Fig. 4.26.d: in genotype-phenotype mappings, like in Grammatical Evolution-like approaches

Figure 4.26: Positional epistasis in Genetic Programming.

Many of the phenotypic and most genotypic representations in Genetic Programming mentioned so far seem to be rather fragile in terms of insertion and crossover points. One of

[44] In order to cover stateful programs, the input and output sets may also comprise sequences of data.

the causes is that their genomes have high positional epistasis (low θ-measures), as sketched in Figure 4.26.

Embryogenic Epistasis (Problems of String-to-Tree GPMs)

Many Grammar-guided Genetic Programming methods like Grammatical Evolution[45], Christiansen Grammar Evolution[46], and Gads[47] employ a genotype-phenotype mapping between an (integer) string genome and trees that represent sentences in a given grammar. According to Ryan et al. [1785], the idea of mapping string genotypes can very well be compared to one of the natural prototypes of artificial embryogeny[48]: the translation of the DNA into proteins. This process depends very much on the proteins already produced and which are now present around the cellular facilities. If a certain piece of DNA has created a protein X and is transcriped again, a molecule of protein type Y may result because of the presence of X.

Although this is a nice analogy, it also bears an important weakness. Search spaces which exhibit such effects usually suffer from weak causality[49] [1382] and only the lord knows why the DNA stuff works at all. Different from the aforementioned positional epistasis, this *embryogenic* episasis interacts with the genotype-phenotype mapping and modifies the phenotypic outcome. In Grammatical Evolution for example, a change in any gene in a genotype g will likely also change the meaning of all alleles following after its locus. This means that mutation and crossover will probably have very destructive impacts on the individuals [1525]. Hence, even the smallest change in the genotype can modify the whole structure and functionality of the phenotype. A valid solution can become infeasible after a single reproduction operation.

Figure 4.27 outlines how the change of a single bit in a genotype (in hexadecimal notation) may lead to a drastic modification in the tree structure when a string-to-tree mapping is applied. The resulting phenotype in the example has more or less nothing in common with its parent except maybe the type of its root node. Furthermore, the efficiency of the reproduction operations of the mentioned approaches will likely decrease with a growing set of non-terminal symbols and corresponding productions.

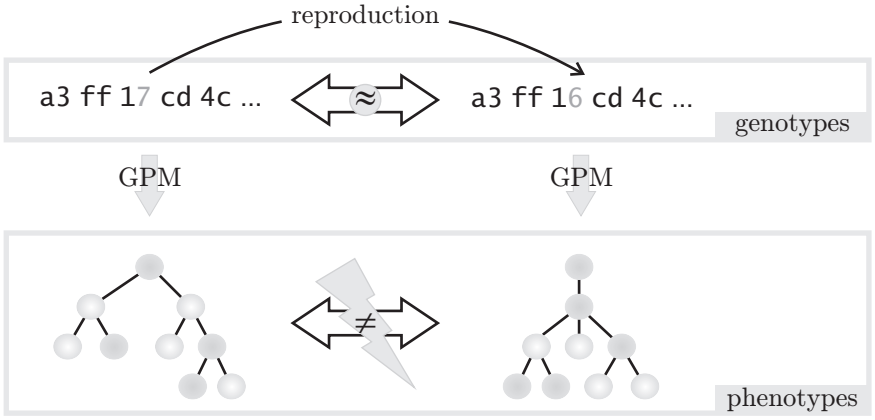


Figure 4.27: Epistasis in a Grammatical Evolution-like approach.

[45] See Section 4.5.6 on page 181, [1785]
[46] See Section 4.5.8 on page 186, [521]
[47] See Section 4.5.5 on page 179, [1620]
[48] Find out more about artificial embryogeny in Section 3.8 on page 155.
[49] The principles of causality and locality are discussed in Section 1.4.3 on page 61.

The points discussed in this section do by no means indicate that the involved Genetic Programming approaches are infeasible or deliver only inferior results. Most of them have provided human-competitive solutions or performed even better. We just point out some classes of problems that, if successfully solved, could even increase the utility of the GGGP methods even further.

4.8.2 Algorithmic Chemistry

Algorithmic Chemistries were first discussed by Fontana [720] on basis of the λ-calculus. The Algorithmic Chemistry approach by Lasarczyk and Banzhaf [1258]BL2005GPOAAC,LB2005GPOAAC,LB2005TSO represents one possible method to circumvent the positional epistasis discussed in Section 4.8.1. To put it bluntly, this form of Algorithmic Chemistries is basically a variant of linear Genetic Programming[50] where the execution order of the single instructions is defined by some random distribution instead of being fixed as in normal programs.

This can probably best be described by using a simple example. Therefore, let us define the set of basic constructs which will make up the programs first. In both, [138] and [1259], an assembler-like language is used where each instruction has three parameters: two input and one output register addresses. Registers are the basic data stores, the variables, of this language. Instructions with a behavior depending on a single input only simply ignore their second parameter. In [138], Banzhaf and Lasarczyk use a language comprising the eight instructions `+`, `-`, `/`, `*`, `^`, `and`, `or`, and `not`. Furthermore, they provided eleven read-only registers with evolved constants and thirty read-write registers, the so-called *connection registers*.

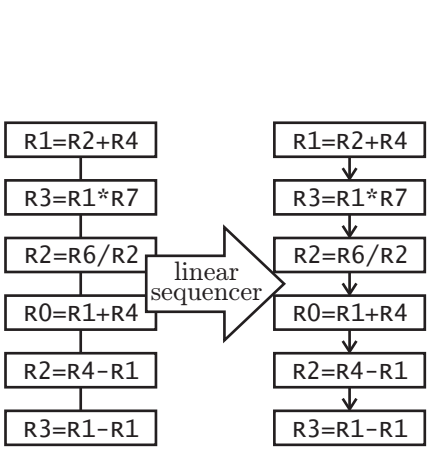


Fig. 4.28.a: Linear Genetic Programming phenotype and execution.

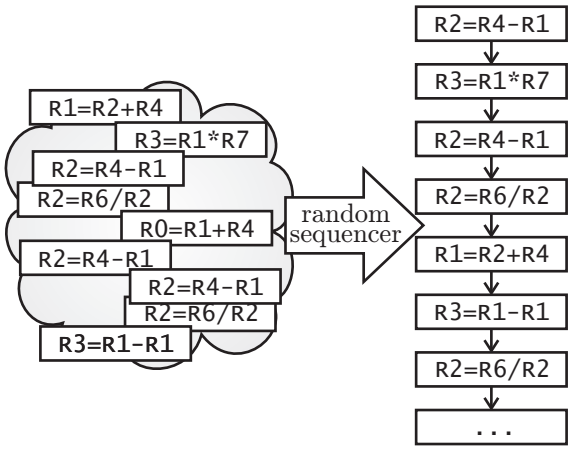


Fig. 4.28.b: Algorithmic Chemistry phenotype and execution.

Figure 4.28: The difference between linear Genetic Programming and Algorithmic Chemistries.

In Fig. 4.28.a we have illustrated the sequential structure of a normal program which might have been evolved in a linear Genetic Programming experiment. Whenever it is executed, be it for determining its fitness or later, as part of an application, the instructions are

[50] The linear Genetic Programming approach is outlined in Section 4.6 on page 191.

processed one after another, step by step. This execution scheme is common to all off-the-shelf PCs where the CPU uses an internal register (the instruction pointer) which points to the instruction to be executed next and which is incremented in this process.

Programs in the Algorithmic Chemistry representation can evolve essentially in the same way as programs in linear Genetic Programming do. As genotypes, exactly the same sequences of instructions can be used. This similarity, however, stops at the phenotypic level[51]. Here, the programs are considered as multisets which do not define any order on their elements (the instructions), as sketched in Fig. 4.28.b. When such a program is executed, a *random sequencer* draws one element from this set in each time step and executes it.[52]

This approach clearly leads to a θ-value of zero, since all positional dependencies amongst the instructions have vanished. As trade-off, however, there are a number of interesting side effects. Since programs no longer are sequences, there is, for example, no last instruction anymore. Thus, the Algorithmic Chemistry programs also have no clear "end". Therefore, the randomized execution step is performed for a fixed number of iterations – five times the number of instructions in [1258]. As pictured in Fig. 4.28.b, a certain instruction may occur multiple times in a program, which increases its probability of being picked for execution.

The biggest drawback of this approach is that the programs are no longer deterministic and their behavior and results may vary between two consecutive executions. Therefore, multiple independent runs should always be performed and the median or mean return value of them should be considered as the true result. Stripping the instructions of their order also will make it harder for higher-level constructs like alternatives or loops to evolve, let alone modules or functions. On the positive side, it also creates a large potential for parallelization and distribution which could be beneficial especially in multi-processor systems.

4.8.3 Soft Assignment

Another approach for reducing the epistasis is the soft assignment method (*memory with memory*) by McPhee and Poli [1385]. It implicitly targets semantic epistasis by weakening the way values are assigned to variables.

In traditional programs, instructions like `x=y` or `mov x, y` will completely overwrite the value of `x` with the value of `y`. McPhee and Poli replace this strict assignment semantic with

$$\mathtt{x}_{t+1} = \mathtt{y}_t \quad \equiv \quad \mathtt{x}_{t+1} \longleftarrow \gamma\mathtt{y}_t + (1-\gamma)\mathtt{x}_t \tag{4.3}$$

where $\mathtt{x}_{t+1}$ is the value that the variable `x` will have after and $\mathtt{x}_t$ its value before the assignment. $\mathtt{y}_t$ is the value of an arbitrary expression which is to be stored in `x`. The parameter γ is "a constant that indicates the *assignment hardness*" [1385]. For $\gamma = 1$, the assignments are completely overwriting as in normal programming and for $\gamma = 0$, the values of the variables cannot be changed. McPhee and Poli [1385] report that $\gamma = 0.7$ performed best (better than $\gamma = 1$) when applied to different symbolic regression[53] problems.

For mathematical or approximation problems, this approach is very beneficial. The drawback of programs using soft assignment is that, although they are deterministic, there are situations where precise values are required which they may not be able to compute. Assume, for instance, that an algorithm is to be evolved which returns the largest element `max` from an input list `l`. This program could contain an instruction like `if l[i]>max then max=l[i]`. If a γ-value smaller than one is applied, the final value in `max` will most likely be no element of the list.

[51] Although we distinguish between genotype and phenotype, no genotype-phenotype mapping is needed since the randomized execution can perfectly be performed on an array of instructions.

[52] Banzhaf and Lasarczyk [138] use the same approach for introducing a new recombination operator which creates an offspring by drawing instructions randomly from both parents.

[53] See Section 23.1 on page 397 for more information on symbolic regression.

4.8.4 Rule-based Genetic Programming

Besides the Algorithmic Chemistry approach of Lasarczyk and Banzhaf [1258, 1259] and soft assignments, there exists one very general class of evolutionary algorithms that elegantly circumvents positional epistasis[54]: the (Learning) Classifier Systems family [948, 946] which you can find discussed in Chapter 7 on page 233.

In the Pittsburgh LCS approach associated with Spears and De Jong [1926], a population of rule sets is evolved with a genetic algorithm [1912, 1926]. Each individual in this population consists of multiple classifiers (the rules) that transform input signals into output signals. The evaluation order of the rules in such a classifier system C plays absolutely no role except maybe for rules concerning the same output bits, i. e., $\theta(C) \approx 1$.[55]

The basic idea behind the Rule-based Genetic Programming approach is to use this knowledge to create a new program representation that retains high θ-values in order to become more robust in terms of reproduction operations [2181]. With RBGP, the aforementioned disadvantages (such as non-determinism) of Algorithmic Chemistries and soft assignments are completely circumvented. RBGP may be considered as a high-level LCS variant which introduces mightier concepts like mathematical operations. It furthermore exhibits a certain amount of non-uniform neutrality which, as we believe, is likely to increase the chance of finding better solutions.

We illustrate this new Genetic Programming method by using an example in Figure 4.29. Like in Pitt-style Learning Classifier Systems, the depicted programs consist of arbitrary many rules which can be encoded binary. A rule evaluates the values of the symbols in its condition part (left of $\Rightarrow$) and, in its action part, assigns a new value to one symbol or may invoke any other procedure provided in its configuration. In its structure, the RBGP language is similar to Dijkstra's Guarded Command Language[56] (GCL) [568].[57]

Genotype and Phenotype

Before the evolution in Rule-based Genetic Programming begins, the number of symbols and their properties must be specified as well as the possible actions. Each symbol identifies an integer variable which is either read-only or read-write. Some read-only symbols are defined for constants like `0` and `1`, for instance. The symbol `start` is only `1` during the first application of the rule set and becomes `0` afterwards (but may be written to by the program). Furthermore, a program can be provided with some general-purpose variables (`a` and `b` in the example). Additional symbols with special meanings can be introduced. For evolving distributed algorithms, for instance, an input symbol `in` where incoming messages will occur and a variable `out` from which outgoing messages can be transmitted from could be added. If messages should be allowed to contain more than one value, multiple such symbols have to be defined. These `out` symbols may trigger message transmission directly when written to as in Figure 4.29. Alternatively, a message can be sent by a special `send` action.

An action set containing mathematical operations like addition, subtraction, value assignment, and an equivalent to logical negation[58] is sufficient for many problems but may be extended arbitrarily. In conjunction with the constants `0` and `1` and the comparison operation, the evolutionary process can build arbitrary complex logical expression.

From the initial symbol and action specifications, the system can determine how many bits are needed to encode a single rule. A binary encoding where this is the size of the genes in variable-length bit string genotypes can then be used. With such simple genotypes, any possible nesting depth of condition statements and all possible logical operations can be

[54] You can find positional epistasis discussed in Section 4.8.1 on page 202

[55] θ as a measure for positional epistasis has been defined in Section 4.8.1 on page 202.

[56] `http://en.wikipedia.org/wiki/Guarded_Command_Language` [accessed 2008-07-24]

[57] I have to thank David R. White for this information – when devising Rule-based Genetic Programming, I didn't even know that the Guarded Command Language existed.

[58] In order to emulate a *logical not*, we use the expression `1-x` where `x` can be an arbitrary symbol.

Symbol	Encoding
0	0000, 1100
1	0001, 1101
start	0010, 1110
id	0011, 1111
netSize	0100
in	0101
out	0110
enterCS	0111
csLeft	1000
a	1001
b	1010

Comp.	Enc.
>	000
≥	001
=	010
≤	011
<	100
≠	101
true	110
false	111

Concat.	Enc.
∧	0
∨	1

Action	Enc.
= x+y	00
= x-y	01
= x	10
= 1-x	11

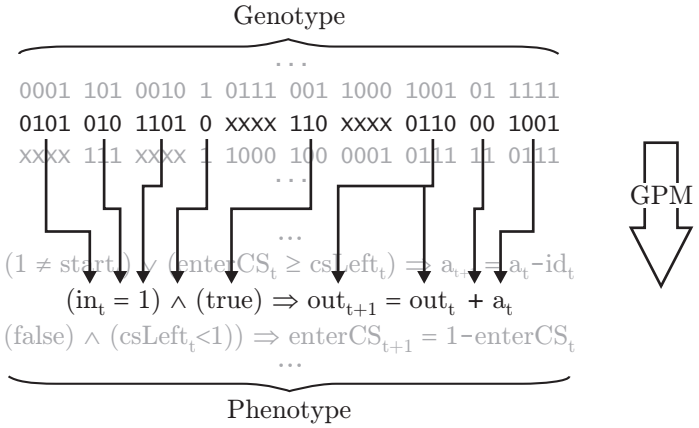


Figure 4.29: Example for a genotype-phenotype mapping in Rule-based Genetic Programming.

encoded. If needed, a tree-like program structure (as in Standard Genetic Programming) can be constructed from the rule sets, since each rule corresponds to a single conditional statement in a normal programming language.

There are similarities between our RBGP and some special types of LCSs, like the abstracted LCS by Browne and Ioannides [295] and the S-expression-based LCS by Lanzi and Perrucci [1250]. The two most fundamental differences lie in the semantics of both, the rules and the approach: In RBGP, a rule may directly manipulate symbols and invoke external procedures with (at most) two in/out-arguments. This includes mathematical operations like multiplication and division which do not exist a priori in LCSs but would have to evolve on basis of binary operations, which is, although possible, very unlikely. Furthermore, the individuals in RBGP are not classifiers but programs. Classifiers are intended to be executed once for a given situation, judge it, and decide upon an optimal output. A program, on the other hand, runs independently, asynchronously performs a computation, and interacts with its environment. Also, the syntax of RBGP is very extensible because the nature of the symbols and actions is not bound to specific data types but can easily be adapted to floating point computation, for instance.

Program Execution and Dimensions of Independence

The simplest method for executing a rule-based program is to loop over it in a cycle. Although this approach is sufficient for simulation purposes, it would result in a waste of CPU power on a real system. This consumption of computational power (and thus, energy) can be reduced very much if the conditional parts of the rules are only evaluated when one of the symbols that they access changes.

Positional Independence

Changes in the values of the symbols can either be caused by data incoming from the outside, like messages which are received (and stored in the `in`-symbols in our example) or by the actions invoked by the program itself. In RBGP, actions do not directly modify the values of the symbols but rather write their results to a temporary storage. After all actions have been processed, the temporary storage is committed to the real memory, as sketched in Figure 4.30. The symbols in the condition part and in the computation parts of the actions are annotated with the index t and those in the assignment part of the actions are marked with $t + 1$ in order to illustrate this issue.

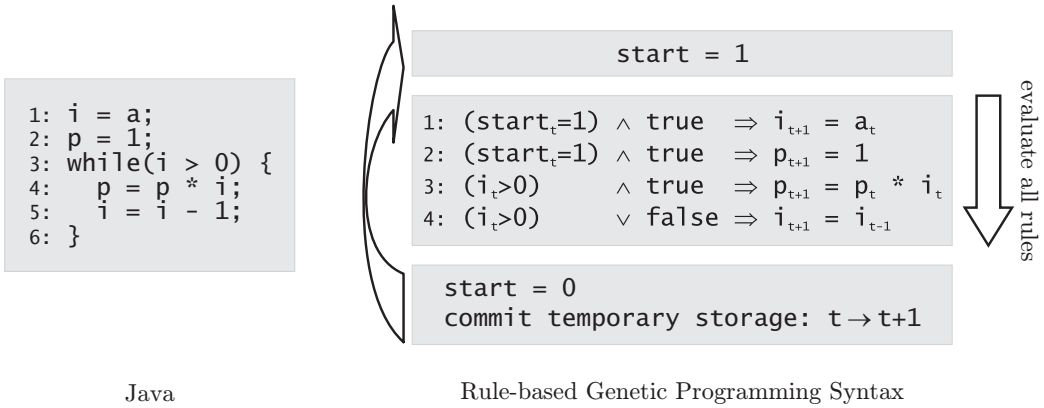


Figure 4.30: A program computing the faculty `p` of a natural number `a` in Java and RBGP syntax.

This approach allows for a great amount of disarray in the rules since the only possible positional dependencies left are those of rules which write to the same symbols. All other rules can be freely permutated without any influence on the behavior of the program. Hence, the positional epistasis in RBGP is very low.

Cardinality Independence

By excluding any explicit learning features (like the Bucket Brigade Algorithm[59] used in Learning Classifier Systems [942, 943]), we also gain insensitivity in terms of rule cardinality. It is irrelevant whether a rule occurs once, twice, or even more often in a program. If triggered, all occurrences of the rule use the same input data and thus, will write the same values to the temporary variable representing their target symbol. Assuming that an additional objective function which puts pressure into the direction of smaller programs is always imposed, superfluous rules will be wiped out during the course of the evolution anyway.

Neutrality

The existence of neutral reproduction operations can have a positive as well as negative influence on the evolutionary progress (see Section 1.4.5 on page 64). The positional and cardinality independence are clear examples of phenotypic neutrality and redundancy in RBGP. They allow a useful rule to be replicated arbitrarliy often in the same program without decreasing its functional fitness. This is likely to happen during crossover. By doing so, the conditional parts of the rule will (obviously) be copied too. Subsequent mutation operations may now slightly modify the rule and lead to improved behavior, i. e., act as

[59] The Bucket Brigade Algorithm is discussed in Section 7.3.8 on page 240.

exploitation operations. Based on the discussion of neutrality, we expect this form of non-uniform redundancy to have a rather positive effect on the evolution.

All these new degrees of freedom are achieved without most of the drawbacks that are inherent in Algorithmic Chemistries. The program flow is fully deterministic and so are its results. Like in Algorithmic Chemistries, it is harder to determine the number of steps needed for program execution, although we can easily detect the termination of local algorithms as the point where an execution does not lead to any change in the symbols.

Complex Statements

From the previous descriptions, it would seem that rule-based programs are strictly sequential, without branching or loops. This is not the case. Instead, a wide variety of complex computations can be expressed with them. Here we will give some intuitive examples for such program structures in RBGP syntax.

Complex Conditions

Assume that we have five variables `a` to `e` and want to express something like

```
1 if( (a<b) && (c>d) && (a<d) ) {
2   a += c;
3   c--;  }
```

Listing 4.9: A complex conditional statement in a C-like language.

We can do this in RBGP with four rules:

$$
\begin{array}{ll}
1 & \texttt{true} \ \wedge \ \texttt{true} \ \Rightarrow \mathtt{e}_{t+1} = 0 \\
2 & (\mathtt{a}_t < \mathtt{b}_t) \wedge (\mathtt{c}_t > \mathtt{d}_t) \Rightarrow \mathtt{e}_{t+1} = 1 \\
3 & (\mathtt{a}_t < \mathtt{d}_t) \wedge (\mathtt{e}_t = 1) \Rightarrow \mathtt{a}_{t+1} = \mathtt{a}_t + \mathtt{c}_t \\
4 & (\mathtt{a}_t < \mathtt{d}_t) \wedge (\mathtt{e}_t = 1) \Rightarrow \mathtt{c}_{t+1} = \mathtt{c}_t - 1
\end{array}
$$

Listing 4.10: The RBGP version of Listing 4.9.

Although this does not look very elegant, it fulfills the task by storing the result of the evaluation of the condition as logical value in the variable `e`. `e` will normally be `0` because of line 1 and is only set to `1` by rule 2. Since both rules write to the same temporary variable, the then-part of the condition in Listing 4.9 (lines 3 and 4 in Listing 4.10) will be reached in the next round if `a<d` holds too. Notice that the only positional dependency in Listing 4.10 is that rule 2 must always occur after rule 1. All rule permutations that obey this statement are equivalent (hence, $\theta = 0.5$) and so is Listing 4.11:

$$
\begin{array}{ll}
1 & (\mathtt{a}_t < \mathtt{d}_t) \wedge (\mathtt{e}_t = 1) \Rightarrow \mathtt{a}_{t+1} = \mathtt{a}_t + \mathtt{c}_t \\
2 & \texttt{true} \ \wedge \ \texttt{true} \ \Rightarrow \mathtt{e}_{t+1} = 0 \\
3 & (\mathtt{a}_t < \mathtt{b}_t) \wedge (\mathtt{c}_t > \mathtt{d}_t) \Rightarrow \mathtt{e}_{t+1} = 1 \\
4 & (\mathtt{a}_t < \mathtt{d}_t) \wedge (\mathtt{e}_t = 1) \Rightarrow \mathtt{c}_{t+1} = \mathtt{c}_t - 1 \\
5 & \texttt{true} \ \wedge \ \texttt{true} \ \Rightarrow \mathtt{e}_{t+1} = 0 \\
6 & (\mathtt{a}_t < \mathtt{d}_t) \wedge (\mathtt{e}_t = 1) \Rightarrow \mathtt{a}_{t+1} = \mathtt{a}_t + \mathtt{c}_t \\
7 & (\mathtt{a}_t < \mathtt{b}_t) \wedge (\mathtt{c}_t > \mathtt{d}_t) \Rightarrow \mathtt{e}_{t+1} = 1 \\
8 & (\mathtt{a}_t < \mathtt{d}_t) \wedge (\mathtt{e}_t = 1) \Rightarrow \mathtt{c}_{t+1} = \mathtt{c}_t - 1
\end{array}
$$

Listing 4.11: An equivalent alternative version of Listing 4.10.

Loops

Loops in RBGP can be created in the very same fashion.

```
1 b = 1;
2 for(a=c; a>0; a--) {
3   b *= a;
4   }
```

Listing 4.12: A loop in a C-like language.

The loop defined in Listing 4.12 can be expressed in RBGP as outlined in Listing 4.13, where we use the `start`-symbol (line 1 and 2) to initialize `a` and `b`. As its name suggests, `start` is only `1` at the very beginning of the program's execution and `0` afterwards (unless modified by an action).

```
1 (start_t > 0) ∨    false    ⇒ a_{t+1} = c_t
2    true    ∧ (start_t > 0) ⇒ b_{t+1} = 1
3    (a_t > 0) ∧    true     ⇒ a_{t+1} = a_t − 1
4   false    ∨    (a_t > 0) ⇒ b_{t+1} = b_t * a_t
```

Listing 4.13: The RBGP-version of Listing 4.12.

Here, no positional or cardinality restrictions occur at all, so Listing 4.14 is equivalent to Listing 4.13 and $\theta = 1$.

```
1    false    ∨    (a_t > 0) ⇒ b_{t+1} = b_t * a_t
2    true     ∧ (start_t > 0) ⇒ b_{t+1} = 1
3 (start_t > 0) ∨    false    ⇒ a_{t+1} = c_t
4    false    ∨    (a_t > 0) ⇒ b_{t+1} = b_t * a_t
5    (a_t > 0) ∧    true     ⇒ a_{t+1} = a_t − 1
6 (start_t > 0) ∨    false    ⇒ a_{t+1} = c_t
```

Listing 4.14: An equivalent, alternative version of Listing 4.13.

Extended Rule-based Genetic Programming

We have shown that, although looking rather simple, the primitives of Rule-based Genetic Programming are mighty enough to express many of the constructs known from high-level programming languages. However, in the original RBGP approach, there are some inherent limitations.

Its most obvious drawback is the lack of Turing completeness. In order to visualize this problem, imagine the restriction that only simple types like integer variables and parameters were allowed was imposed on the Java programming language. Then, no complex types like arrays could be used. In this case, it would become hard to create programs which process data structures like lists, since single variables for each and every of their elements would have to be defined and accessed independently. Writing a method for sorting a list of arbitrary length would even become impossible.

The same restrictions hold in Rule-based Genetic Programming as introduced in Section 4.8.4 – the symbols there resemble plain integer variables. In Java, the problems stated above are circumvented with arrays, a form of memory which can be accessed indirectly. Adding indirect memory access to the programming language forming the basis of Rule-based Genetic Programming would allow the evolution of more complex algorithms – matter of fact, this is the standard approach for creating Turing-complete representations. We therefore define the notation $[\mathtt{a}_t]_t$, which stands for the value of the $(\mathtt{a}_t)$th symbol at time step t in the ordered list of all symbols. In this, it equals a simple pointer dereferentiation (`*a`) in the C language.

With this extension alone, it becomes possible to use the RBGP language for defining list sorting algorithms, for instance. Assume that the following symbols $(\mathtt{i}_0, \mathtt{i}_1, .., \mathtt{i}_{n-1}, \mathtt{l}, \mathtt{a}, \mathtt{b})$ have been defined. The symbols $\mathtt{i}_0$ to $\mathtt{i}_{n-1}$ constitute the memory which can be used to store the list elements and `l` is initialized with the length of the list, i. e., the number of the `i`-elements actually used (which has to be smaller or equal to n). `a` and `b` are multi-purpose variables. In the symbol list, $\mathtt{i}_0$ is at position 0, `l` at position n, `a` at index $n + 1$ and so on. With very little effort, Listing 4.15 can be defined which performs a variant of selection sort[60]. Notice that, since writing to variables is not committed before all rules were applied, no explicit temporary variable is required in the third and fourth rule.

[60] http://en.wikipedia.org/wiki/Selection_sort [accessed 2008-05-09]

```
1 (start_t > 0)  ∧      true      ⇒    a_{t+1} = 0
2 (start_t > 0)  ∧      true      ⇒    b_{t+1} = 0
3     (a_t < l_t) ∧ ([a_t]_t < [b_t]_t) ⇒ [a_t]_{t+1} = [b_t]_t
4     (a_t < l_t) ∧ ([a_t]_t < [b_t]_t) ⇒ [b_t]_{t+1} = [a_t]_t
5     (b_t ≥ a_t) ∧    (a_t < l_t)    ⇒    a_{t+1} = a_t + 1
6     (b_t < a_t) ∧      true         ⇒    b_{t+1} = b_t + 1
7     (b_t ≥ a_t) ∧    (a_t < l_t)    ⇒    b_{t+1} = 0
```

Listing 4.15: A simple selection sort algorithm written in the eRBGP language.

Listing 4.10, one of our previous examples shows another feature of RBGP which might prove troublesome: The condition part of a rule always consists of two single conditions. This is totally unimportant as long as a logical expression to be represented contains only one or two conditions. (If it needs only one, the second condition of the rule may be set to `true` and concatenated with an ∧-operator.) In Listing 4.10, however, we try to represent the conditional statement from Listing 4.9 which consists of three conditions. In order to do so, we needed to introduce the additional symbol `e`.

Here we can draw an analogy to the human memory[61] which may be divided into procedural[62] (implicit[63]) memory [848, 1715, 1818, 2229] storing, for instance, motor skills and declarative[64] (explicit[65]) memory [1145, 1155] holding facts and data. In comparison with RBGP, we would find that the expressiveness of the equivalent of the procedural memory in RBGP is rather limited, which needs to be mitigated by using more of it and storing additional information in the declarative memory. We used this approach when translating Listing 4.9 to Listing 4.10, for instance. This issue can be compared to a hypothetical situation in which we were not able to learn the complete motion of lifting a jar to our lips and instead, could only learn how to lift a jar from the table and how to move an already lifted jar to our mouth while needing to explicitly remember that both moves belong together.

Admittedly, this analogy may be a bit farfetched, but it illustrates that Rule-based Genetic Programming could be forced to go through a seemingly complex learning process for building a simple algorithm under some circumstances. We therefore extend its expressiveness by dropping the constraints on the structure of its rules which increases the number of ways that RBGP can be utilized for representing complicated expressions. The ability of using genetic algorithms with fixed-size genes for evolving rule-based programs, however, has to be traded in in order to facilitate this extension. Additionally, this extension might bring back some of the epistasis which we had previously successfully decreased.

```
1 ((a_t < b_t) ∧ ((c_t > d_t) ∧ (a_t < d_t))) ⇒ a_{t+1} = (a_t + c_t)
2 ((a_t < b_t) ∧ ((c_t > d_t) ∧ (a_t < d_t))) ⇒ c_{t+1} = (c_t − 1)
```

Listing 4.16: The eRBGP version of Listing 4.9 and Listing 4.10.

```
1 (start_t ≠ 0) ⇒ b_{t+1} = 1
2 (start_t ≠ 0) ⇒ c_{t+1} = a_t
3     (a_t > 0) ⇒ a_{t+1} = (a_t − 1)
4     (a_t > 0) ⇒ b_{t+1} = (b_t ∗ a_t)
```

Listing 4.17: The eRBGP version of Listing 4.12 and Listing 4.13.

In Listing 4.16 and Listing 4.17, we repeat the RBGP examples Listing 4.10 and Listing 4.13 – this time in eRBGP syntax. As already mentioned, we cannot use a simple genetic algorithm to evolve these programs since their structure does not map to a fixed gene size anymore. However, tree-based Standard Genetic Programming as discussed in Section 4.3 can perfectly fulfill this purpose. Listing 4.16, for instance, fits to the tree phenotype depicted in Figure 4.31.

[61] http://en.wikipedia.org/wiki/Memory [accessed 2008-05-08]
[62] http://en.wikipedia.org/wiki/Procedural_memory [accessed 2008-05-08]
[63] http://en.wikipedia.org/wiki/Implicit_memory [accessed 2008-05-08]
[64] http://en.wikipedia.org/wiki/Declarative_memory [accessed 2008-05-08]
[65] http://en.wikipedia.org/wiki/Explicit_memory [accessed 2008-05-08]

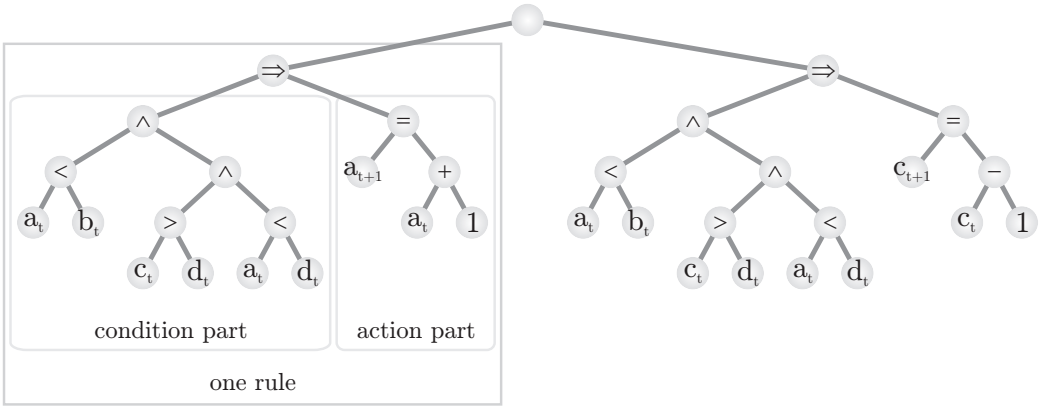


Figure 4.31: The tree phenotype (and genotype) of Listing 4.16.

With these changes, Extended Rule-based Genetic Programming becomes much more powerful in comparison with plain Rule-based Genetic Programming and is now able to evolve arbitrary algorithms and data structures. Also, the proof for Turing completeness of Genetic Programming languages with indexed memory by Teller [2012] can easily be adapted to Extended Rule-based Genetic Programming (as well as the simpler strategy by Nordin and Banzhaf [1543]).

4.9 Artificial Life and Artificial Chemistry

It is not hard to imagine what artificial life is. Matter of fact, I assume that everyone of us has already seen numerous visualizations and simulations showing artificial creatures. Even some examples from this book like the Artificial Ant may well be counted to that category.

Definition 4.4 (Artificial Life). Artificial life[66], also abbreviated with *ALife* or *AL*, is a field of research that studies the general properties of life by synthesizing and analyzing life-like behavior [165].

Definition 4.5 (Artificial Chemistry). The area of artificial chemistries[67] subsumes all computational systems which are based on simulations of entities similar to molecules and the reactions amongst them.

According to Dittrich et al. [575], an artificial chemistry is defined by a triple (S, R, A), where $S = \{s_1, s_2, s_3, \dots\}$ is the set of possible molecules S, R is the set of reactions that can occur between them, and A is an algorithm defining how the reactions are applied.

Artificial chemistry and artificial life strongly influence each other and often merge into each other. The work of Hutton [977, 978, 979], for example, focuses on generating and evolving self-replicating molecules and cells. There also exists real-world applications of artificial chemistry in many areas of chemistry, computer networking, economics, and sociology. The Algorithmic Chemistries, which we have analyzed in Section 4.8.2 on page 205, are also closely related to artificial chemistries, for instance. In this section, we will discuss some more artificial life and artificial chemistry approaches in the context of Genetic Programming.

[66] http://en.wikipedia.org/wiki/Artificial_life [accessed 2007-12-13]
[67] http://en.wikipedia.org/wiki/Artificial_chemistry [accessed 2008-05-01]

4.9.1 Push, PushGP, and Pushpop

In 1996, early research in self-modification or self-evolution of programs has been conducted by Spector and Stoffel [1935] in form of the *ontogenic* extension of their HiGP system [1965]. Basically, they extended the programs evolved with linear Genetic Programming method with the capabilities of shifting and copying segments of their code at runtime.

About half of a decade later, Spector [1930] developed *Push*, a stack-based programming language especially suitable for evolutionary computation [1930, 1934, 1932]. Programs in that language can be evolved by adapting existing Standard Genetic Programming systems (as done in *PushGP*) or, more interestingly, by themselves in an autoconstructive manner, which has been realized in the *Pushpop* system. Currently, the Push language is currently available in its third release, *Push3* [1938, 1939].

A Push program is either a single instruction, a literal, or a sequence of zero or more Push programs inside parentheses.

```
program ::= instruction | literal | ( {program} )
```

An instruction may take zero or more arguments from the stack. If insufficient many arguments are available, it acts as `NOOP`, i. e., does nothing. The same goes if the arguments are invalid, like when a division by zero would occur.

In Push, there is a stack for each data type, including integers, Boolean values, floats, name literals, and code itself. The instructions are usually named according to the scheme `<type>.<operation>`, like `INTEGER.+`, `BOOLEAN.DUP`, and so on. One simple example for a Push program borrowed from Spector [1932], Spector et al. [1938] is

```
 ( 5 1.23 INTEGER.+ ( 4 ) INTEGER.- 5.67 FLOAT.* )
Which will leave the stacks in the following states:
FLOAT STACK  : (6.9741)
CODE STACK   : ( ( 5 1.23 INTEGER.+ ( 4 ) INTEGER.- 5.67
                   FLOAT.* ) )
INTEGER STACK: (1)
```

Listing 4.18: A first, simple example for a Push program.

Since all operations take their arguments from the corresponding stacks, the initial `INTEGER.+` does nothing because only one integer, `5`, is available on the `INTEGER` stack. `INTEGER.-` subtracts the value on the top of `INTEGER` stack (`4`) from the one beneath it (`5`) and leaves the result (`1`) there. On the float stack, the result of the multiplication `FLOAT.*` of `1.23` and `5.67` is left while the whole program itself resides on the `CODE` stack.

Code Manipulation

One of the most interesting features of Push is that we can easily express new forms of control flow or self-modifying code with it. Here, the `CODE` stack and, since Push3, the `EXEC` stack play an important role. Let us take the following example from [1930, 1938]:

```
(CODE.QUOTE (2 3 INTEGER.+) CODE.DO)
```

Listing 4.19: An example for the usage of the CODE stack.

The instruction `CODE.QUOTE` leads to the next piece of code (`(2 3 INTEGER.+)` in this case) being pushed onto the `CODE` stack. `CODE.DO` then invokes the interpreter on whatever is on the top of this stack. Hence, `2` and `3` will land on the `INTEGER` stack as arguments for the following addition. In other words, Listing 4.19 is just a complicated way to add $2 + 3 = 5$.

```
(CODE.QUOTE
  (CODE.QUOTE (INTEGER.POP 1)
   CODE.QUOTE (CODE.DUP INTEGER.DUP 1 INTEGER.- CODE.DO INTEGER.*)
   INTEGER.DUP 2 INTEGER.< CODE.IF)
```

```
5 CODE.DO)
```

Listing 4.20: Another example for the usage of the CODE stack.

Listing 4.20 outlines a Push program using a similar mechanism to compute the factorial of an input provided on the `INTEGER` stack. It first places the whole program on the `CODE` stack and executes it (with the `CODE.DO` at its end). This in turn leads on the code in lines 2 and 3 being placed on the code stack. The `INTEGER.DUP` instruction now duplicates the top of the `INTEGER` stack. Then, `2` is pushed and the following `INTEGER.<` performs a comparison of the top two elements on the `INTEGER` stack, leaving the result (`true` or `false`) on the `BOOLEAN` stack. The instruction `CODE.IF` executes one of the top two items of the `CODE` stack, depending on the value it finds on the `BOOLEAN` stack and removes all three elements. So in case that the input element was smaller than 2, the top element of the `INTEGER` stack will be removed and `1` will be pushed into its place. Otherwise, the next instruction `CODE .DUP` duplicates the whole program on the `CODE` stack (remember, that everything else has already been removed from the stack when `CODE.IF` was executed). `INTEGER.DUP` copies the top of the `INTEGER` stack, `1` is stored and then subtracted from this duplicate. The result is then multiplied with the original value, leaving the product on the stack. So, Listing 4.20 realizes a recursive method to compute the factorial of a given number.

Name Binding

As previously mentioned, there is also a `NAME` stack in the Push language. It enables us to bind arbitrary constructs to names, allowing for the creation of named procedures and variables.

```
1 ( DOUBLE CODE.QUOTE ( INTEGER.DUP INTEGER.+ ) CODE.DEFINE )
```

Listing 4.21: An example for the creation of procedures.

In Listing 4.21, we first define the literal `DOUBLE` which will be pushed onto the `NAME` stack. This definition is followed by the instruction `CODE.QUOTE`, which will place code for adding an integer number to itself on the `CODE` stack. This code is then assigned to the name on top of the `NAME` stack (`DOUBLE` in our case) by `CODE.DEFINE`. From there on, `DOUBLE` can be used as a procedure.

The EXEC Stack

Many control flow constructs of Push programs up to version 2 of the language are executed by similar statements in the interpreter. Beginning with Push3, all instructions are pushed onto the new `EXEC` stack prior their invocation. Now, now additional state information or flags are required in the interpreter except from the stacks and name bindings. Furthermore, the `EXEC` stack supports similar manipulation mechanisms like the `CODE` stack.

```
1 ( DOUBLE EXEC.DEFINE ( INTEGER.DUP INTEGER.+ ) )
```

Listing 4.22: An example for the creation of procedures similar to Listing 4.21.

The `EXEC` stack is very similar to the `CODE` stack, except that its elements are pushed in the inverse order. The program in Listing 4.22 is similar to Listing 4.21 [1938].

Autoconstructive Evolution

Push3 programs can be considered as tree structures and hence be evolved using standard Genetic Programming. This approach has been exercised with the *PushGP* system [1930, 1934, 1933, 463, 1745]. However, the programs can also be equipped with the means to create their own offspring. This idea has been realized in a software called *Pushpop* [1930, 1934, 1931]. In Pushpop, whatever is left on top of the `CODE` stack after a programs execution is regarded as its child. Programs may use the above mentioned code manipulation facilities to create their descendants and can also access a variety of additional functions, like

1. `CODE.RAND` pushes newly created random code onto the `CODE` stack.
2. `NEIGHBOR` takes an integer n and returns the code of the individual in distance n. The population is defined as a linear list where siblings are grouped together.
3. `ELDER` performs a tournament between n individuals of the previous generation and returns the winner.
4. `OTHER` performs a tournament between n individuals of the current generation, comparing individuals according to their parents fitness, and returns the winner.

After the first individuals able to reproduce have been evolved the system can be used to derive programs solving a given problem. The only external influence on the system is a selection mechanism required to prevent uncontrolled growth of the population by allowing only the children of fit parents to survive.

4.9.2 Fraglets

In his seminal work, Tschudin [2058] introduced Fraglets[68], a new artificial chemistry suitable for the development and even evolution of network protocols. Fraglets represent an execution model for communication protocols which resembles chemical reactions.

How do Fraglets work?

From the theoretical point of view, the Fraglet approach is an instance of Post's string rewriting systems[69] [1672] and Gamma systems [127, 131, 128, 129, 130]. Fraglets are symbolic strings of the form $[s_1 : s_2 : \ldots : s_n]$. The symbols $\mathtt{s}_i$ either represent control information or payload. Each node in the network has a *Fraglet store* which corresponds to a reaction vessel in chemistry. Such vessels usually contain equal molecules multiple times and the same goes for Fraglet stores which can be implemented as multisets keeping track on the multiplicity of the Fraglets they contain.

Tschudin [2058] defines a simple prefix programming language with a fixed instruction set comprising transformation and reaction rules for Fraglets. Transformations like `dup` and `nop` modify a single Fraglet whereas reactions such as `match` and `matchP` combine two Fraglets. For the definition of these rules in Table 4.2, we will use the syntax $[\mathtt{s}_1 : \mathtt{s}_2 : \ldots : \mathtt{tail}]$ where $\mathtt{s}_i$ is a symbol and `tail` is a possibly empty sequence of symbols.[70]

Obviously, the structure of Fraglets is very different from other program representations. There are no distinguishable modules or functions, no control flow statements such as jumps or function invocations, and no distinction exists between memory and code. Nevertheless, the Fraglet system is powerful, has a good expressiveness, and there are indications that it is likely Turing-complete [571].

Examples

After defining the basics of the Fraglet approach, let us now take a look on a few simple examples.

Election

Election in a distributed system means to select one node in the network and to ensure that all nodes receive knowledge of the ID of the selected one. One way to perform such an election is to determine the maximum ID of all nodes, which is what we will do here.

[68] See `http://en.wikipedia.org/wiki/Fraglets` [accessed 2008-05-02] and `http://www.fraglets.net/` [accessed 2008-05-02] for more information.

[69] `http://en.wikipedia.org/wiki/Post_canonical_system` [accessed 2008-05-02]

[70] See `http://www.fraglets.net/frag-instrset-20070924.txt` [accessed 2008-05-02] for the full instruction set as of 2007-09-24.

tag	transformation/reaction
► basic transformations	
dup	$[dup : t : a : tail] \longrightarrow [t : a : a : tail]$ duplicate a single symbol
exch	$[exch : t : a : b : tail] \longrightarrow [t : b : a : tail]$ swap two tags
fork	$[fork : a : b : tail] \longrightarrow [a : tail]\,,\ [b : tail]$ copy Fraglet and prepend different header symbols
nop	$[nop : tail] \longrightarrow [tail]$ does nothing (except consuming the instruction tag)
null	$[null : tail] \longrightarrow \emptyset$ destroy a Fraglet
pop2	$[pop2 : h : t : a : tail] \longrightarrow [h : a]\,,\ [t : tail]$ pop head element a out of a list $[a : b : tail]$
split	$[split : tail_1 : * : tail_2] \longrightarrow [tail_1]\,,\ [tail_2]$ break a Fraglet into two at the first occurrence of $*$
► arithmetic transformations	
sum	$[sum : t : \{m\} : \{n\} : tail] \longrightarrow [t : \{m+n\} : tail]$ an operation comparing two numbers
lt	$[lt : yes : no : \{a\} : \{b\} : tail] \longrightarrow \begin{cases} [yes : \{a\} : \{b\} : tail] & \text{if } a < b \\ [no : \{a\} : \{b\} : tail] & \text{otherwise} \end{cases}$ a logic operation comparing two numbers a and b
► communication primitives	
broadcast	$[broadcast : tail] \longrightarrow {}_{\mathbf{n}}[tail]$ broadcast *tail* to all nodes $\mathbf{n}$ in the network $\mathbf{N}$
send	$[send : dest : tail] \longrightarrow {}_{dest}[tail]$ send *tail* to a single node *dest*
node	${}_{n}[node : t : tail] \longrightarrow {}_{n}[t : \{\mathrm{id}(\mathbf{n})\} : tail]$ obtain the ID $\mathrm{id}(\mathbf{n})$ of the current node $\mathbf{n}$
► reactions	
match	$[match : a : tail_1]\,,\ [a : tail_2] \longrightarrow [tail_1 : tail_2]$ two Fraglets react, their tails are concatenated
matchP	$[matchP : a : tail_1]\,,\ [a : tail_2] \longrightarrow [matchP : a : tail_1]\,,\ [tail_1 : tail_2]$ "catalytic match", i. e., the matchp rule persists

Table 4.2: Some Fraglet instructions (from [2058] and `http://www.fraglets.net/` (2008-05-02)).

First of all, we define five additional symbols A, B, C, L, and R. These symbols do not react on their own behalf, i. e., $[A : tail] \longrightarrow [A : tail]$. After a bootstrap reaction, each node in the network will contain a Fraglet of the form $[L : \{id\}]$ containing the identifier id of the node that it thinks has won/currently leads in the election. It broadcasts this information to its neighbors, which will receive it in the form of the Fraglet $[R : \{id\}]$. A, B, and C have no further meaning. The election algorithm is constituted by six Fraglets $[node : L]$ which creates the first L-type Fraglet at bootstrap, $[matchP : L : fork : L : A]$ and $[matchP : A : broadcast : R]$ which are used to transmit the highest node ID currently known in a R-type Fraglet, $[matchP : R : match : L : lt : B : C]$ which initiates a comparison with incoming Fraglets of that type, and $[matchP : B : pop2 : null : L]$ and $[matchP : B : pop2 : L : null]$ evaluating the outcome of this comparison and producing a new L-type Fraglet. Figure 4.32 shows the flow of reactions in this system, which will finally lead to all nodes knowing the highest node ID in the network.

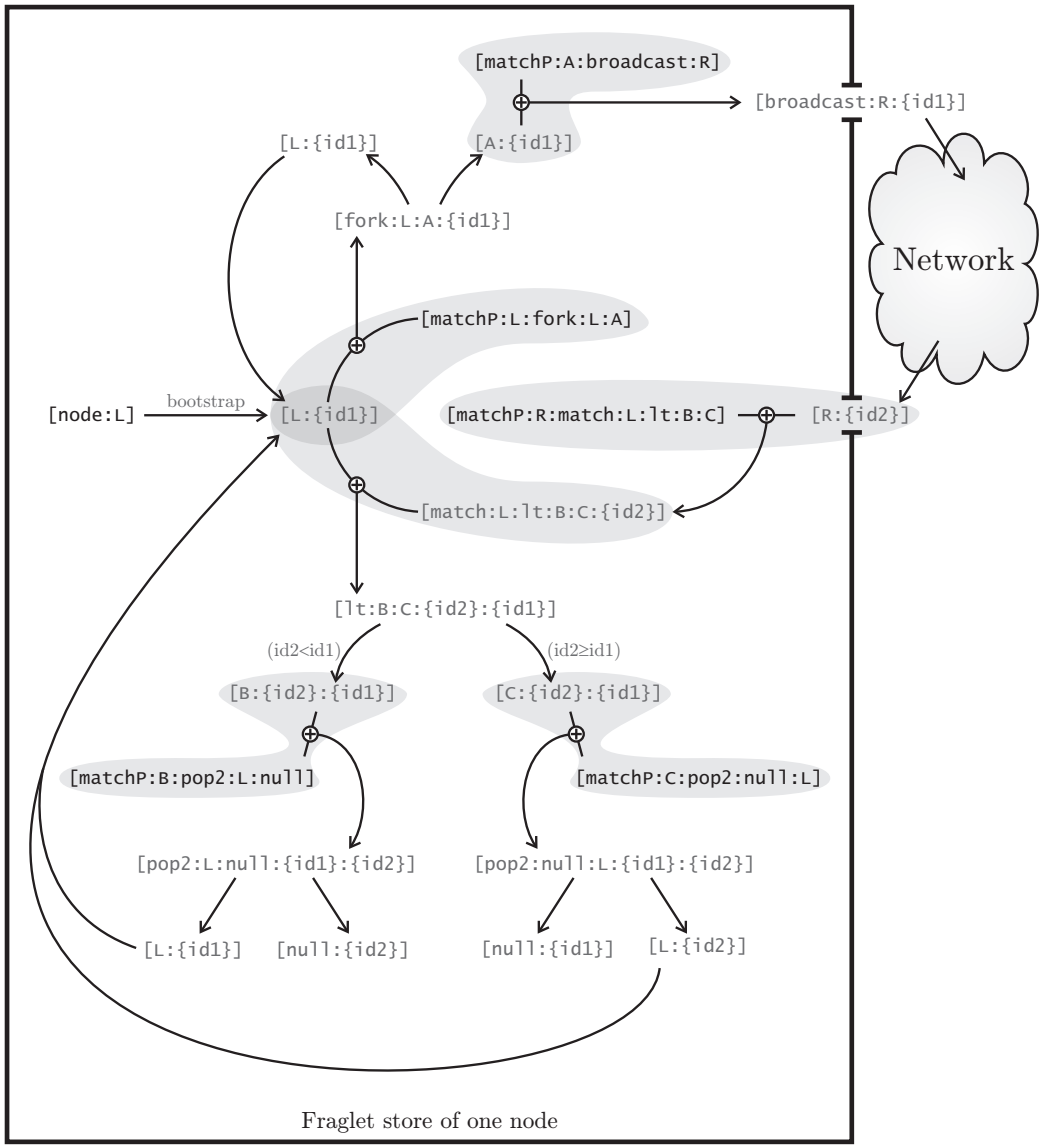


Figure 4.32: A Fraglet-based election algorithm

Quines

Definition 4.6 (Quine). A *quine*[71] is a computer program which produces a copy of itself (or its source code) as output.

From Kleene's second recursion theorem[72] [1148], it follows that quines can be defined in each Turing-complete language. Yamamoto et al. [2276, 1399] have introduced quine Fraglets like the one in Figure 4.33 as vehicle for self-replicating and self-modifying programs.

Fraglets as a program representation are predestined for evolutionary protocol synthesis. Indeed, they have a low positional epistasis (see Section 4.8.1 on page 202), since the order of the Fraglets in the Fraglet store plays no role. The order of the single commands inside a Fraglet, however, is significant. In Section 23.2.2 on page 404, we discuss the application of

[71] http://en.wikipedia.org/wiki/Quine_%28computing%29 [accessed 2008-05-04]
[72] http://en.wikipedia.org/wiki/Kleene%27s_recursion_theorem [accessed 2008-05-04]

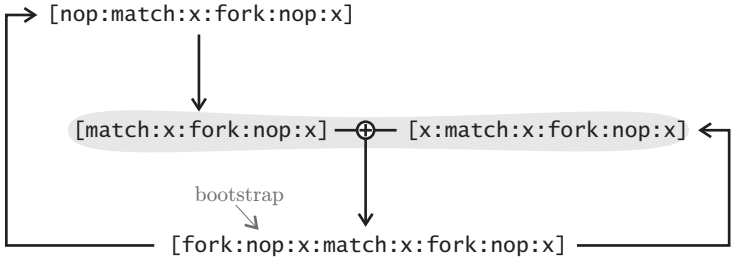


Figure 4.33: A simple quine Fraglet (borrowed from [2276])

Fraglets for protocol evolution based on the work by Tschudin [2058] and in Section 23.2.2 on page 404, the joint work of Yamamoto and Tschudin [2275] on online adaptation of Fraglet protocols is outlined.

4.10 Problems Inherent in the Evolution of Algorithms

Genetic Programming can be utilized to breed programs or algorithms and programs suitable for a given problem class. In order to guide such an evolutionary process, these programs bred have to be evaluated. They are assessed in terms of functional and non-functional requirements. The functional properties comprise all features regarding how good the algorithm solves the specified problem and the non-functional aspects are concerned with, for example, its size and memory consumption. Normally, a set $F = \{f_1, .., f_n\}$ of objective functions is specified in order to map these attributes to the subsets $Y_1, .., Y_n$ of the real numbers $\mathbb{R}$.

4.10.1 Correctness of the Evolved Algorithms

Introduction

Genetic Programming can be utilized to breed programs or algorithms suitable for a given problem class. In order to guide such an evolutionary process, the synthesized programs have to be evaluated, i. e., assessed in terms of functional and non-functional requirements. The functional properties comprise all features regarding how good a program solves the specified problem and the non-functional aspects are concerned with, for example, its size and memory consumption. Often, a set $F = \{f_1, .., f_n\}$ of objective functions is specified in order to map these attributes to the real numbers.

The Problems

In Genetic Programming, some of the non-functional objective values such as the size of the evolved programs can easily be computed. Determining their functional utility, however, cannot be achieved by any arithmetically closed function or algorithm – at least if a Turing-complete representation is chosen – since this would be an instance of the Entscheidungsproblem[73] [496] as well as of the Halting Problem[74] [1894].

[73] http://en.wikipedia.org/wiki/Entscheidungsproblem [accessed 2007-07-03]
[74] http://en.wikipedia.org/wiki/Halting_problem [accessed 2007-07-03]

Entscheidungsproblem

The *Entscheidungsproblem*, formulated by David Hilbert [926, 927], asks for an algorithm that, if provided with a description of a formal language and a statement in that language, can decide whether or not the statement holds [2219]. In the case of Genetic Programming, the formal language is the language in which the programs are evolved, i. e., the problem space, and the statements are the programs themselves. Church [407, 408] and Turing [2065, 2066] both proved that an algorithm solving the Entscheidungsproblem cannot exist.

No Exhaustive Testing

It is not possible to use some kind *algorithm* in order to determine whether the evolved programs will provide correct results. Thus, training cases, (simulation) scenarios in which a program is executed test-wise, must be used to find out whether it is suitable for the given problem. *Software Testing* is a very important field in software engineering [168, 664, 784, 1090]. The core problem of testing programs for their functionality and performance is the size of the input space. Assume that we pursued the evolution of a program that takes two integer numbers (32 bit) as input and computes another one as output. For testing this program with all possible inputs, $2^{32} * 2^{32} = 2^{64} = 18\,446\,744\,073\,709\,551\,616$ single tests would be required. Even if each test run took only 1µs, exhaustive testing would take approximately 584 542 years. In most practical applications, the input space is much larger. Exhaustive testing of the evolved algorithms is thus not feasible in virtuall all cases.

Instead, we can only pick a very small fraction of the possible test scenarios for training and hope that they will provide significant results. The probability that this will happen depends very much on the method with which the training cases are selected.

Most often, one cannot be sure whether evolved behavioral patterns (or algorithms) are perfect and free from errors in all possible situations. Here, nature indeed has the same problem as the noble-minded scientists who apply Genetic Programming, as the following small analogy[75] will show.

The Monkey and the Orange Consider a certain scheme in the behavior of monkeys. If a monkey sees or smells something delicious in, for example, a crevice, it sticks its hand in, grabs the item of desire and pulls it out. This simple behavior itself is quite optimal and has served the monkeys well for generations. With the occurrence of *homo sapiens*, the situation changed. African hunters still use this behavior against the monkeys by creating a situation that was never relevant during its "evolutionary testing period": They slice a coconut in half and put a hole in one side just big enough for a monkey's hand to fit through. Now they place an orange between the two coconut halves, tie them closely together and secure the trap with a rope to a tree. Sooner or later, a monkey will smell the orange, find the coconut with the hole, stick its hand inside and grab the fruit. However, with the orange in its fist, it cannot pull the hand out anymore. The hunter can now easily catch the monkey, to whom it never occurs that it could let go the fruit and save its life.

In other words, although evolutionary algorithms like Genetic Programming may provide good solutions for many problems, their results still need to be analyzed and interpreted by somebody at least a bit more cunning than an average monkey.

This implies that the solutions need to be delivered in a human-readable way. In some optimization problems, we may, however, choose a representation for the solution candidates which is very hard to understand for human beings but more suitable for evolution. Hence, it is not always possible to perform a sanity check on the evolved programs by hand.

[75] It is hard to find references confirming this story. It occurred in one scene of the movie *Animals are beautiful people* by Uys [2085], roughly resembles one of Aesop's fables [14], and is mentioned on the Wikipedia [2219] page on coconuts from 2007-07-21 (`http://en.wikipedia.org/w/index.php?title=Coconut&oldid=146038518`) where they had the same problem and eventually removed the corresponding text. But regardless whether it is just an urban legend or not – it is a nice story.

Halting Problem

The *Halting Problem* is basically an instance of the Entscheidungsproblem and asks for an algorithm that decides whether another algorithm will terminate at some point in time or runs forever if provided with a certain, finite input. Again, Turing [2065, 2066] proved that a general algorithm solving the Halting Problem cannot exist in general. One possible way to show this is to use a simple counter-example: Assume that a correct algorithm *doesHalt* exists (as presumed in Algorithm 4.2) which takes a program *algo* as input and determines whether it will terminate or not. It is now possible to specify a program *trouble* which, in turn, uses *doesHalt* to determine if it will halt at some point in time. If *doesHalt* returns `true`, *trouble* loops forever. Otherwise it halts immediately. In other words, *doesHalt* cannot return the correct result for *trouble* and hence, cannot be applied universally. Thus, it is not possible to solve the Halting Problem algorithmically for Turing-complete programs in a Turing-complete representation. One consequence of this fact is that there are no means to determine when an evolved program will terminate or whether it will do so at all (if its representation allows infinite execution, that is) [2011, 2254]. Langdon and Poli [1243] have shown that in Turing-complete linear Genetic Programming systems, most synthesized programs loop forever and the fraction of halting programs of size *length* is proportional to $\sqrt{length}$, i. e., small.

Algorithm 4.2: *Halting Problem: reductio ad absurdum*

```
1  begin
2  |  doesHalt(algo) ∈ {true, false}
3  |  begin
4  |  |  ...
5  |  end
6  |  Subalgorithm trouble()
7  |  begin
8  |  |  if doesHalt(trouble) then
9  |  |  |  while true do
10 |  |  |  |  ...
11 |  end
12 end
```

Countermeasures

Against the Entscheidungsproblem

For general, Turing-complete program representations, neither exhaustive testing nor algorithmic detection of correctness is possible.

Model Checking Model checking[76] techniques [413, 1483] have made great advance since the 1980s. According to Clarke and Emerson [412], *"Model checking is an automated technique that, given a finite-state model of a system and a logical property, systematically checks whether this property holds for (a given initial state in) that model."* The result of the checking process is either a confirmation of the correctness of the checked model, a counterexample in which it fails to obey its specification, or failure, i. e., a situation in which no conclusion could be reached.

Hence, in the context of Genetic Programming, a model checker can be utilized as a Boolean function $\varphi : \mathbb{X} \mapsto \mathbb{B}$ which maps the evolved programs to *correct* ($\equiv$ `true`) or

[76] http://en.wikipedia.org/wiki/Model_checking [accessed 2008-10-02]

incorrect (≡ `false`). As objective function, φ therefore is rather infeasible, since it would lead directly to the all-or-nothing problem discussed in Section 4.10.2.[77]

Still, model checkers can be an interesting way to define termination criteria for the evolution or to verify its results. This may require a reduction of the expressiveness of the GP approaches utilized in order to make them compliant with the input languages of the model checkers. Then again, there are very powerful model checkers such as SPIN[78] [955, 176, 256], which processes systems written in the Promela[79] (the Process Meta Language) with which asynchronous distributed algorithms can be specified [175]. If such a system was used, no reduction of the expressiveness of the program representation would be needed at all. Nevertheless, a formal transformation of the GP representation to these languages must be provided in any circumstance. Creating such a transformation is complicated and requires a formal proof of correctness – checking a model without having shown the correctness of the model representation first is, basically, nonsense.[80]

The idea of using model checkers like SPIN is very tempting. One important drawback of this method is the unforeseeable runtime of the checking process which spans from almost instantaneous return up to almost half an hour [2031]. In the same series of experiments ([2031]), the checking process also failed in a fraction of cases ($\approx 18\%$) depending on the problem to be verified. Especially the unpredictable runtime for general problems led us to the decision to not use SPIN in our own works yet, since in the worst case, a few thousand program verifications could be required per generation in the GP system. Still, it is an interesting idea to evolve programs in Promela language and we will reconsider it in our future work and evaluate the utility and applicability of SPIN for the said purposes in detail.

Functional Adequacy In the face of this situation where we cannot automatically determine whether an evolved algorithm is correct, overfitted, or oversimplified, a notation for which solutions are acceptable and which are not is required. One definition which fits perfectly in this context is the idea of *functional adequacy* provided by Camps et al. [327], Gleizes et al. [809]:

Definition 4.7 (Functional Adequacy). When a system has the "right" behavior – judged by an external observer knowing the environment – we say that it is functionally adequate [809].

In the context of Genetic Programming, the *external observer* is represented by the objective functions which evaluate the *behavior* of the programs in the simulation *environments*. According to Gleizes et al. [809], functional adequacy also subsumes non-functional criteria such as memory consumption or response time if they become crucial in a certain environment, i. e., influence the functionality. For optimizing such criteria, different additional approaches are provided in Section 4.10.3.

Against The Halting Problem

In order to circumvent the Halting Problem, the evolved programs can be executed in simulations which allow limiting their runtime [2254, 1027]. Programs which have not finished until the time limit has elapsed are terminated automatically. Especially in linear Genetic Programming approaches, it is easy to do so by simply defining an upper bound for the number of instructions being executed. For tree-based representations, this is slightly more complicated.

Teller [2011] suggests to apply time-limiting approaches too, but also the use of so-called *anytime algorithms*, i. e., algorithms that store their best guess of the result in a certain

[77] One approach to circumvent this problem would be to check for several properties separately.
[78] `http://en.wikipedia.org/wiki/SPIN_model_checker` [accessed 2008-10-02]
[79] `http://en.wikipedia.org/wiki/Promela` [accessed 2008-10-02]
[80] Thanks to Hendrik Skubch for discussing this issue with me.

memory cell and update it during their run. Anytime algorithms can be stopped at any time, since the result is always there, although it would have been refined in the future course of the algorithm.

Another way to deal with this problem is to prohibit the evolution of infinite loops or recursions from the start by restricting the structural elements in the programming language. If there are no loops, there surely cannot be infinite ones either. Imposing such limitations, however, also restricts the programs that can evolve: A representation which does not allow infinite loops cannot be Turing-complete either.

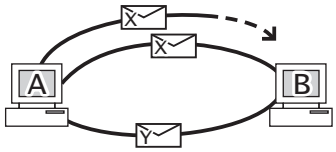


Figure 4.34: A sketch of an infinite message loop.

Often it is not sufficient to restrict just the programming language. An interesting example for this issue is the evolution of *distributed* algorithms. Here, the possible network situations and the reactions to them would also need to be limited. One would need to exclude situations like the one illustrated in Figure 4.34 where

1. node `A` sends message `X` to node `B` which
2. triggers an action there, leading to a response message `Y` from `B` back to node `A` which, in turn,
3. causes an action on `A` that includes sending `X` to `B` again
4. and so on...

Preventing such a situation is even more complicated and will, most likely, also prevent the evolution of useful solutions.

4.10.2 All-Or-Nothing?

The evolution of algorithms often proves as a special instance of the needle-in-a-haystack problem. From a naïve and, at the same time, mathematically precise point of view, an algorithm computing the greatest common divisor of two numbers, for instance, is either correct or wrong. Approaching this problem straightforwardly leads to the application of a single objective function which can take on only two values, provoking the *all-or-nothing* problem in Genetic Programming. In such a fitness landscape, a few steep spikes of equal height represent the correct algorithms and are distributed over a large plane of infeasible solution candidates with equally bad fitness.

The negative influence of all-or-nothing problems have been reported from many areas of Genetic Programming, such as the evolution of distributed protocols [2058] (see Section 23.2.2), quantum algorithms [1932], expression parsers [1027], and mathematical algorithms (such as the GCD).

In Section 21.3.2, we show how to some means to mitigate this problem for the GCD evolution. However, like those mentioned in some of the previously cited works, such methods are normally application dependent and often cannot be transferred to other problems in a simple manner.

Countermeasures

There are two direct countermeasures against the all-or-nothing problem in GP. The first one is to devise objective functions which can take on as many values as possible, i. e., which also reward partial solutions.

The second countermeasure is using as many test cases as possible and applying the objective functions to all of them, setting the final objective values to be the average of the results. Testing with ten training cases will transform a binary objective function to one which (theoretically) can take on eleven values, for instance: 1.0 if all training cases were processed correctly, 0.9 if one training case failed while nine worked out properly, ..., and 0.0 if the evolved algorithm was unable to behave adequately in any of the training cases. Using multiple training cases has, of course, the drawback that the time needed for the objective function evaluation will increase (linearly).

Vaguely related to these two measures is another approach, the utilization of Lamarckian evolution [522, 2215] or the Baldwin effect [123, 929, 930, 2215] (see Section 15.2 and Section 15.3, respectively). As already pointed out in Section 1.4.3, they incorporate a local search into the optimization process which may further help to smoothen out the fitness landscape [864].

In our experiments reported in [2177], an approach similar to Lamarckian evolution was incorporated. Although providing good results, the runtime of the approaches increased to a degree rendering it unfeasible for large-scale.[81]

4.10.3 Non-Functional Features of Algorithms

Besides evaluating an algorithm in terms of its functionality, there always exists a set of non-functional features that should be regarded too. For most non-functional aspects (such as code size, runtime requirements, and memory consumption) and the *parsimony*[82] principle holds: *less is better*. In this section, we will discuss various reasons for applying parsimony pressure in Genetic Programming.

Code Size

In Section 30.1.1 on page 547, we define what algorithms are: compositions of atomic instructions that, if executed, solve some kind of problem or a class of problems. Without specifying any closer what *atomic instructions* are, we can define the following:

Definition 4.8 (Code Size). The code size of an algorithm or program is the number of atomic instructions it is composed of.

The atomic instructions cannot be broken down into smaller pieces. Therefore, the code size is a positive integer number in $\mathbb{N}_0$. Since algorithms are statically finite per definition (see Definition 30.9 on page 550), the code size is always finite.

Code Bloat

Definition 4.9 (Bloat). In Genetic Programming, *bloat* is the uncontrolled growth in size of the individuals during the course of the evolution [1318, 229, 140, 1196, 1241].

The term code bloat is often used in conjunction with code *introns*, which are regions inside programs that do not contribute to the functional objective values (because they can never be reached, for instance; see Definition 3.2 on page 146). Limiting the code size and increasing the code efficiency by reducing the number of introns is an important task in Genetic Programming since disproportionate program growth has many bad side effects like:

1. The evolving programs become unnecessarily big while elegant solutions should always be as small and simple as possible.

[81] These issues were not the subject of the paper and thus, not discussed there.
[82] http://en.wikipedia.org/wiki/Parsimony [accessed 2008-10-14]

2. Mutation and recombination operators always have to select the point in an individual where they will apply their changes. If there are many points that do not contribute to functionality, the probability of selecting such a point for modification is high. The generated offspring will then have exactly the same functionality as its parents and the genetic operation performed was literarily useless.
3. Bloat slows down both, the evaluation [872] and the breeding process of new solution candidates.
4. Furthermore, it leads to increased memory consumption of the Genetic Programming system.

There are many theories about how code bloat emerges [1318], some of them are:

1. Unnecessary code hitchhikes with good individuals. If it is part of a fit solution candidate that creates many offspring, it is likely to be part of many new individuals. According to Tackett [1994], high selection pressure is thus likely to cause code growth. This idea is supported by the research of Langdon and Poli [1241], Smith and Harries [1906], and Gustafson et al. [872].
2. As already stated, unnecessary code makes it harder for genetic operations to alter the functionality of an individual. In most cases, genetic operators yield offspring with worse fitness than its parents. If a solution candidate has good objective values, unnecessary code can be one defense method against recombination and mutation. If the genetic operators are neutralized, the offspring will have the same fitness as its parent. This idea has been suggested in many sources, such as [229, 228, 1544, 1384, 1756, 140, 1244, 1906]. From this point of view, introns are a "bad" form of neutrality[83]. By the way, the reduction of the destructive effect of recombination on the fitness may also have positive effects, as pointed out by Nordin et al. [1546, 1547], since it may lead to a more durable evolution.
3. Luke [1318] defines a theory for tree growth based on the fact that recombination is likely to destroy the functionality of an individual. However, the deeper the crossover point is located in the tree, the smaller is its influence because fewer instructions are removed. If only a few instructions are replaced from a functionally adequate program, they are likely to be exchanged by a larger sub-tree. A new offspring that retains the functionality of its parents therefore tends to be larger.
4. Similar to the last two theories, the idea of removal bias by Soule and Foster [1922] states that removing code from an individual will preserve the individual's functionality if the code removed is non-functional. Since the portion of useless code inside a program is finite, there also exists an upper limit of the amount of code that can be removed without altering the functionality of the program. For the size of new sub-trees that could be inserted instead (due to mutation or crossover), no such limit exists. Therefore, programs tend to grow [1922, 1244].
5. According to the diffusion theory of Langdon et al. [1244], the number of large programs in the problem space that are functionally adequate is higher than the number of small adequate programs. Thus, code bloat could correspond to the movement of the population into the direction of equilibrium [1318].
6. Another theory considers the invalidators that make code unreachable or dysfunctional. In the formula $4+0*(4-x)$ for example, the multiplication with 0 makes the whole part $(4-x)$ *inviable*. Luke [1318] argues that the influence of invalidators would be higher in large trees than in small trees. If programs grow while the fraction of invalidators remains constant and those inherited from the parents stay in place, their chance to occur proportionally closer to the root increases. Then, the amount of unnecessary instructions would increase too and naturally approach 100%.
7. Instead of being real solutions, programs that grow uncontrolled also tend to be some sort of decision tables. This phenomenon is called *overfitting* and has already discussed

[83] You can find the topic of neutrality discussed in Section 1.4.5 on page 64.

in Section 1.4.8 on page 72 and Section 23.1.3 on page 399 in detail. The problem is that overfitted programs tend to have marvelous good fitness for the training cases/sample data, but are normally useless for any other input.

8. Like Tackett [1994], Gustafson et al. [872] link code growth to high selection pressure but also to loss of diversity in general. In populations with less diversity, recombination will frequently be applied to very similar individuals, which often yields slightly larger offspring.

Some approaches for fighting bloat are discussed in Section 4.10.3.

Runtime and Memory Consumption

Another aspect subject to minimization is generally the runtime of the algorithms grown. The amount of steps needed to solve a given task, i. e., the time complexity, is only loosely related to the code size. Although large programs with many instructions tend to run longer than small programs with few instructions, the existence of loops and recursion invalidates a direct relation.

Like the complexity in time, the complexity in memory space of the evolved solutions often is minimized, too. The number of variables and memory cells needed by program in order to perform its work should be as small as possible. Section 30.1.3 on page 550 provides some additional definitions and discussion about the complexity of algorithms.

Errors

An example for an application where the non-functional errors that can occur should be minimized is symbolic regression. Therefore, the property of *closure* specified in Definition 4.1 on page 178 is usually ensured. Then, the division operator `div` is re-defined in order to prevent division-by-zero errors. Therefore, such a division could either be rendered to a `nop` (i. e., does nothing) or yields 1 or the dividend as result. However, the number of such arithmetical errors could also be counted and made the subject to minimization too.

Transmission Count

If evolving distributed algorithms, the number of messages required to solve a problem should be as low as possible since transmissions are especially costly and time-consuming operations.

Optimizing Non-Functional Aspects

Optimizing the non-functional aspects of the individuals evolved is a topic of scientific interest.

1. One of the simplest means of doing so is to define additional objective functions which minimize the program size and to perform a multi-objective optimization. Successful and promising experiments by Bleuler et al. [227], de Jong et al. [510], and Ekárt and Németh [626] showed that this is a viable countermeasure for code bloat, for instance.
2. Another method is limiting the aspect of choice. A very simple measure to limit code bloat, for example, is to prohibit the evolution of trees with a depth surpassing a certain limit [1320].
3. Poli [1660] furthermore suggests that the fitness of a certain portion of the population with above-average code size should simply be set to the worst possible value. These artificial *fitness holes* will repel the individuals from becoming too large and hence, reduce the code bloat.

5

Evolution Strategy

5.1 Introduction

Evolution Strategies[1] (ES) introduced by Rechenberg [1712, 1713, 1714] are a heuristic optimization technique based in the ideas of adaptation and evolution, a special form of evolutionary algorithms [1712, 1713, 1714, 103, 200, 1841, 198, 916]. Evolution Strategies have the following features:

1. They usually use vectors of real numbers as solution candidates, i. e., $\mathbb{G} = \mathbb{X} = \mathbb{R}^n$. In other words, both the search and the problem space are fixed-length strings of floating point numbers, similar to the real-encoded genetic algorithms mentioned in Section 3.3 on page 145.
2. Mutation and selection are the primary operators and recombination is less common.
3. Mutation most often changes the elements $\mathbf{x}_{[i]}$ of the solution candidate vector $\mathbf{x}$ to a number drawn from a normal distribution $N\left(\mathbf{x}_{[i]}, \sigma_i^2\right)$. For reference, you can check Equation 11.1 on page 259 in the text about Random Optimization.
4. Then, the values σ_i are governed by self-adaptation [891, 1400, 1214] such as covariance matrix adaptation [888, 889, 890, 1041].
5. In all other aspects, they perform exactly like basic evolutionary algorithms as defined in Algorithm 2.1 on page 99.

5.2 General Information

5.2.1 Areas Of Application

Some example areas of application of Evolution Strategy are:

Application	References
Data Mining and Data Analysisanalysis	[445]
Scheduling	[971]
Chemistry, Chemical Engineering	[1755, 470, 632]
Ressource Minimization, Environment Surveillance/Protection	[1556]
Combinatorial Optimization	[1536, 193, 197]
Geometry and Physics	[1122, 2173]
Optics and Image Processing	[859, 860, 101, 2218, 2217, 1279]

[1] http://en.wikipedia.org/wiki/Evolution_strategy [accessed 2007-07-03], http://www.scholarpedia.org/article/Evolution_Strategies [accessed 2007-07-03]

5.2.2 Conferences, Workshops, etc.

Some conferences, workshops and such and such on Evolution Strategy are:

EUROGEN: Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems see Section 2.2.2 on page 106

5.2.3 Books

Some books about (or including significant information about) Evolution Strategy are:

Schwefel [1841]: *Evolution and Optimum Seeking: The Sixth Generation*
Rechenberg [1713]: *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*
Rechenberg [1714]: *Evolutionsstrategie '94*
Beyer [198]: *The theory of evolution strategies*
Schwefel [1840]: *Numerical Optimization of Computer Models*
Schöneburg, Heinzmann, and Feddersen [1831]: *Genetische Algorithmen und Evolutionsstrategien*
Bäck [99]: *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*

5.3 Populations in Evolution Strategy

Evolution Strategies usually combine truncation selection (as introduced in Section 2.4.2 on page 122) with one of the following population strategies. These strategies listed below have partly been borrowed from German Wikipedia [2219] site for Evolution Strategy[2].

5.3.1 $(1 + 1)$-ES

The population only consists of a single individual which is reproduced. From the elder and the offspring, the better individual will survive and form the next population. This scheme is very close to hill climbing which will be introduced in Chapter 10 on page 253.

5.3.2 $(\mu + 1)$-ES

Here, the population contains μ individuals from which one is drawn randomly. This individual is reproduced from the joint set of its offspring and the current population, the least fit individual is removed.

5.3.3 $(\mu + \lambda)$-ES

Using the reproduction operations, from μ parent individuals $\lambda \geq \mu$ offspring are created. From the joint set of offspring and parents, only the μ fittest ones are kept [936].

[2] http://de.wikipedia.org/wiki/Evolutionsstrategie [accessed 2007-07-03]

5.3.4 (μ, λ)-ES

In (μ, λ) Evolution Strategies, introduced by Schwefel [1840], again $\lambda \geq \mu$ children are created from μ parents. The parents are subsequently deleted and from the λ offspring individuals, only the μ fittest are retained [1840, 196].

5.3.5 $(\mu/\rho, \lambda)$-ES

Evolution Strategies named $(\mu/\rho, \lambda)$ are basically (μ, λ) strategies. The additional parameter ρ is added, denoting the number of parent individuals of one offspring. As already said, normally, we only use mutation ($\rho = 1$). If recombination is also used as in other evolutionary algorithms, $\rho = 2$ holds. A special case of $(\mu/\rho, \lambda)$ algorithms is the $(\mu/\mu, \lambda)$ Evolution Strategy [1369].

5.3.6 $(\mu/\rho + \lambda)$-ES

Analogously to $(\mu/\rho, \lambda)$-Evolution Strategies, the $(\mu/\rho + \lambda)$-Evolution Strategies are (μ, λ) approaches where ρ denotes the number of parents of an offspring individual.

5.3.7 $(\mu', \lambda'(\mu, \lambda)^\gamma)$-ES

Geyer et al. [791, 792, 793] have developed nested Evolution Strategies where λ' offspring are created and isolated for γ generations from a population of the size μ'. In each of the γ generations, λ children are created from which the fittest μ are passed on to the next generation. After the γ generations, the best individuals from each of the γ isolated solution candidates propagated back to the top-level population, i. e., selected. Then, the cycle starts again with λ' new child individuals. This nested Evolution Strategy can be more efficient than the other approaches when applied to complex multimodal fitness environments [1714, 793].

5.4 One-Fifth Rule

The $\frac{1}{5}$ success rule defined by Rechenberg [1713] states that the quotient of the number of successful mutations (i. e., those which lead to fitness improvements) to the total number of mutations should be approximately $\frac{1}{5}$. If the quotient is bigger, the σ-values should be increased and with that, the scatter of the mutation. If it is lower, σ should be decreased and thus, the mutations are narrowed down.

5.5 Differential Evolution

5.5.1 Introduction

Differential Evolution[3] (DE, DES) is a method for mathematical optimization of multidimensional functions that belongs to the group of evolution strategies [1676, 653, 1404, 288, 1234, 1391, 189]. Developed by Storn and Price [1974], the DE technique has been invented in order to solve the Chebyshev polynomial fitting problem. It has proven to be a very reliable optimization strategy for many different tasks where parameters that can be encoded in real vectors.

The essential idea behind Differential Evolution is the way the (ternary) recombination operator "deRecombination" is defined for creating new solution candidates. The difference

[3] http://en.wikipedia.org/wiki/Differential_evolution [accessed 2007-07-03], http://www.icsi.berkeley.edu/~storn/code.html [accessed 2007-07-03]

$\mathbf{x}_1 - \mathbf{x}_2$ of two vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ in $\mathbb{X}$ is weighted with a weight $w \in \mathbb{R}$ and added to a third vector $\mathbf{x}_3$ in the population.

$$\mathbf{x} = \text{deRecombination}(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) \Rightarrow \mathbf{x} = \mathbf{x}_3 + w\,(\mathbf{x}_1 - \mathbf{x}_2) \tag{5.1}$$

Except for determining w, no additional probability distribution has to be used and the Differential Evolution scheme is completely self-organizing. This classical reproduction strategy has been complemented with new ideas like triangle mutation and alternations with weighted directed strategies.

Gao and Wang [770] emphasize the close similarities between the reproduction operators of Differential Evolution and the search step of the downhill simplex. Thus, it is only logical to combine or to compare the two methods (see Section 16.4 on page 286). Further improvements to the basic Differential Evolution scheme have been contributed, for instance, by Kaelo and Ali. Their DERL and DELB algorithms outperformed [1078, 1079, 1077] standard DE on the test benchmark from Ali et al. [38].

5.5.2 General Information

Areas Of Application

Some example areas of application of Differential Evolution are:

Application	References
Engineering, Structural Optimization, and Design	[1233, 1506]
Chemistry, Chemical Engineering	[2148, 1846, 2052, 399]
Scheduling	[1289]
Function Optimization	[1972]
Electrical Engineering and Circuit Design	[1971, 1973]

Journals

Some journals that deal (at least partially) with Differential Evolution are:

Journal of Heuristics (see Section 1.6.3 on page 91)

Books

Some books about (or including significant information about) Differential Evolution are:

Price, Storn, and Lampinen [1676]: *Differential Evolution – A Practical Approach to Global Optimization*
Feoktistov [653]: *Differential Evolution – In Search of Solutions*
Corne, Dorigo, Glover, Dasgupta, Moscato, Poli, and Price [448]: *New Ideas in Optimisation*

6

Evolutionary Programming

6.1 Introduction

Different from the other major types of evolutionary algorithms introduced, there exists no clear specification or algorithmic variant for evolutionary programming[1] (EP) to the knowledge of the author. There is though a semantic difference: while single individuals of a species are the biological metaphor for solution candidates in other evolutionary algorithms, in evolutionary programming, a solution candidate is thought of as a species itself.[2] Hence, mutation and selection are the only operators used in EP and recombination is usually not applied. The selection scheme utilized in evolutionary programming is normally quite similar to the $(\mu + \lambda)$ method in Evolution Strategies.

Evolutionary programming was pioneered by Fogel [705] in his PhD thesis back in 1964. Fogel et al. [708] experimented with the evolution of finite state machines as predictors for data streams [623]. Evolutionary programming is also the research area of his son David Fogel [697, 699, 700] with whom he also published joint work [707, 1671].

Generally, it is hard to distinguish evolutionary programming from Genetic Programming, genetic algorithms, and Evolution Strategy. Although there are semantic differences (as already mentioned), the author thinks that the many aspects of the evolutionary programming approach have merged into these other research areas.

6.2 General Information

6.2.1 Areas Of Application

Some example areas of application of evolutionary programming are:

Application	References
Machine Learning	[697]
Cellular Automata and Finite State Machines	[708]
Evolving Behaviors, e.g., for Agents or Game Players	[699, 700]
Machine Learning	[1671]
Chemistry, Chemical Engineering and Biochemistry	[779, 609, 778]
Electrical Engineering and Circuit Design	[1135, 1518]
Data Mining and Data Analysis	[1802]
Robotics	[1136]

[1] http://en.wikipedia.org/wiki/Evolutionary_programming [accessed 2007-07-03]

[2] In this aspect it is very similar to the much newer Extremal Optimization approach which will be discussed in Chapter 13.

6.2.2 Conferences, Workshops, etc.

Some conferences, workshops and such and such on evolutionary programming are:

EP: International Conference on Evolutionary Programming
now part of *CEC*, see Section 2.2.2 on page 105
History: 1998: San Diego, California, USA, see [1670]
1997: Indianapolis, Indiana, USA, see [68]
1996: San Diego, California, USA, see [709]
1995: San Diego, California, USA, see [1380]
1994: see [1849]
1993: see [702]
1992: see [701]

EUROGEN: Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems
see Section 2.2.2 on page 106

6.2.3 Books

Some books about (or including significant information about) evolutionary programming are:

Fogel, Owens, and Walsh [708]: *Artificial Intelligence through Simulated Evolution*

Fogel [706]: *Intelligence Through Simulated Evolution: Forty Years of Evolutionary Programming*

Fogel [697]: *System Identification through Simulated Evolution: A Machine Learning Approach to Modeling*

Fogel [700]: *Blondie24: playing at the edge of AI*

Bäck [99]: *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*

7

Learning Classifier Systems

7.1 Introduction

In the late 1970s, Holland, the father of genetic algorithms, also invented the concept of classifier systems (CS) [948, 941, 946]. These systems are a special case of production systems [497, 498] and consist of four major parts:

1. a set of interacting production rules, called *classifiers*,
2. a performance algorithm which directs the actions of the system in the environment,
3. a learning algorithm which keeps track on the success of each classifier and distributes rewards, and
4. a genetic algorithm which modifies the set of classifiers so that variants of good classifiers persist and new, potentially better ones are created in an efficient manner [947].

By time, classifier systems have undergone some name changes. In 1986, reinforcement learning was added to the approach and the name changed to Learning Classifier Systems[1] (LCS) [916, 1909]. Learning Classifier Systems are sometimes subsumed under a machine learning paradigm called evolutionary reinforcement learning (ERL) [916] or Evolutionary Algorithms for Reinforcement Learning (EARLs) [1460].

7.2 General Information

7.2.1 Areas Of Application

Some example areas of application of Learning Classifier Systems are:

Application	References
Data Mining and Data Analysisg	[768, 92, 479, 444, 2178]
Grammar Induction	[2073, 2074, 472]
Medicine	[951]
Image Processing	[1287, 1376]
Sequence Prediction	[1736]

7.2.2 Conferences, Workshops, etc.

Some conferences, workshops and such and such on Learning Classifier Systems are:

[1] http://en.wikipedia.org/wiki/Learning_classifier_system [accessed 2007-07-03]

IWLCS: International Workshop on Learning Classifier Systems
Nowadays often co-located with GECCO (see Section 2.2.2 on page 107).
History: 2007: London, England, see [1946]
2006: Seattle, WA, USA, see [1847]
2005: Washington DC, USA, see [2157, 1181]
2004: Seattle, Washington, USA, see [1848, 1181]
2003: Chicago, IL, USA, see [2022, 1181]
2002: Granada, Spain, see [1254]
2001: San Francisco, CA, USA, see [1944]
2000: Paris, France, see [1253]
1999: Orlando, Florida, USA, see [1585]
1992: Houston, Texas, USA, see [1501]

7.2.3 Books

Some books about (or including significant information about) Learning Classifier Systems are:

Bull [301]: *Applications Of Learning Classifier Systems*
Bull and Kovacs [303]: *Foundations of Learning Classifier Systems*
Butz [314]: *Anticipatory Learning Classifier Systems*
Butz [315]: *Rule-Based Evolutionary Online Learning Systems: A Principled Approach to LCS Analysis and Design*
Lanzi, Stolzmann, and Wilson [1252]: *Learning Classifier Systems, From Foundations to Applications*

7.3 The Basic Idea of Learning Classifier Systems

Figure 7.1 illustrates the structure of a Michigan-style Learning Classifier System. A classifier system is connected via detectors *(b)* and effectors *(c)* to its environment *(a)*. The input in the system (coming from the detectors) is encoded in form of binary messages that are written into a message list *(d)*. On this list, simple `if-then` rules *(e)*, the so-called classifiers, are applied. The result of a classification is again encoded as a message and written to the message list. These new messages may now trigger other rules or are signals for the effectors [507]. The payoff of the performed actions is distributed by the credit apportionment system *(f)* to the rules. Additionally, a rule discovery system *(g)* is responsible for finding new rules and adding them to the classifier population [794].

Classifier systems are special instances of production systems, which were shown to be Turing-complete by Post [1672] and Minsky [1427, 1426]. Thus, Learning Classifier Systems are as powerful as any other Turing-equivalent programming language and can be pictured as something like computer programs where the rules play the role of the instructions and the messages are the memory.

7.3.1 A Small Example

In order to describe how rules and messages are structured in a basic classifier systems, we borrow a simple example from Heitkötter and Beasley [916]. We will orient our explanation at the syntax described by Geyer-Schulz [794]. You should, however, be aware that there are many different forms of classifier system and take this as an example for *how it could be done* rather than as *the way it is to be done*.

So let us imagine that we want to find a classifier system that is able to control the behavior of a frog. Our frog likes to eat nutritious flies. Therefore, it can detect small,

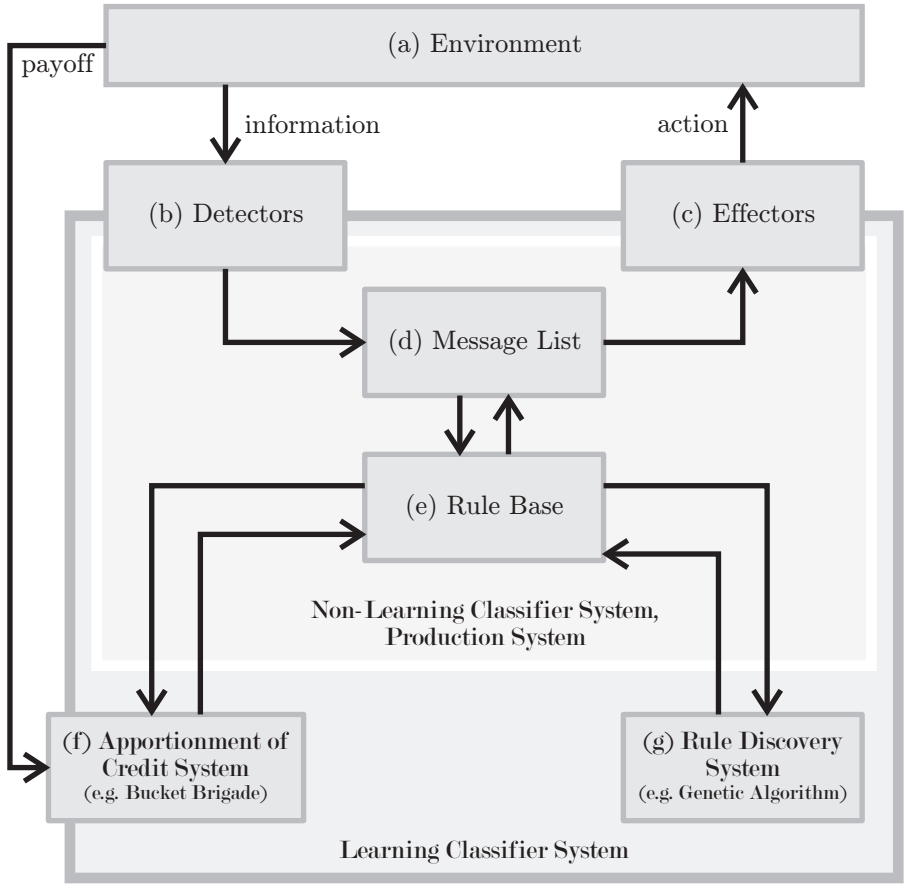


Figure 7.1: The structure of a Michigan style Learning Classifier System according to Geyer-Schulz [794].

flying objects and eat them if they are right in front of it. The frog also has a sense of direction and can distinguish between objects which are in front, to the left, or to the right of it and may also turn into any of these directions. It can furthermore distinguish objects with stripes from those without. Flying objects with stripes are most likely bees or wasps, eating of which would probably result in being stung. The frog can also sense large, looming objects far above: birds, which should be avoided by jumping away quickly. We can compile a corresponding behavior into the form of simple `if-then` rules which are listed in Table 7.1.

No.	premise (`if`-part)	conclusion (`then`-part)
1	small, flying object with no stripes to the left	send a
2	small, flying object with no stripes to the right	send b
3	small, flying object with no stripes to the front	send c
4	large, looming object	send d
5	a and not d	turn left
6	b and not d	turn right
7	c and not d	eat
8	d	move away rapidly

Table 7.1: `if-then` rules for frogs

7.3.2 Messages

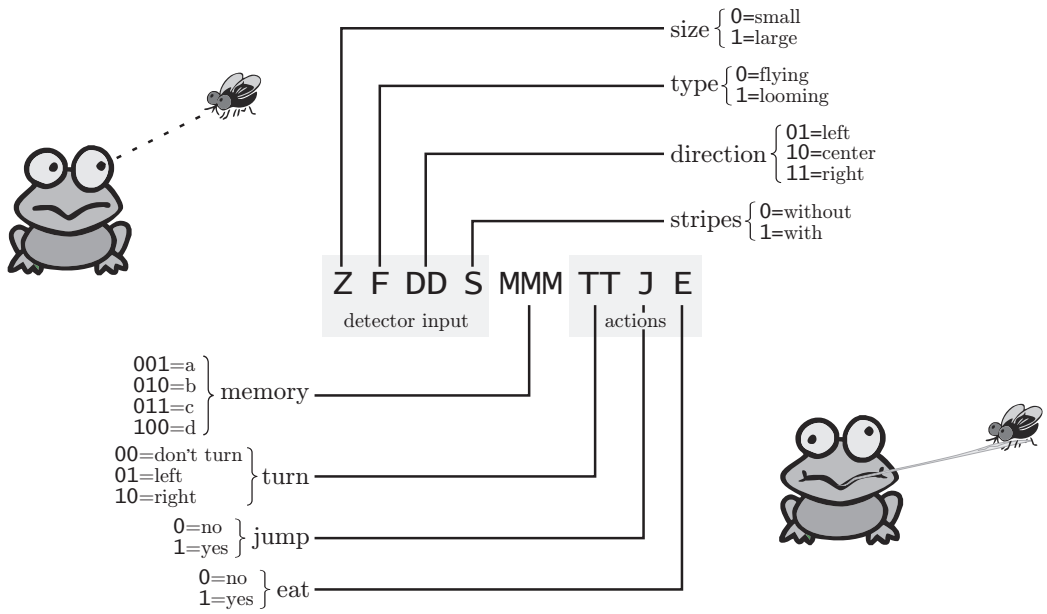


Figure 7.2: One possible encoding of messages for a frog classifier system

In Figure 7.2, we demonstrate how the messages in a classifier system that drives such a frog can be encoded. Here, input information as well as action commands (the conclusions of the rules) are compiled in one message type. Also, three bits are assigned for encoding the internal messages a to d. Two bits would not suffice, since `00` occurs in all "original" input messages. At the beginning of a classification process, the input messages are written to the message list. They contain information only at the positions reserved for detections and have zeros in the bits for memory or actions. The classifiers transform them to internal messages which normally have only the bits marked as "memory" set. These messages are finally transformed to output messages by setting some action bits. In our frog system, a message is in total $k = 12$ bits long, i. e., $\text{len}(m) = 12\ \forall \textit{message}\ m$.

7.3.3 Conditions

Rules in classifier systems consist of a condition part and an action part. The conditions have the same length k as the messages. Instead of being binary encoded strings, a ternary system consisting of the symbols `0`, `1`, and `*` is used. In a condition,

1. `0` means that the corresponding bit in the message must be `0`,
2. `1` means that the corresponding bit in the message must be `1`, and
3. `*` means *don't care*, i. e., the corresponding bit in the message may be `0` as well as `1` for the condition to match.

Definition 7.1 (match**).** A message m matches to a condition c if $\text{match}(m, c)$ evaluates to `true`.

$$\text{match}(m, c) = \ \forall 0 \leq i < |m| \Rightarrow m_{[i]} = c_{[i]} \vee c_{[i]} = * \tag{7.1}$$

The conditional part of a rule may consist of multiple conditions which are implicitly concatenated with logical and ($\wedge$). A classifier is satisfied if all its conditions are satisfied

by at least one message in the current message list. It is allowed that each of the conditions of a classifier may match to different messages.

We can precede each single condition c with an additional ternary digit which defines if it should be negated or not: `*` stands for the negation $\overline{c}$ and `0` as well as `1` denotes c. Here we deviate from the syntax described in Geyer-Schulz [794] because the definition of the "conditionSpecifity" (see Definition 7.2) becomes more beautiful this way. A negated condition evaluates to `true` if no message exists that matches it. By combining `and` and `not`, we get `nand`s with which we can build all other logic operations and, hence, whole computers [2045]. Algorithm 7.1 illustrates how the condition part C is matched against the message list M. If the matching is successful, it returns the list S of messages that satisfied the conditions. Otherwise, the output will be the empty list ().

Algorithm 7.1: $S \longleftarrow \text{matchesConditions}(M, C)$

Input: M: the message list
Input: C: the condition part of a classifier
Input: [implicit] k: the length of the messages $m \in M$ and the single conditions $c \in C$
Input: [implicit] $havePrefix$: `true` if and only if the single conditions have a prefix which determines whether or not they are negated, `false` if no such prefixes are used
Data: i: a counter variable
Data: c: a condition
Data: neg: should the condition be negated?
Data: m: a single message from M
Data: b: a Boolean variable
Output: S: the messages that match the condition part C, or () if none such message exists

```
1  begin
2      S ⟵ ()
3      b ⟵ true
4      i ⟵ 0
5      while (i < len(C)) ∧ b do
6          if havePrefix then
7              neg ⟵ (C[i] = ∗)
8              i ⟵ i + 1
9          else neg ⟵ false
10         c ⟵ subList(C, i, k)
11         i ⟵ i + k
12         if ∃m ∈ M : match(m, c) then
13             b ⟵ ¬neg
14             if b then S ⟵ addListItem(S, m)
15         else
16             b ⟵ neg
17             if b then S ⟵ addListItem(S, createList(k, 0))
18     if b then return S
19     else return ()
20 end
```

Definition 7.2 (Condition Specificity). The condition specificity conditionSpecifity(x) of a classifier x is the number of non-`*` symbols in its condition part $C(x)$.

$$\text{conditionSpecifity}(x) = |\{\forall i : C(x)_{[i]} \neq *\}| \tag{7.2}$$

A classifier (rule) x_1 with a higher condition specificity is more specific than another rule x_2 with a lower condition specificity. On the other hand, a rule x_2 with

conditionSpecifity(x_1) > conditionSpecifity(x_2) is more general than the rule x_1. We can use this information if two rules match to one message, and only one should be allowed to post a message. Preferring the more specific rule in such situations leads to *default hierarchies* [949, 1737, 1739, 1908] which allows general classifications to "delegate" special cases to specialized classifiers. Even more specialized classifiers can then represent exceptions to these refined rules.

7.3.4 Actions

The action part of a rule has normally exactly the same length as a message. It can be represented by a string of either binary or ternary symbols. In the first case, the action part of a rule is simple copied to the message list if the classifier is satisfied. In the latter case, some sort of merging needs to be performed. Here,

1. a `0` in the action part will lead to a `0` in the corresponding message bit,
2. a `1` in the action part will lead to a `1` in the corresponding message bit,
3. and for a `*` in the action part, we copy the corresponding bit from the (first) message that matched the classifier's condition to the newly created message.

Definition 7.3 (*mergeAction*). The function "mergeAction" computes a new message n as product of an action a. If the alphabet the action is based on is ternary and may contain $*$-symbols, mergeAction needs access to the message m which has satisfied the first condition of the classifier to which a belongs. If the classifier contains negation symbols and the first condition was negated, m is assumed to be a string of zeros ($m = \text{createList}(\text{len}(a), 0)$). Notice that we do not explicitly distinguish between binary and ternary encoding in mergeAction, since $*$ cannot occur in actions based on a binary alphabet and Equation 7.3 stays valid.

$$\begin{aligned} n = \text{mergeAction}(a, m) \Leftrightarrow\ & (\text{len}(n) = \text{len}(a)) \wedge \\ & (n_{[i]} = a_{[i]}\ \forall i \in 0..\text{len}(a) - 1 : a_{[i]} \neq *) \wedge \\ & (n_{[i]} = m_{[i]}\ \forall i \in 0..\text{len}(a) - 1 : a_{[i]} = *) \end{aligned} \tag{7.3}$$

7.3.5 Classifiers

So we know that a rule x consists of a condition part $C(x)$ and an action part $a(x)$. C is a list of $r \in \mathbb{N}$ conditions c_i, and we distinguish between representations with ($C = (n_1, c_1, n_2, c_2, \ldots, n_r, c_r)$) and without negation symbol ($C = (c_1, c_2, \ldots, c_r)$). Let us now go back to our frog example. Based on the encoding scheme defined in Figure 7.2, we can translate Table 7.1 into a set of classifiers. We therefore compose the condition parts of two conditions c_1 and c_2 with the negation symbols n_1 and n_2, i. e., $r = 2$. Table 7.2 contains

No.	n_1	c_1	n_2	c_2	a
1	0	0 0 01 0 *** ** * *	0	* * ** * *** ** * *	0 0 00 0 001 00 0 0
2	0	0 0 11 0 *** ** * *	0	* * ** * *** ** * *	0 0 00 0 010 00 0 0
3	0	0 0 10 0 *** ** * *	0	* * ** * *** ** * *	0 0 00 0 011 00 0 0
4	0	1 1 ** * *** ** * *	0	* * ** * *** ** * *	0 0 00 0 100 00 0 0
5	0	* * ** * 001 ** * *	*	* * ** * 100 ** * *	0 0 00 0 000 01 0 0
6	0	* * ** * 010 ** * *	*	* * ** * 100 ** * *	0 0 00 0 000 10 0 0
7	0	* * ** * 011 ** * *	*	* * ** * 100 ** * *	0 0 00 0 000 00 0 1
8	0	* * ** * 100 ** * *	0	* * ** * *** ** * *	0 0 00 0 000 00 1 0

Table 7.2: The encoded form of the `if-then` rules for frogs from Table 7.1.

the result of this encoding. We can apply this classifier to a situation in the life of our frog where it detects

1. a fly to its left,
2. a bee to its right, and
3. a stork left in the air.

How will it react? The input sensors will generate three messages and insert them into the message list $M_1 = (m_1, m_2, m_3)$:

1. $m_1 =$ (000100000000) for the fly,
2. $m_2 =$ (001110000000) for the bee, and
3. $m_3 =$ (110100000000) for the stork.

The first message triggers rule 1 and the third message triggers rule 4 whereas no condition fits to the second message. As a result, the new message list M_2 contains two messages, m_4 and m_5, produced by the corresponding actions.

1. $m_4 =$ (000000010000) from rule 1 and
2. $m_5 =$ (000001000000) from rule 4.

m_4 could trigger rule 5 but is inhibited by the negated second condition c_2 because of message m_5. m_5 matches to classifier 8 which finally produces message $m_6 =$ (000000000010) which forces the frog to jump away. No further classifiers become satisfied with the new message list $M_3 = (m_6)$ and the classification process is terminated.

7.3.6 Non-Learning Classifier Systems

So far, we have described a non-learning classifier system. Algorithm 7.2 defines the behavior of such a system which we also could observe in the example. It still lacks the credit apportionment and the rule discovery systems (see *(f)* and *(g)* in Figure 7.1). A non-learning classifier is able to operate correctly on a fixed set of situations. It is sufficient for all applications where we are able to determine this set beforehand and no further adaptation is required. If this is the case, we can use genetic algorithms to evolve the classifier systems offline, for instance.

Algorithm 7.2 illustrates how a classifier system works. No optimization or approximation of a solution is done; this is a complete control system in action. Therefore we do not need a termination criterion but run an infinite loop.

7.3.7 Learning Classifier Systems

In order to convert this non-learning classifier system to Learning Classifier System as proposed by Holland [943] and sketched in Algorithm 7.3, we have to add the aforementioned missing components. Heitkötter and Beasley [916] suggest two ways for doing so:

1. Currently, the activation of a classifier x results solely from the message-matching process. If a message matches the condition(s) $C(x)$, the classifier may perform its action $a(x)$. We can change this mechanism by making it also dependent on an additional parameter $v(x)$ – a strength value, which can be modified as a result of experience, i. e., by reinforcement from the environment. Therefore, we have to solve the *credit assignment problem* first defined by Minsky [1425, 1428], since chains of multiple classifiers can cause a certain action.
2. Furthermore (or instead), we may also modify the set of classifiers P by adding, removing, or combining condition/action parts of existing classifiers.

A Learning Classifier System hence is a control system which is able to learn while actually running and performing its work. Usually, a training phase will precede any actual deployment. Afterwards, the learning may even be deactivated, which turns the LCS into an ordinary classifier system or the learning rate is decreased.

Algorithm 7.2: nonLearningClassifierSystem(P)

Input: P: the list of rules x_i that determine the behavior of the classifier system
Input: [implicit] readDetectors: a function which creates a new message list containing only the input messages from the detectors
Input: [implicit] sendEffectors: a function which translates all messages concerning effectors to signals for the output interface
Input: [implicit] $t_{max} \in \mathbb{N}$: the maximum number of iterations for the internal loop, avoids endless loops
Data: t: a counter the internal loop
Data: M, N, S: the message lists
Data: x: a single classifier

```
1  begin
2      while true do
3          M ⟵ readDetectors()
4          t ⟵ 0
5          repeat
6              N ⟵ ()
7              foreach x ∈ P do
8                  S ⟵ matchesConditions(M, C(x))
9                  if len(S) > 0 then
10                     N ⟵ addListItem(N, mergeAction(a(x), S[0]))
11             M ⟵ N
12             t ⟵ t + 1
13         until (len(M) = 0) ∨ (t > t_max)
14         if len(M) > 0 then sendEffectors(M)
15 end
```

7.3.8 The Bucket Brigade Algorithm

The Bucket Brigade Algorithm has been developed by Holland [942, 943] as one method of solving the credit assignment problem in Learning Classifier Systems. Research work concerning this approach and its possible extensions has been conducted by Westerdale [2195, 2196, 2197], Antonisse [74], Huang [969], Riolo [1738, 1737], Dorigo [579], Spiessens [1942], Wilson [2234], Holland and Burks [946], and Hewahi and Bharadwaj [922] and has neatly been summarized by Hewahi [920, 921]. In the following, we will outline this approach with the notation of de Boer [507].

The Bucket Brigade Algorithm selects the classifiers from the match set X that are allowed to post a message (i. e., becoming member in the activated set U) by an auction. Therefore, each matching classifier x places a bid $B(x)$ which is the product of a linear function ϑ of the condition specificity of x, a constant $0 < \beta \leq 1$ that determines the fraction of the strength of $x$should be used and its strength $v(x)$ itself. In practical applications, values like $\frac{1}{8}$ or $\frac{1}{16}$ are often chosen for β.

$$B(x) = \vartheta(x) * \beta * v(x) + \text{random}_n\left(0, \sigma^2\right) \tag{7.4}$$

Sometimes, a normal distributed random number is added to each bid in order to make the decisions of the system less deterministic, as done in Equation 7.4.

The condition specificity is included in the bid calculation because it gives a higher value to rules with fewer $*$-symbols in their conditions. These rules match to fewer messages and can be considered more relevant in the cases they do match. For ϑ, the quotient of the number non-$*$-symbols and the condition length plus some constant $0 < \alpha$ determining the importance of the specificity of the classifier is often used [507].

$$\vartheta(x) = \frac{\text{conditionSpecifity}(x)}{\text{len}(C(x))} + \alpha \tag{7.5}$$

Algorithm 7.3: learningClassifierSystem()

Input: P: the list of rules x_i that determine the behavior of the classifier system
Input: [implicit] generateClassifiers: a function which creates randomly a population P of classifiers
Input: [implicit] readDetectors: a function which creates a new message list containing only the input messages from the detectors
Input: [implicit] sendEffectors: a function which translates all messages concerning effectors to signals for the output interface
Input: [implicit] selectMatchingClassifiers: a function that determines at most k classifiers from the matching set that are allowed to trigger their actions
Input: [implicit] generationCriterion: a criterion that becomes `true` if new classifiers should be created
Input: [implicit] updateRules: a function that finds new rules and deletes old ones
Input: [implicit] $t_{max} \in \mathbb{N}$: the maximum number of iterations for the internal loop, avoids endless loops
Data: t, i: counter variables
Data: M, N, S: the message lists
Data: X: a list of tuples containing classifiers and the (first) messages that satisfied their conditions
Data: v: the strength values
Data: x: a single classifier

```
1  begin
2      P ⟵ generateClassifiers(s)
3      foreach x ∈ P do v(x) ⟵ 1
4      while true do
5          M ⟵ readDetectors()
6          t ⟵ 0
7          repeat
8              X ⟵ ()
9              foreach x ∈ P do
10                 S ⟵ matchesConditions(M, C(x))
11                 if len(S) > 0 then
12                     X ⟵ addListItem(X, (x, S[0]))
13             N ⟵ ()
14             if len(X) > 0 then
15                 (X, v) ⟵ selectMatchingClassifiers(X, v)
16                 for i ⟵ 0 up to len(X) − 1 do
17                     x ⟵ X[i,0]
18                     N ⟵ addListItem(N, mergeAction(a(x), X[i,1]))
19             M ⟵ N
20             t ⟵ t + 1
21         until (len(M) = 0) ∨ (t > t_max)
22         if len(M) > 0 then
23             sendEffectors(M)
               // distribute Payoffs
24             if generationCriterion() then P ⟵ updateRules(P, v)
25 end
```

The bucket brigade version of the selectMatchingClassifiers-function introduced in Algorithm 7.3 then picks the k classifiers with the highest bids and allows them to write their messages into the new message list. They are charged with the payment part $P(x)$ of their bids. The payment does not contain the condition specificity-dependent part and also not the possible random addend. It is added as reward $R(y)$ to the strength of classifier y that wrote the message which allowed them to become active. In the case that this was an input message, it is simple thrown away. The payment of classifiers that are not activated is null.

$$P(x) = \beta * v(x) \tag{7.6}$$

In some Learning Classifier Systems, a life-tax $T(x)$ is collected from all classifiers in each cycle. It is computed as a small fraction τ of their strength.

$$T(x) = \tau * v(x) \tag{7.7}$$

Those classifiers that successfully triggered an action of the effectors receive a reward $R(x)$ from the environment which is added to their strength. Together with the payment method, all rules that are involved in a successful action receive some of the reward which is handed down stepwise – similar to how water is transported by a bucket brigade. For all classifiers that do not produce output to the effectors and also do not receive payment from other classifier they have triggered, this reward is null.

In total, the new strength $v_{t+1}(x)$ of a classifier x is composed of its old strength, its payment $P(x)$, the life-tax $T(x)$, and the reward $R(x)$.

$$v_{t+1}(x) = v_t(x) - P(x) - T(x) + R(x) \tag{7.8}$$

Instead of the Bucket Brigade Algorithm, it is also possible to use Q-Learning in Learning Classifier Systems, as shown by Wilson [2235]. Dorigo and Bersini [580] have shown that both concepts are roughly equivalent [916].

7.3.9 Applying the Genetic Algorithm

With the credit assignment alone, no new rules can be discovered – only the initial, randomly create rule set P is rated. At some certain points in time, a genetic algorithm (see Chapter 3 on page 141) replaces old rules by new ones. In Learning Classifier Systems we apply steady-state genetic algorithms which are discussed in Section 2.1.6 on page 102. They will retain most of the classifier population and only replace the weakest rules. Therefore, the strength $v(x)$ of a rule x is directly used as its fitness and is subject to maximization.

For mutation and crossover, the well known reproduction operations for fixed-length string chromosomes discussed in Section 3.4 on page 147 are employed.

7.4 Families of Learning Classifier Systems

The exact definition of Learning Classifier Systems [1180, 950, 1909, 1251] still seems contentious and there exist many different implementations. There are, for example, versions without message list where the action part of the rules does not encode messages but direct output signals. The importance of the role of genetic algorithms in conjunction with the reinforcement learning component is also not quite clear. There are scientists who emphasize more the role of the learning components [2239] and others who tend to grant the genetic algorithms a higher weight [466, 948]. The families of Learning Classifier Systems have been listed and discussed by Brownlee [296] elaborately. Here we will just summarize their differences in short. De Jong [514, 513] and Grefenstette [852] divide Learning Classifier Systems into two main types, depending on how the genetic algorithm acts: The *Pitt approach* originated at the University of Pittsburgh with the LS-1 system developed by Smith [1912].

It was then developed further and applied by Spears and De Jong [1926], De Jong and Spears [516], De Jong et al. [517], Bacardit i Peñarroya [92, 93], and Bacardit i Peñarroya and Krasnogor [94]. Pittsburgh-style Learning Classifier Systems work on a population of separate classifier systems, which are combined and reproduced by the genetic algorithm.

The original idea of Holland and Reitman [948] were *Michigan-style* LCSs, where the whole population itself is considered as classifier system. They focus on selecting the best rules in this rule set [820, 507, 1297].

Wilson [2235, 2236] developed two subtypes of Michigan-style LCS:

1. In ZCS systems, there is no message list use fitness sharing [2235, 418, 302, 300] for a Q-learning-like reinforcement learning approach called QBB.
2. ZCS have later been somewhat superseded by XCS systems in which the Bucket Brigade Algorithm has fully been replaced by Q-learning. Furthermore, the credit assignment is based on the accuracy (usefulness) of the classifiers. The genetic algorithm is applied to sub-populations containing only classifiers which apply to the same situations. [2236, 1179, 2237, 2238, 1256]

8

Ant Colony Optimization

8.1 Introduction

Inspired by the research done by Deneubourg et al. [554], [553, 839] on real ants and probably by the simulation experiments by Stickland et al. [1964], Dorigo et al. [584] developed the Ant Colony Optimization[1] (ACO) Algorithm for problems that can be reduced to finding optimal paths in graphs in 1996. [581, 585, 1352, 1355, 593] Ant Colony Optimization is based on the metaphor of ants seeking food. In order to do so, an ant will leave the anthill and begin to wander into a random direction. While the little insect paces around, it lays a trail of pheromone. Thus, after the ant has found some food, it can track its way back. By doing so, it distributes another layer of pheromone on the path. An ant that senses the pheromone will follow its trail with a certain probability. Each ant that finds the food will excrete some pheromone on the path. By time, the pheromone density of the path will increase and more and more ants will follow it to the food and back. The higher the pheromone density, the more likely will an ant stay on a trail. However, the pheromones vaporize after some time. If all the food is collected, they will no longer be renewed and the path will disappear after a while. Now, the ants will head to new, random locations.

This process of distributing and tracking pheromones is one form of stigmergy[2] and was first described by Grassé [849]. Today, we subsume many different ways of communication by modifying the environment under this term, which can be divided into two groups: sematectonic and sign-based [1833]. According to Wilson [2231], we call modifications in the environment due to a task-related action which leads other entities involved in this task to change their behavior sematectonic stigmergy. If an ant drops a ball of mud somewhere, this may cause other ants to place mud balls at the same location. Step by step, these effects can cumulatively lead to the growth of complex structures. Sematectonic stigmergy has been simulated on computer systems by, for instance, Théraulaz and Bonabeau [2032] and with robotic systems by Werfel and Nagpal [2192, 1837, 2193].

The second form, sign-based stigmergy, is not directly task-related. It has been attained evolutionary by social insects which use a wide range of pheromones and hormones for communication. Computer simulations for sign-based stigmergy were first performed by Stickland et al. [1964] in 1992.

The sign-based stigmergy is copied by Ant Colony Optimization [584], where optimization problems are visualized as (directed) graphs. First, a set of *ants* performs randomized walks through the graphs. Proportional to the goodness of the solutions denoted by the paths, pheromones are laid out, i. e., the probability to walk into the direction of the paths is shifted. The ants run again through the graph, following the previously distributed pheromone. However, they will not exactly follow these paths. Instead, they may deviate

[1] http://en.wikipedia.org/wiki/Ant_colony_optimization [accessed 2007-07-03]
[2] http://en.wikipedia.org/wiki/Stigmergy [accessed 2007-07-03]

from these routes by taking other turns at junctions, since their walk is still randomized. The pheromones modify the probability distributions.

It is interesting to note that even real vector optimizations can be mapped to a graph problem, as introduced by Korošec and Šilc [1176]. Thanks to such ideas, the applicability of Ant Colony Optimization is greatly increased.

8.2 General Information

8.2.1 Areas Of Application

Some example areas of application of Ant Colony Optimization are:

Application	References
Combinatorial Optimization	[763, 582, 869, 765, 764, 304, 305, 577]
Scheduling	[1392]
Networking and Communication	[1833, 1832, 2033, 1880, 725, 1281, 559, 561, 245] see Section 23.2 on page 401
Combinatorial Optimization	[1509]

8.2.2 Conferences, Workshops, etc.

Some conferences, workshops and such and such on Ant Colony Optimization are:

ANTS: International Conference on Ant Colony Optimization and Swarm Intelligence
http://iridia.ulb.ac.be/~ants/ [accessed 2008-08-20]
History: 2008: Brussels, Belgium, see [1947]
2006: Brussels, Belgium, see [594]
2004: Brussels, Belgium, see [592]
2002: Brussels, Belgium, see [590]
2000: Brussels, Belgium, see [589]
1998: Brussels, Belgium, see [588]

BIOMA: International Conference on Bioinspired Optimization Methods and their Applications
see Section 2.2.2 on page 105

CEC: Congress on Evolutionary Computation
see Section 2.2.2 on page 105

GECCO: Genetic and Evolutionary Computation Conference
see Section 2.2.2 on page 107

ICNC: International Conference on Advances in Natural Computation
see Section 1.6.2 on page 89

8.2.3 Journals

Some journals that deal (at least partially) with Ant Colony Optimization are:

Adaptive Behavior, ISSN: Online: 1741-2633, Print: 1059-7123, appears quaterly, editor(s): Peter M. Todd, publisher: Sage Publications, http://www.isab.org/journal/ [accessed 2007-09-16], http://adb.sagepub.com/ [accessed 2007-09-16]

Artificial Life, ISSN: 1064-5462, appears quaterly, editor(s): Mark A. Bedau, publisher: MIT Press, http://www.mitpressjournals.org/loi/artl [accessed 2007-09-16]

IEEE Transactions on Evolutionary Computation (see Section 2.2.3 on page 108)

The Journal of the Operational Research Society (see Section 1.6.3 on page 91)

8.2.4 Online Resources

Some general, online available ressources on Ant Colony Optimization are:

http://iridia.ulb.ac.be/~mdorigo/ACO/ [accessed 2007-09-13]

Last update:	up-to-date
Description:	Repository of books, publications, people, jobs, and software about ACO.

http://uk.geocities.com/markcsinclair/aco.html [accessed 2007-09-13]

Last update:	2006-11-17
Description:	Small intro to ACO, some references, and a nice applet demonstrating its application to the travelling salesman problem [1263, 78].

8.2.5 Books

Some books about (or including significant information about) Ant Colony Optimization are:

Chan and Tiwari [372]: *Swarm Intelligence – Focus on Ant and Particle Swarm Optimization*

Dorigo and Stützle [583]: *Ant Colony Optimization*

Engelbrecht [633]: *Fundamentals of Computational Swarm Intelligence*

Nedjah and de Macedo Mourelle [1509]: *Systems Engineering using Particle Swarm Optimisation*

Bonabeau, Dorigo, and Theraulaz [245]: *Swarm Intelligence: From Natural to Artificial Systems*

8.3 River Formation Dynamics

River Formation Dynamics (RFD) is a heuristic optimization method recently developed by Rabanal Basalo et al. [1689, 1690]. It is inspired by the way water forms rivers by eroding the ground and depositing sediments. In its structure, it is very close to Ant Colony Optimization. In Ant Colony Optimization, paths through a graph are searched by attaching attributes (the pheromones) to its edges. The pheromones are laid out by ants (+) and vaporize as time goes by (-). In River Formation Dynamics, the heights above sea level are the attributes of the vertexes of the graph. On this landscape, rain begins to fall. Forced by gravity, the drops flow downhill and try to reach the sea. The altitudes of the points in the graph are decreased by erosion (-) when water flows over them and increased by sedimentation (+) if drops end up in a dead end, vaporize, and leave the material which they have eroded somewhere else behind. Sedimentation punishes inefficient paths: If drops reaching a node surrounded only by nodes of higher altitudes will increase height more

and more until it reaches the level of its neighbors and is not a dead end anymore. While flowing over the map, the probability that a drop takes a certain edge depends on gradient of the down slope. This gradient, in turn, depends on the difference in altitude of the nodes it connects and their distance (i. e., the cost function). Initially, all nodes have the same altitude except for the destination node which is a hole. New drops are inserted in the origin node and flow over the landscape, reinforce promising paths, and either reach the destination or vaporize in dead ends.

Different from ACO, cycles cannot occur in RFD because the water always flows downhill. Of course, rivers in nature may fork and reunite, too. But, unlike ACO, River Formation Dynamics implicitly creates direction information in its resulting graphs. If this information is considered to be part of the solution, then cycles are impossible. If it is stripped away, cycles may occur.

9

Particle Swarm Optimization

9.1 Introduction

Particle Swarm Optimization[1] (PSO), developed by Eberhart and Kennedy [615, 1124] in 1995, is a form of swarm intelligence in which the behavior of a biological social system like a flock of birds or a school of fish [1616] is simulated. When a swarm looks for food, its individuals will spread in the environment and move around independently. Each individual has a degree of freedom or randomness in its movements which enables it to find food accumulations. So, sooner or later, one of them will find something digestible and, being social, announces this to its neighbors. These can then approach the source of food, too. Particle Swarm Optimization has been discussed, improved, and refined by many researchers such as Venter and Sobieszczanski-Sobieski [2113], Cai et al. [324], Gao and Duan [771], and Gao and Ren [772]. Comparisons with other evolutionary approaches have been provided by Eberhart and Shi [616] and Angeline [64].

With Particle Swarm Optimization, a swarm of particles (individuals) in a n-dimensional search space $\mathbb{G}$ is simulated, where each particle p has a position $p.g \in \mathbb{G} \subseteq \mathbb{R}^n$ and a velocity $p.\mathbf{v} \in \mathbb{R}^n$. The position $p.g$ corresponds to the genotypes, and, in most cases, also to the solution candidates, i. e., $p.x = p.g$, since most often the problem space $\mathbb{X}$ is also the $\mathbb{R}^n$ and $\mathbb{X} = \mathbb{G}$. However, this is not necessarily the case and generally, we can introduce any form of genotype-phenotype mapping in Particle Swarm Optimization. The velocity vector $p.\mathbf{v}$ of an individual p determines in which direction the search will continue and if it has an explorative (high velocity) or an exploitive (low velocity) character.

In the initialization phase of Particle Swarm Optimization, the positions and velocities of all individuals are randomly initialized. In each step, first the velocity of a particle is updated and then its position. Therefore, each particle p has a memory holding its best position $\text{best}(p) \in \mathbb{G}$. In order to realize the social component, the particle furthermore knows a set of topological neighbors $\text{N}(p)$. This set could be defined to contain adjacent particles within a specific perimeter, i. e., all individuals which are no further away from $p.g$ than a given distance δ according to a certain distance measure[2] dist. Using the Euclidian distance measure dist_{eucl} specified in Definition 29.8 on page 538 we get:

$$\forall\, p, q \in Pop : q \in \text{N}(p) \Leftrightarrow \text{dist}_{eucl}(p.g, q.g) \leq \delta \tag{9.1}$$

Each particle can communicate with its neighbors, so the best position found so far by any element in $\text{N}(p)$ is known to all of them as $\text{best}(\text{N}(p))$. The best position ever visited by any individual in the population (which the optimization algorithm always keeps track of) is $\text{best}(Pop)$.

The PSO algorithm may make use of either $\text{best}(\text{N}(p))$ or $\text{best}(Pop)$ for adjusting the velocity of the particle p. If it relies on the global best position, the algorithm will converge

[1] http://en.wikipedia.org/wiki/Particle_swarm_optimization [accessed 2007-07-03]

[2] See Section 29.1 on page 537 for more information on distance measures.

fast but may find the global optimum less probably. If, on the other hand, neighborhood communication is used, the convergence speed drops but the global optimum is found more likely.

Definition 9.1 (psoUpdate**).** The search operation $q = \text{psoUpdate}(p, Pop)$ applied in Particle Swarm Optimization creates a new particles q to replace an existing one (p) by incorporating its genotype $p.g$, its velocity $p.\mathbf{v}$. We distinguish local updating (Equation 9.3) and global updating (Equation 9.2), which additionally uses the data from the whole population Pop. psoUpdate thus fulfills one of these two equations and Equation 9.4, showing how the i^{th} components of the corresponding vectors are computed.

$$q.\mathbf{v}_i = p.\mathbf{v}_i + (\text{random}_u(0, \mathbf{c}_i) * (\text{best}(p).g_i - p.g_i)) + (\text{random}_u(0, \mathbf{d}_i) * (\text{best}(Pop).g_i - p.g_i)) \tag{9.2}$$

$$q.\mathbf{v}_i = p.\mathbf{v}_i + (\text{random}_u(0, \mathbf{c}_i) * (\text{best}(p).g_i - p.g_i)) + (\text{random}_u(0, \mathbf{d}_i) * (\text{best}(\text{N}(p)).g_i - p.g_i)) \tag{9.3}$$

$$q.g_i = p.g_i + p.\mathbf{v}_i \tag{9.4}$$

The learning rate vectors $\mathbf{c}$ and $\mathbf{d}$ have strong influence of the convergence speed of Particle Swarm Optimization. The search space $\mathbb{G}$ (and thus, also the values of $p.g$) is normally confined by minimum and maximum boundaries. For the absolute values of the velocity, normally maximum thresholds also exist. Thus, real implementations of "psoUpdate" have to check and refine their results before the utility of the solution candidates is evaluated.

Algorithm 9.1 illustrates the native form of the Particle Swarm Optimization using the update procedure from Definition 9.1. Like hill climbing, this algorithm can easily be generalized for multi-objective optimization and for returning sets of optimal solutions (compare with Section 10.3 on page 254).

Algorithm 9.1: $x^\star \longleftarrow \text{psoOptimizer}fps$

Input: f: the function to optimize
Input: ps: the population size
Data: Pop: the particle population
Data: i: a counter variable
Output: $x^\star$: the best value found

```
1 begin
2     Pop ⟵ createPop(ps)
3     while ¬terminationCriterion() do
4         for i ⟵ 0 up to len(Pop) − 1 do
5             Pop[i] ⟵ psoUpdate(Pop[i], Pop)
6     return best(Pop).x
7 end
```

9.2 General Information

9.2.1 Areas Of Application

Some example areas of application of particle swarm optimization are:

Application	References
Machine Learning	[1124, 1386, 1708]
Function Optimization	[1124, 1617]
Geometry and Physics	[2263]
Operations Research	[125]
Chemistry, Chemical Engineering	[356, 1864]
Electrical Engineering and Circuit Design	[1509]

9.2.2 Online Resources

Some general, online available ressources on particle swarm optimization are:

`http://www.swarmintelligence.org/` [accessed 2007-08-26]
Last update: up-to-date
Description: Particle Swarm Optimization Website by Xiaohui Hu

`http://www.red3d.com/cwr/boids/` [accessed 2007-08-26]
Last update: up-to-date
Description: Boids – Background and Update by Craig Reynolds

`http://www.projectcomputing.com/resources/psovis/` [accessed 2007-08-26]
Last update: 2004
Description: Particle Swarm Optimization (PSO) Visualisation (or "PSO Visualization")

`http://www.engr.iupui.edu/~eberhart/` [accessed 2007-08-26]
Last update: 2003
Description: Russ Eberhart's Home Page

`http://www.cis.syr.edu/~mohan/pso/` [accessed 2007-08-26]
Last update: 1999
Description: Particle Swarm Optimization Homepage

`http://tracer.uc3m.es/tws/pso/` [accessed 2007-11-06]
Last update: up-to-date
Description: Website on Particle Swarm Optimization

9.2.3 Conferences, Workshops, etc.

Some conferences, workshops and such and such on particle swarm optimization are:

GECCO: Genetic and Evolutionary Computation Conference
see Section 2.2.2 on page 107

ICNC: International Conference on Advances in Natural Computation
see Section 1.6.2 on page 89

SIS: IEEE Swarm Intelligence Symposium
`http://www.computelligence.org/sis/` [accessed 2007-08-26]
History: 2007: Honolulu, Hawaii, USA, see [1867]
2006: Indianapolis, IN, USA, see [1022]
2005: Pasadena, CA, USA, see [1021]
2003: Indianapolis, IN, USA, see [1020]

9.2.4 Books

Some books about (or including significant information about) particle swarm optimization are:

Nedjah and de Macedo Mourelle [1508]: *Swarm Intelligent Systems*
Chan and Tiwari [372]: *Swarm Intelligence – Focus on Ant and Particle Swarm Optimization*
Clerc [415]: *Particle Swarm Optimization*
Bui and Alam [299]: *Multi-Objective Optimization in Computational Intelligence: Theory and Practice*
Engelbrecht [633]: *Fundamentals of Computational Swarm Intelligence*
Kennedy, Eberhart, and Shi [1125]: *Swarm Intelligence: Collective, Adaptive*
Nedjah and de Macedo Mourelle [1509]: *Systems Engineering using Particle Swarm Optimisation*

10

Hill Climbing

10.1 Introduction

Hill climbing[1] (HC) [1780] is a very old and simple search and optimization algorithm for single objective functions f. In principle, hill climbing algorithms perform a loop in which the currently known best solution individual $p^\star$ is used to produce one offspring p_{new}. If this new individual is better than its parent, it replaces it. Then, the cycle starts all over again. In this sense, it is similar to an evolutionary algorithm with a population size $ps$of 1. Although the search space $\mathbb{G}$ and the problem space $\mathbb{X}$ are most often the same in hill climbing, we distinguish them in Algorithm 10.1 for the sake of generality. Hill climbing furthermore normally uses a parameterless search operation to create the first solution candidate and, from there on, unary operations to produce the offspring. Without loss of generality, we will thus make use of the reproduction operations from evolutionary algorithms defined in Section 2.5 on page 137, i. e., set $Op = \{\text{create}, \text{mutate}\}$.

The major problem of hill climbing is premature convergence, i.e., it gets easily stuck on a local optimum. It *always* uses the best known solution candidate $x^\star$ to find new points in the problem space $\mathbb{X}$. Hill climbing utilizes a unary reproduction operation similar to mutation in evolutionary algorithms. It should be noted that hill climbing can be implemented in a deterministic manner if the neighbor sets in search space $\mathbb{G}$, which here most often equals the problem space $\mathbb{X}$, are always finite and can be iterated over.

[1] http://en.wikipedia.org/wiki/Hill_climbing [accessed 2007-07-03]

Algorithm 10.1: $x^\star \longleftarrow$ hillClimber(f)

Input: f: the objective function subject to minization
Data: p_{new}: the new element created
Data: $p^\star$: the (currently) best individual
Output: $x^\star$: the best element found

```
1 begin
2     p⋆.g ⟵ create()
      // Implicitly: p⋆.x ⟵ gpm(p⋆.g)
3     while terminationCriterion() do
4         p_new.g ⟵ mutate(p⋆.g)
          // Implicitly: p_new.x ⟵ gpm(p_new.g)
5         if f(p_new.x) < f(p⋆.x) then p⋆ ⟵ p_new
6     return p⋆.x
7 end
```

10.2 General Information

10.2.1 Areas Of Application

Some example areas of application of hill climbing are:

Application	References
Networking and Communication	[2268] see Section 23.2 on page 401
Robotics	[790]
Data Mining and Data Analysis	[646]
Evolving Behaviors, e.g., for Agents or Game Players	[2017]
Combinatorial Optimization	[953, 347]

10.3 Multi-Objective Hill Climbing

As illustrated in Algorithm 10.2 on the next page, we can easily extend hill climbing algorithms with a support for multi-objective optimization by using some of the methods of evolutionary algorithms. This extended approach will then return a set $X^\star$ of the best solutions found instead of a single individual $x^\star$ as done in Algorithm 10.1. The set of currently known best individuals *Arc* may contain more than one element. Therefore, we employ a selection scheme in order to determine which of these individuals should be used as parent for the next offspring in the multi-objective hill climbing algorithm. The selection algorithm applied must not solely rely on the prevalence comparison, since no element in *Arc* prevails any other. Thus, we also copy the idea of fitness assignment from evolutionary algorithms. For maintaining the optimal set, we apply the updating and pruning methods defined in Chapter 19 on page 307.

Algorithm 10.2: $x^\star \longleftarrow$ hillClimberMO(cmp_F, a)

Input: cmp_F: the prevalence comparator
Input: as: the maximum archive size
Data: p_{new}: the new individual generated
Data: Arc: the set of best individuals known
Output: $X^\star$: the set of the best elements found

```
1  begin
2      Arc ⟵ ()
3      p_new.g ⟵ create()
       // Implicitly: p_new.x ⟵ gpm(p_new.g)
4      while terminationCriterion() do
5          Arc ⟵ updateOptimalSet(Arc, p_new)
6          Arc ⟵ pruneOptimalSet(Arc, as)
7          v ⟵ assignFitness(Arc, cmp_F)
8          p_new ⟵ select(Arc, v, 1) [0]
9          p_new.g ⟵ mutate(p_new.g)
           // Implicitly: p_new.x ⟵ gpm(p_new.g)
10     return extractPhenotypes(Arc)
11 end
```

10.4 Problems in Hill Climbing

Both versions of the algorithm are still very likely to get stuck on local optima. They will only follow a path of solution candidates if it is monotonously[2] improving the objective function(s). Hill climbing in this form is a local search rather than global optimization algorithm. By making a few slight modifications to the algorithm however, it can become a valuable global optimization technique:

1. A tabu-list which stores the elements recently evaluated can be added. By preventing the algorithm from visiting them again, a better exploration of the problem space $\mathbb{X}$ can be enforced. This technique is used in Tabu Search which is discussed in Chapter 14 on page 273.
2. Another way of preventing premature convergence is to not always transcend to the better solution candidate in each step. Simulated Annealing introduces a heuristic based on the physical model the cooling down molten metal to decide whether a superior offspring should replace its parent or not. This approach is described in Chapter 12 on page 263.
3. The Dynamic Hill Climbing approach by Yuret and de la Maza [2303] uses the last two visited points to compute unit vectors. With this technique, the directions are adjusted according to the structure of the problem space and a new coordinate frame is created which points more likely into the right direction.
4. Randomly restarting the search after so-and-so many steps is a crude but efficient method to explore wide ranges of the problem space with hill climbing. You can find it outlined in Section 10.5.
5. Using a reproduction scheme that not necessarily generates solution candidates directly neighboring $x^\star$, as done in Random Optimization, an optimization approach defined in Chapter 11 on page 259, may prove even more efficient.

[2] http://en.wikipedia.org/wiki/Monotonicity [accessed 2007-07-03]

10.5 Hill Climbing with Random Restarts

Hill climbing with random restarts is also called *Stochastic Hill Climbing* (SH) or *Stochastic gradient descent*[3] [1923, 605]. We have mentioned it as a measure for preventing premature convergence; here we want to take a deeper look on this approach.

Let us further combine it directly with the multi-objective hill climbing approach defined in Algorithm 10.2. The new algorithm incorporates two archives for optimal solutions: Arc_1, the overall optimal set, and Arc_2 the set of the best individuals in the current run. We additionally define the criterion shouldRestart() which is evaluated in every iteration and determines whether or not the algorithm should be restarted. shouldRestart() therefore could, for example, count the iterations performed or check if any improvement was produced in the last ten iterations. After each single run, Arc_2 is incorporated into Arc_1, from which we extract and return the problem space elements at the end of the hill climbing process, as defined in Algorithm 10.3.

Algorithm 10.3: $X^\star \longleftarrow$ hillClimberMO_RR(cmp_F, a)

Input: cmp_F: the prevalence comparator
Input: as: the maximum archive size
Data: p_{new}: the new individual generated
Data: Arc_1, Arc_2: the sets of best individuals known
Output: $X^\star$: the set of the best elements found

```
1  begin
2      Arc_1 ⟵ ()
3      while ¬terminationCriterion() do
4          Arc_2 ⟵ ()
5          p_new.g ⟵ create()
           // Implicitly: p_new.x ⟵ gpm(p_new.g)
6          while ¬(terminationCriterion() ∨ shouldRestart()) do
7              Arc_2 ⟵ updateOptimalSet(Arc_2, p_new)
8              Arc_2 ⟵ pruneOptimalSet(Arc_2, as)
9              v ⟵ assignFitness(Arc_2, cmp_F)
10             p_new ⟵ select(Arc_2, v, 1) [0]
11             p_new.g ⟵ mutate(p_new.g)
               // Implicitly: p_new.x ⟵ gpm(p_new.g)
12         Arc_1 ⟵ updateOptimalSetN(Arc_1, Arc_2)
13         Arc_1 ⟵ pruneOptimalSet(Arc_1, as)
14     return extractPhenotypes(Arc_1)
15 end
```

10.6 GRASP

Greedy Randomized Adaptive Search Procedures (*GRASP*s) [663, 652, 1648, 1722] are metaheuristics which repeatedly create new starting points and refine these with a local search algorithm until a termination criterion is met. In this, they are similar to hill climbing with random restarts.

The initial construction phase of each iteration, however, may be much more complicated than just randomly picking a new point in the search space. Feo and Resende [652] describe it as an iterative construction process where one element [gene] is "added" at a time where

[3] http://en.wikipedia.org/wiki/Stochastic_gradient_descent [accessed 2007-07-03]

the element to be added is chosen with respect to a greedy function. Here, not necessarily the best possible allele is set, but one of the top candidates is picked randomly.

After the initial solution is generated this way, a local search is applied to refine it. Therefore, hill climbing, for instance, could be used as well as a deterministic search such as a IDDFS (see Section 17.3.4) or a greedy approach (see Section 17.4.1). Feo and Resende [652] argue that the efficiency and quality of the solutions produced by such GRASP processes are often much better than those of local searches started at random points.

10.6.1 General Information

Areas Of Application

Some example areas of application of GRASP are:

Application	References
Combinatorial Optimization	[651, 652, 1288]
Scheduling	[650]

Online Resources

Some general, online available ressources on GRASP are:

http://www.graspheuristic.org/ [accessed 2008-10-20]
Last update: 2004-02-29
Description: A website leading to a lage annontated bibliography on GRASP.

10.7 Raindrop Method

Only three years ago, Bettinger and Zhu [191, 2322] contributed a new search heuristic for constrained optimization, the Raindrop Method, which they used for forest planning problems. In the original description of the algorithm, the search and problem space are identical ($\mathbb{G} = \mathbb{X}$). The algorithm is based on precise knowledge of the components of the solution candidates and on how their interaction influences the validity of the constraints. It works as follows:

1. The Raindrop Method starts out with a single, valid solution candidate $x^\star$ (i.e., one that violates none of the constraints). This candidate may be found with a random search process or may be provided created by a human operator.
2. Create a copy x of $x^\star$. Set the iteration counter t to a user-defined maximum value T of iterations of modifying and correcting x that are allowed without improvements before reverting to $x^\star$.
3. Perturb x by randomly modifying one of its components. Let us refer to this randomly selected component as s. This modification may lead to constraint violations.
4. If no constraint was violated, continue at step 11, otherwise proceed as follows.
5. Set a distance value d to 0.
6. Create a list L of the components of x that lead to constraint violations. Here we make use the knowledge of the interaction of components and constraints.

7. From L, we pick the component c physically closest to s, that is, the component with the minimum distance $\text{dist}(c, s)$. In the original application of the Raindrop Method, *physically close* was properly defined due to the fact that the solution candidates were basically two-dimensional maps. For applications different from forest planning, appropriate definitions for the distance measure have to be supplied.
8. Set $d = \text{dist}(c, s)$.
9. Find the next best value for c which does not induce any new constraint violations in components f which are at least as close to s, i. e., with $\text{dist}(f, s) \leq \text{dist}(c, s)$. This change may, however, cause constraints violations in components farther away from s. If no such change is possible, go to point 13. Otherwise, modify the component c in x.
10. Go back to step 4.
11. If x is better than $x^\star$, that is, $x \succ x^\star$, set $x^\star = x$. Otherwise, decrease the iteration counter t.
12. If the termination criterion has not yet been met, go back to step 3 if $t > 0$ and to 2 if $t = 0$.
13. Return $x^\star$ to the user.

The iteration counter t here is used to allow the search to explore solutions more distance from the current optimum $x^\star$. The higher the initial value T specified user, the more iterations without improvement are allowed before reverting x to $x^\star$. By the way, the name Raindrop Method comes from the fact that the constraint violations caused by the perturbation of the valid solution radiate away from the modified component s like waves on a water surface radiate away from the point where a raindrop hits.

11

Random Optimization

11.1 Introduction

The Random Optimization[1] method for single-objective, numerical problems, i. e., $\mathbb{G} = \mathbb{R}^n$ and $|F| = 1$, was first proposed by Rastrigin [1709] in the early 1960s. It was studied thoroughly by Gurin and Rastrigin [870], Schumer [1838] and further improved by Schumer and Steiglitz [1839] [2206]. A different Random Optimization approach has been introduced by Matyas [1371, 1372] around the same time. Matyas gave theorems about the convergence properties of his approach for unimodal optimization. Baba [91] then showed theoretically that the global optimum of an optimization problem can even be found if the objective function is multimodal.

There are, however, three important differences between the two approaches:

1. In traditional hill climbing, the new solution candidates are created from a good individual are always very close neighbors of it. In Random Optimization, this is not *necessary* but only *probably*.
2. In Random Optimization, the unary search operation explicitly uses random numbers whereas the unary search operations of hill climbing may be deterministic or randomized.
3. In Random Optimization, the search space $\mathbb{G}$ is always the $\mathbb{R}^n$, the space of n-dimensional real vectors.
4. In Random Optimization, we explicitly distinguish between objective functions $f \in F$ and constraints $c \in C$.

Random Optimization introduces a new search operation "roReproduce" specialized for the numerical search space similar to mutation in Evolution Strategies. This operation is constructed in a way that all points in the search space $\mathbb{G} = \mathbb{R}^n$ can be reached in one step when starting out from every other point. In other words, the operator "roReproduce" is *complete* in the sense of Definition 1.27.

$$\text{roReproduce}(g) = g + \mathbf{r} : \mathbf{r} = \begin{pmatrix} \text{random}_n\left(\mu_1, \sigma_1^2\right) & \left(\sim N\left(\mu_1, \sigma_1^2\right)\right) \\ \text{random}_n\left(\mu_2, \sigma_2^2\right) & \left(\sim N\left(\mu_2, \sigma_2^2\right)\right) \\ & \cdots \\ \text{random}_n\left(\mu_n, \sigma_m^2\right) & \left(\sim N\left(\mu_n, \sigma_n^2\right)\right) \end{pmatrix} \tag{11.1}$$

Equation 11.1 illustrates one approach to realize such a complete search operation. To the (genotype) of the best solution candidate discovered so far, we add a vector $\mathbf{r} \in \mathbb{R}^n$. Each component $\mathbf{r}_{[i]}$ of this vector is normally distributed around a value μ_i. Hence, the probability density function underlying the components of this vector is greater than zero for all real numbers. The μ_i are the expected values and the σ_i the standard deviations of the

[1] http://en.wikipedia.org/wiki/Random_optimization [accessed 2007-07-03]

normal distributions, as introduced in Section 28.5.2 on page 486. The μ_i define a general direction for the search, i. e., if $\mu_i > 0$, $\text{random}_n\left(\mu_i, \sigma_i^2\right)$ will likely also be greater zero and for $\mu_i < 0$, it will probably be smaller than zero, too (given that $|\sigma_i| \ll |expectedValueM_i|$). The σ_i can be imagined as the range in which the random numbers are distributed around the μ_i and denote a step width of the random numbers. If we choose the absolute values of both, μ_i and σ_i, very small, we can exploit a local optimum whereas larger values lead to a rougher exploration of search space. If the μs are set to 0, the probability distribution of random numbers will be "centered" around the genotype it is applied to. Since the normal distribution is generally unbounded, it is possible that the random elements in $\mathbf{r}$ can become very large, even for small $\sigma_i \approx 0$. Therefore, local optima can be left again even with bad settings of μ and σ.

Equation 11.1 is only one way to realize the completeness of the "roReproduce"-operation. Instead of the normal distribution, any other probability distribution with $f_X(y) > 0 \; \forall y \in \mathbb{R}$ would do. Good properties can, for instance, be attributed to the bell-shaped distribution used by Worakul et al. [2255, 2256] and discussed in Section 28.9.3 on page 530 in this book.

In order to respect the idea of constraint optimization in Random Optimization as introduced in [91], we define a set of constraint functions C. A constraint $c \in C$ is satisfied by a solution candidate $x \in \mathbb{X}$, if $c(x) \leq 0$ holds.

Algorithm 11.1 illustrates how random optimization works, clearly showing connatural traits in comparison with the hill climbing approach Algorithm 10.1 on page 254.

Algorithm 11.1: $x^\star \longleftarrow \text{randomOptimizer}f$

Input: f: the objective function subject to minization
Data: p_{new}: the new element created
Data: $p^\star$: the (currently) best individual
Output: $x^\star$: the best element found

```
1 begin
2     p*.g ⟵ create()
      // Implicitly: p*.x ⟵ gpm(p*.g)
3     while terminationCriterion() do
4         p_new.g ⟵ roReproduce(p*.g)
          // Implicitly: p_new.x ⟵ gpm(p_new.g)
5         if c(p_new.x) ≤ 0 ∀c ∈ C then
6             if f(p_new.x) < f(p*.x) then p* ⟵ p_new
7     return p*.x
8 end
```

Setting the values of μ and σ adaptively can lead to large improvements in convergence speed. The Heuristic Random Optimization (HRO) algorithm introduced by Li and Rhinehart [1277] and its successor method Random Optimization II developed by Chandran and Rhinehart [373] for example update them by utilizing gradient information or reinforcement learning.

11.2 General Information

11.2.1 Areas Of Application

Some example areas of application of (heuristic) Random Optimization are:

Application	References
Medicine	[2255, 2256]
Biology and Medicine	[558]
Machine Learning	[1249]
Function Optimization	[1277, 1917]

12

Simulated Annealing

12.1 Introduction

In 1953, Metropolis et al. [1396] developed a Monte Carlo method for "calculating the properties of any substance which may be considered as composed of interacting individual molecules". With this so-called "Metropolis" procedure stemming from statistical mechanics, the manner in which metal crystals reconfigure and reach equilibria in the process of annealing can be simulated. This inspired Kirkpatrick et al. [1142] to develop the Simulated Annealing[1] (SA) algorithm for global optimization in the early 1980s and to apply it to various combinatorial optimization problems. Independently, Černý [363] employed a similar approach to the travelling salesman problem [1263, 78]. Simulated Annealing is an optimization method that can be applied to arbitrary search and problem spaces. Like simple hill climbing algorithms, Simulated Annealing only needs a single initial individual as starting point and a unary search operation.

In metallurgy and material science, annealing[2] is a heat treatment of material with the goal of altering its properties such as hardness. Metal crystals have small defects, dislocations of ions which weaken the overall structure. By heating the metal, the energy of the ions and, thus, their diffusion rate is increased. Then, the dislocations can be destroyed and the structure of the crystal is reformed as the material cools down and approaches its equilibrium state. When annealing metal, the initial temperature must not be too low and the cooling must be done sufficiently slowly so as to avoid the system getting stuck in a meta-stable, non-crystalline, state representing a local minimum of energy.

In physics, each set of positions of all atoms of a system pos is weighted by its Boltzmann probability factor $e^{-\frac{E(pos)}{k_B T}}$ where $E(pos)$ is the energy of the configuration pos, T is the temperature measured in Kelvin, and k_B is the Boltzmann's constant[3] $k_B = 1.380\,650\,524 \cdot 10^{-23}\text{J/K}$.

The Metropolis procedure was an exact copy of this physical process which could be used to simulate a collection of atoms in thermodynamic equilibrium at a given temperature. A new nearby geometry pos_{i+1} was generated as a random displacement from the current geometry pos_i of an atom in each iteration. The energy of the resulting new geometry is computed and ΔE, the energetic difference between the current and the new geometry, was determined. The probability that this new geometry is accepted, $P(\Delta E)$ is defined in Equation 12.2.

[1] http://en.wikipedia.org/wiki/Simulated_annealing [accessed 2007-07-03]
[2] http://en.wikipedia.org/wiki/Annealing_(metallurgy) [accessed 2008-09-19]
[3] http://en.wikipedia.org/wiki/Boltzmann%27s_constant [accessed 2007-07-03]

$$\Delta E = E(pos_{i+1}) - E(pos_i) \tag{12.1}$$

$$P(\Delta E) = \begin{cases} e^{-\frac{\Delta E}{k_B T}} & \text{if } \Delta E > 0 \\ 1 & \text{otherwise} \end{cases} \tag{12.2}$$

Thus, if the new nearby geometry has a lower energy level, the transition is accepted. Otherwise, a uniformly distributed random number $r = \text{random}_u() \in [0, 1)$ is drawn and the step will only be accepted in the simulation if it is less or equal the Boltzmann probability factor, i. e., $r \leq P(\Delta E)$. At high temperatures T, this factor is very close to 1, leading to the acceptance of many uphill steps. As the temperature falls, the proportion of steps accepted which would increase the energy level decreases. Now the system will not escape local regions anymore and (hopefully) comes to a rest in the global minimum at temperature $T = 0$K.

The abstraction of this method in order to allow arbitrary problem spaces is straightforward – the energy computation $E(pos_i)$ is replaced by an objective function f or even by the result v of a fitness assignment process. Algorithm 12.1 illustrates the basic course of Simulated Annealing. Without loss of generality, we reuse the definitions from evolutionary algorithms for the search operations and set $Op = \{\text{create}, \text{mutate}\}$.

Algorithm 12.1: $x^\star \longleftarrow \text{simulatedAnnealing}(f)$

Input: f: the objective function to be minimized
Data: p_{new}: the newly generated individual
Data: p_{cur}: the point currently investigated in problem space
Data: $p^\star$: the best individual found so far
Data: T: the temperature of the system which is decreased over time
Data: t: the current time index
Data: ΔE: the enery difference of the x_{new} and x_{cur}
Output: $x^\star$: the best element found

```
1  begin
2      p_new.g ⟵ create()
       // Implicitly: p_new.x ⟵ gpm(p_new.g)
3      p_cur ⟵ p_new
4      p* ⟵ p_new
5      t ⟵ 0
6      while ¬terminationCriterion() do
7          ΔE ⟵ f(p_new.x) − f(p_cur.x)
8          if ΔE ≤ 0 then
9              p_cur ⟵ p_new
10             if f(p_cur.x) < f(p*.x) then p* ⟵ p_cur
11         else
12             T ⟵ getTemperature(t)
13             if random_u() < e^(−ΔE/(k_B T)) then p_cur ⟵ p_new
14         p_new.g ⟵ mutate(p_cur.g)
           // Implicitly: p_new.x ⟵ gpm(p_new.g)
15         t ⟵ t + 1
16     return p*.x
17 end
```

It has been shown that Simulated Annealing algorithms with appropriate cooling strategies will asymptotically converge to the global optimum. Nolte and Schrader [1540] and van Laarhoven and Aarts [2095] provide lists of the most important works showing that Simulated Annealing will converge to the global optimum if $t \to \infty$ iterations are performed, including the studies of Hajek [879]. Nolte and Schrader [1540] further list research providing deterministic, non-infinite boundaries for the asymptotic convergence by Anily and

Federgruen [70], Gidas [804], Nolte and Schrader [1539], and Mitra et al. [1437]. In the same paper, they introduce a significantly lower bound, which, however, still states that Simulated Annealing is probably in an optimal configuration after the number of iterations exceeds the cardinality of the problem space – which is, well, slow [1017]. In other words, it would be faster to enumerate all possible solution candidates in order to find the global optimum with absolute certainty than applying Simulated Annealing. This does not mean that Simulated Annealing is always slow. It only needs that much time if we persist on the optimality. Speeding up the cooling process will result in a faster search, but voids the guaranteed convergence on the other hand. Such speeded-up algorithms are called *Simulated Quenching* (SQ) [1014, 1813, 808].

12.2 General Information

12.2.1 Areas Of Application

Some example areas of application of Simulated Annealing are:

Application	References
Combinatorial Optimization	[363, 1142, 298, 286, 473]
Function Optimization	[818]
Chemistry, Chemical Engineering	[1292, 297, 1075, 1401]
Image Processing	[1982, 2246, 2287, 298]
Economics and Finance	[1015, 1016]
Electrical Engineering and Circuit Design	[1781, 298]
Machine Learning	[1366, 1349, 298, 2070]
Geometry and Physics	[1367, 298, 1368, 1853]
Networking and Communication	[1683] see Section 23.2 on page 401

For more information see also [2095].

12.2.2 Books

Some books about (or including significant information about) Simulated Annealing are:

van Laarhoven and Aarts [2095]: *Simulated Annealing: Theory and Applications*
Tan [2001]: *Simulated Annealing*
Badiru [113]: *Handbook of Industrial and Systems Engineering*
Davis [494]: *Genetic Algorithms and Simulated Annealing*

12.3 Temperature Scheduling

The temperature schedule defines how the temperature in Simulated Annealing is decreased. As already mentioned, this has major influence on whether the Simulated Annealing algorithm will succeeded, on whether how long it will take to find the global optimum, and on whether or not it will degenerate to simulated quenching. For the later use in the Simulated Annealing algorithm, let us define the new operator getTemperature(t) which computes the temperature to be used at iteration t in the optimization process. For "getTemperature", a few general rules hold. All schedules start with a temperature T_{start} which is greater than

zero. If the number of iterations t approaches infinity, the temperature must become 0K. This is a very weak statement, since we have shown that there exist finite boundaries after which Simulated Annealing is most likely to have converged. So there will be a finite t_{end} in all practical realizations after which the temperature drops to 0K, as shown in Equation 12.6.

$$T \in \mathbb{R}^+, t \in \mathbb{N}_0 \; \forall \quad T = \text{getTemperature}(t) \tag{12.3}$$

$$T_{start} = \text{getTemperature}(0) > 0 \tag{12.4}$$

$$\lim_{t \to \infty} \text{getTemperature}(t) = 0\text{K} \tag{12.5}$$

$$\exists t_{end} \in \mathbb{N} : \text{getTemperature}(t) = 0\text{K} \; \forall t \geq t_{end} \tag{12.6}$$

There exists a wide range of methods to determine this temperature schedule. Miki et al. [1414], for example, used genetic algorithms for this purpose. We will introduce only the three simple variants here given by Press et al. [1675].

1. Reduce T to $(1 - \epsilon)\,T$ after every m iterations, where the exact values of $0 < \epsilon < 1$ and $m > 0$ are determined by experiment.
2. Grant a total of K iterations, and reduce T after every m steps to a value $T = T_{start}\left(1 - \frac{t}{K}\right)^{\alpha}$ where t is the inted of the current iteration and α is a constant, maybe 1, 2, or 4. α depends on the positions of the relative minima. Large values of α will spend more iterations at lower temperature.
3. After every m moves, set T to β times $\Delta E_c = f(x_{cur}) - f(x^\star)$, where β is an experimentally determined constant, $f(x_{cur})$ is the objective value of the currently examined solution candidate x_{cur}, and $f(x^\star)$ is the objective value of the best phenotype $x^\star$ found so far. Since ΔE_c may be 0, we limit the temperature change to a maximum of $T * \gamma$ with $0 < \gamma < 1$.

If we let the temperature sink fast, we will lose the property of guaranteed convergence. In order to avoid getting stuck at local optima, we can then apply random restarting, which already has been discussed in the context of hill climbing in Section 10.5 on page 256.

12.4 Multi-Objective Simulated Annealing

Again, we want to combine this algorithm with multi-objective optimization and also enable it to return a set of optimal solutions. This can be done even simpler than in multi-objective hill climbing. Basically, we just need to replace the single objective function f with the fitness values v computed by a fitness assignment process on basis of the set of currently known best solutions (Arc), the currently investigated individual (p_{cur}), and the newly created points in the search space (p_{new}).

Algorithm 12.2: $x^\star \longleftarrow \text{simulatedAnnealingMO}(\text{cmp}_F, a)$

Input: f: the objective function to be minimized
Input: as: the maximum number of individuals allowed to be stored in the archive
Data: p_{new}: the newly generated individual
Data: p_{cur}: the point currently investigated in problem space
Data: Arc: the set of best individuals found so far
Data: T: the temperature of the system which is decreased over time
Data: t: the current time index
Data: ΔE: the enery difference of the x_{new} and x_{cur}
Output: $x^\star$: the best element found

```
1  begin
2      p_new.g ⟵ create()
       // Implicitly: p_new.x ⟵ gpm(p_new.g)
3      p_cur ⟵ p_new
4      Arc ⟵ createList(1, p_cur)
5      t ⟵ 0
6      while ¬terminationCriterion() do
7          v ⟵ assignFitness(Arc ∪ {p_new, p_cur}, cmp_F)
8          ΔE ⟵ v(p_new.x) − v(p_cur.x)
9          if ΔE ≤ 0 then
10             p_cur ⟵ p_new
11         else
12             T ⟵ getTemperature(t)
13             if random_u() < e^(−ΔE/(k_B T)) then p_cur ⟵ p_new
14         Arc ⟵ updateOptimalSet(Arc, p_new)
15         Arc ⟵ pruneOptimalSet(Arc, as)
16         p_new.g ⟵ mutate(p_cur.g)
           // Implicitly: p_new.x ⟵ gpm(p_new.g)
17         t ⟵ t + 1
18     return extractPhenotypes(Arc)
19 end
```

13

Extremal Optimization

13.1 Introduction

13.1.1 Self-Organized Criticality

Different from Simulated Annealing, is a optimization method based on the metaphor of thermal equilibria from physics, the Extremal Optimization[1] (EO) algorithm of Boettcher and Percus [237, 238, 239, 240, 236] is inspired by ideas of non-equilibrium physics. Especially important in this context is the property of *self-organized criticality*[2] (SOC) [119, 1049]. The theory of SOC states that large interactive systems evolve to a state where a change in one single of their elements may lead to avalanches or domino effects that can reach any other element in the system. The probability distribution of the number of elements n involved in these avalanches is proportional to $n^{-\tau}$ ($\tau > 0$). Hence, mass changes involving only few elements are most likely, but even avalanches involving the whole system are possible with a non-zero probability. [526]

13.1.2 The Bak-Sneppens model of Evolution

The Bak-Sneppens model of evolution [118] exhibits self-organizing criticality and was the inspiration for Extremal Optimization. Rather than focusing on single species, this model considers a whole ecosystem and the co-evolution of many different species.

In the model, each species is represented only by a real fitness value between 0 and 1. In each iteration, the species with the lowest fitness is mutated. The model does not include any representation for genomes, instead, mutation changes the fitness of the species directly by replacing it with a random value uniformly distributed in $[0, 1]$. In nature, this corresponds to the process where one species has developed further or was replaced by another one.

So far, mutation (i. e., development) would become less likely the more the fitness increases. Fitness can also be viewed as a barrier: New characteristics must be at least as fit as the current ones to proliferate. In an ecosystem however, no species lives alone but depends on others, on its successors and predecessors in the food chain, for instance. Bak and Sneppen [118] consider this by arranging the species in a one dimensional line. If one species is mutated, the fitness values of its successor and predecessor in that line are also set to random values. In nature, the development of one species can foster the development of others and this way, even highly fit species may become able to (re-)adapt.

After a certain amount of iterations, the species in simulations based on this model reach a highly-correlated state of self-organized critically where all of them have a fitness above a certain threshold. This state is very similar to the idea of punctuated equilibria from evolutionary biology and groups of species enter a state of passivity lasting multiple cycles.

[1] http://en.wikipedia.org/wiki/Extremal_optimization [accessed 2008-08-24]
[2] http://en.wikipedia.org/wiki/Self-organized_criticality [accessed 2008-08-23]

Sooner or later, this state is interrupted because mutations occurring nearby undermine their fitness. The resulting fluctuations may propagate like avalanches through the whole ecosystem. Thus, such non-equilibrium systems exhibit a state of high adaptability without limiting the scale of change towards better states [236].

13.2 Extremal Optimization and Generalized Extremal Optimization

Boettcher and Percus [237] want to utilize this phenomenology to obtain near-optimal solutions for optimization problems. In Extremal Optimization, the search spaces $\mathbb{G}$ are always spaces of structured tuples $g = (g_{[1]}, g_{[2]}, \ldots, g_{[n]})$. Extremal Optimization works on a single individual and requires some means to determine the contributions of its genes to the overall fitness.

Extremal Optimization was originally applied to a graph bi-partitioning problem [237], were the n points of a graph had to be divided into two groups, each of size $n/2$. The objective was to minimize the number of edges connecting the two groups. Search and problem space can be considered as identical and a solution candidate $x = \text{gpm}(g) = g$ consisted of n genes, each of which standing for one point of the graph and denoting the Boolean decision to which set it belongs. Analogously to the Bak-Sneppens model, each such gene $g_{[i]}$ had an *own* fitness contribution $\lambda(g_{[i]})$, the ratio of its outgoing edges connected to nodes from the same set in relation to its total edge number. The higher this value, the better, but notice that $f(x = g) \neq \sum_{i=1}^{n} \lambda(g_{[i]})$, since $f(g)$ corresponds to the number of edges crossing the cut. In general, the Extremal Optimization algorithm proceeds as follows:

1. Create an initial individual p with a random genotype $p.g$ and set the currently best known solution candidate $x^\star$ to its phenotype: $x^\star = p.x$.
2. Sort all genes $p.g_{[i]}$ of $p.g$ in a list in ascending order according to their fitness contribution $\lambda(p.g_{[i]})$.
3. Then, the gene $p.g_{[i]}$ with the lowest fitness contribution is selected from this list and modified randomly, leading to a new individual p and a new solution candidate $x = p.x = \text{gpm}(p.g)$.
4. If $p.x$ is better that $x^\star$, i. e., $p.x \succ x^\star$, set $x^\star = p.x$.
5. If the termination criterion has not yet been met, continue at step 2.

Instead of always picking the weakest part of g, Boettcher and Percus [238] selected the gene(s) to be modified randomly in order to prevent the method from getting stuck in local optima. In their work, the probability of a gene at list index j for being drawn is proportional to $j^{-\tau}$. This variation was called τ-EO and showed superior performance compared to the simple Extremal Optimization. In the graph partitioning problem on which Boettcher and Percus [238] have worked, two genes from different sets needed to be drawn this way in each step, since always two nodes had to be swapped in order to keep the size of the sub-graphs constant. Values of τ in $1.3 \ldots 1.6$ have been reported to produce good results [238].

The major problem a user is confronted with in Extremal Optimization is how to determine the fitness contributions $\lambda(p.g_{[i]})$ of the elements $p.g_{[i]}$ of the genotypes $p.g$ of the solution candidates $p.x$. Boettcher and Percus [239] point out themselves that the "drawback to EO is that a general definition of fitness for individual variables may prove ambiguous or even impossible" [526]. de Sousa and Ramos [524, 525, 526] therefore propose and extension to EO, called the Generalized Extremal Optimization (GEO) for fixed-length binary genomes $\mathbb{G} = \mathbb{B}^n$. Each gene (bit) $p.g_{[i]}$ in the element $p.g$ of the search space currently examined, the following procedure is performed:

1. Create a copy g' of $p.g$.
2. Toggle bit i in g'.

3. Set $\lambda(p.g_{[i]})$ to $-f(\text{gpm}(g'))$ for maximization and to $f(\text{gpm}(g'))$ in case of minimization.[3]

By doing so, $\lambda(p.g_{[i]})$ becomes a measure for how adapted the gene is. If f is subject to maximization, high positive values of $f(\text{gpm}(g'))$ (corresponding to low $\lambda(p.g_{[i]})$) indicate that gene i should be mutated and has a low fitness. For minimization, low $f(\text{gpm}(g'))$ indicated the mutating gene i would yield high improvements in the objective value.

13.3 General Information

13.3.1 Areas Of Application

Some example areas of application of Extremal Optimization are:

Application	References
Combinatorial Optimization	[1363, 237, 238, 236]
Engineering, Structural Optimization, and Design	[526, 762, 1753]
Networking and Communication	[1363] see Section 23.2 on page 401
Function Optimization	[524]

[3] In the original work of de Sousa et al. [526], $f(x^\star)$ is subtracted from this value. Since we rank the genes, this has basically no influence and is omitted here.

14

Tabu Search

14.1 Introduction

Tabu Search[1] (TS) has been developed by Glover [810] in the mid 1980s [816]. Some of the basic ideas were introduced by Hansen [892] and further contributions in terms of formalizing this method have been made by Glover [811, 812], and de Werra and Hertz [529] (as summarized by Hertz et al. [919] in their tutorial on Tabu Search) as well as by Battiti and Tecchiolli [158] and Cvijović and Klinowski [471].

The word "tabu"[2] stems from Polynesia and describes a sacred place or object. Things that are *tabu* must be left alone and may not be visited or touched. Tabu Search extends hill climbing by this concept – it declares solution candidates which have already been visited as tabu. Hence, they must not be visited again and the optimization process is less likely to get stuck on a local optimum. The simplest realization of this approach is to use a list *tabu* which stores all solution candidates that have already been tested. If a newly created phenotype can be found in this list, it is not investigated but rejected right away. Of course, the list cannot grow infinitely but has a finite maximum length n. If the $n+1$st solution candidate is added, the first one must be removed. Alternatively, this list could also be reduced with clustering. If some distance measure in the problem space $\mathbb{X}$ is available, a certain perimeter around the listed solution candidates can be declared as tabu. More complex approaches will store specific properties of the individuals instead of the phenotypes themselves in the list. This will not only lead to more complicated algorithms, but may also reject new solutions which actually are very good. Therefore, aspiration criteria can be defined which override the tabu list and allow certain individuals.

[1] http://en.wikipedia.org/wiki/Tabu_search [accessed 2007-07-03]

[2] http://en.wikipedia.org/wiki/Tapu_%28Polynesian_culture%29 [accessed 2008-03-27]

Algorithm 14.1: $x^\star \longleftarrow \text{tabuSearch}(f, n)$

Input: f: the objective function subject to minization
Input: n: the maximum length of the tabu list ($n > 0$)
Data: p_{new}: the new element created
Data: $p^\star$: the (currently) best individual
Data: *tabu*: the tabu list
Output: $x^\star$: the best element found

```
1  begin
2      p*.g ⟵ create()
       // Implicitly: p*.x ⟵ gpm(p*.g)
3      tabu ⟵ createList(1, p*.x)
4      while ¬terminationCriterion() do
5          p_new.g ⟵ mutate(p*.g)
           // Implicitly: p_new.x ⟵ gpm(p_new.g)
6          if searchItem_u(p_new.x, tabu) < 0 then
7              if f(p_new.x) < f(p*.x) then p* ⟵ p_new
8              if len(tabu) ≥ n then tabu ⟵ deleteListItem(tabu, 0)
9              tabu ⟵ addListItem(tabu, p_new.x)
10     return p*.x
11 end
```

14.2 General Information

14.2.1 Areas Of Application

Some example areas of application of Tabu Search are:

Application	References
Combinatorial Optimization	[336, 1829, 1010, 815, 814, 983, 2049, 1612, 47, 112, 285]
Machine Learning	[1855, 529]
Biochemistry	[1989]
Operations Research	[674]
Networking and Communication	[1641, 1643, 1642, 1683, 1953] see Section 23.2 on page 401

14.2.2 Books

Some books about (or including significant information about) Tabu Search are:

Pardalos and Du [1612]: *Handbook of Combinatorial Optimization*
Badiru [113]: *Handbook of Industrial and Systems Engineering*
Reeves [1716]: *Modern Heuristic Techniques for Combinatorial Problems*
Jaziri [1045]: *Local Search Techniques: Focus on Tabu Search*

14.3 Multi-Objective Tabu Search

The simple Tabu Search is very similar to hill climbing and Simulated Annealing, as you can see when comparing it with Chapter 10 on page 253 and Chapter 12 on page 263). With Algorithm 14.2, we thus can define a multi-objective variant for Tabu Search in a manner very similar to the multi-objective hill climbing or multi-objective Simulated Annealing.

Algorithm 14.2: $x^\star \longleftarrow \text{tabuSearchMO}(\text{cmp}_F, n, a)$

Input: cmp_F: the prevalence comparator
Input: n: the maximum length of the tabu list ($n > 0$)
Input: as: the maximum archive size
Data: $tabu$: the tabu list
Data: p_{new}: the new individual generated
Data: Arc: the set of best individuals known
Output: $X^\star$: the set of the best elements found

```
1  begin
2      Arc ⟵ ()
3      p_new.x ⟵ create()
       // Implicitly: p_new.x ⟵ gpm(p_new.g)
4      tabu ⟵ ()
5      while terminationCriterion() do
6          if searchItem_u(p_new.x, tabu ∪ Arc) < 0 then
7              Arc ⟵ updateOptimalSet(Arc, p_new)
8              Arc ⟵ pruneOptimalSet(Arc, as)
9              v ⟵ assignFitness(Arc, cmp_F)
10             if len(tabu) ≥ n then tabu ⟵ deleteListItem(tabu, 0)
11             tabu ⟵ addListItem(tabu, p_new.x)
12         p_new ⟵ select(Arc, v, 1) [0]
13         p_new.g ⟵ mutate(p_new.g)
           // Implicitly: p_new.x ⟵ gpm(p_new.g)
14     return extractPhenotypes(Arc)
15 end
```

15

Memetic and Hybrid Algorithms

Starting with the research contributed by Bosworth et al. [257] (1972), Bethke [190] (1980), and Brady [268] (1985), there is a long tradition of hybridizing evolutionary algorithms with other optimization methods such as hill climbing, Simulated Annealing, or Tabu Search [1893]. A comprehensive review on this topic has been provided by Grosan and Abraham [861, 862]. Such approaches are not limited to GAs as "basis", in Section 16.4 for example, we have already listed a wide variety of approaches to combine the downhill simplex with population-based optimization methods spanning from genetic algorithms to Differential Evolution and Particle Swarm Optimization. Today, many of these approaches can be subsumed under the umbrella term *Memetic Algorithms*[1] (MAs).

15.1 Memetic Algorithms

The principle of genetic algorithms is to simulate the natural evolution (where phenotypic features are encoded in genes) in order to solve optimization problems. The term Memetic Algorithm was coined by Moscato [1468, 1469] as allegory for simulating a social evolution (where behavioral patterns are passed on in *memes*[2]) for the same purpose. The concept *meme* has been defined by Dawkins [501] as "unit of imitation in cultural transmission".

Moscato [1468] uses the example of Chinese martial art Kung-Fu which has developed over many generations of masters teaching their students certain sequences of movements, the so-called *forms*. Each form is composed of a set of elementary aggressive and defensive patterns. These undecomposable sub-movements can be interpreted as memes. New memes are rarely introduced and only few amongst the masters of the art have the ability to do so. Being far from random, such modifications involve a lot of problem-specific knowledge and almost always result in improvements. Furthermore, only the best of the population of Kung-Fu practitioners can become masters and teach decibels. Kung-Fu fighters can determine their fitness by evaluating their performance or by competing with each other in tournaments.

Based on this analogon, Moscato [1468] creates an example for the travelling salesman problem [1263, 78] involving the three principles of

1. intelligent improvement based on local search with problem-specific operators,
2. competition in form of a selection procedure, and
3. cooperation in form of a problem-specific crossover operator.

Further research work directly focusing on Memetic Algorithms has been contributed by Moscato et al. [1468, 1551, 952, 307, 220, 1470], Radcliffe and Surry [1693], Digalakis and Margaritis [565, 566], and Krasnogor and Smith [1215]. Other contributors of early work

[1] http://en.wikipedia.org/wiki/Memetic_algorithm [accessed 2007-07-03]
[2] http://en.wikipedia.org/wiki/Meme [accessed 2008-09-10]

on genetic algorithm hybridization are Ackley [12] (1987), Goldberg [821] (1989), Gorges-Schleuter [835] (1989), Mühlenbein [1476, 1477, 1479] (1989), Brown et al. [294] (1989) and Davis [495] (1991).

The definition of Memetic Algorithms given by Moscato [1468] is relatively general and encompass many different approaches. Even though Memetic Algorithms are a metaphor based on social evolution, there also exist two theories in natural evolution which fit to the same idea of hybridizing evolutionary algorithms with other search methods [1598]. Lamarckism and the Baldwin effect are both concerned with phenotypic changes in living creatures and their influence on the fitness and adaptation of species.

15.2 Lamarckian Evolution

Lamarckian evolution[3] is a model of evolution accepted by science before the discovery of genetics. Superseding early the ideas of Erasmus Darwin [486] (the grandfather of Charles Darwin), de Lamarck [522] laid the foundations of the theory later known as *Lamarckism* with his book *Philosophie Zoologique* published in 1809. Lamarckism has two basic principles:

1. Individuals can attain new, beneficial characteristics during their lifetime and lose unused abilities.
2. They inherit their traits (also those acquired during their life) to their offspring.

While the first concept is obviously correct, the second one contradicts the state of knowledge in modern biology. This does not decrease the merits of de Lamarck, who provided an early idea about how evolution could proceed. In his era, things like genes and the DNA simply had not been discovered yet. Weismann [2189] was the first to argue that the heredity information of higher organisms is separated from the somatic cells and, thus, could not be influenced by them [2067]. In nature, no phenotype-genotype mapping can take place.

Lamarckian evolution can be "included" in evolutionary algorithms by performing a local search starting with each new individual resulting from applications of the reproduction operations. This search can be thought of as training or learning and its results are coded back into the genotypes $g \in \mathbb{G}$ [2215]. Therefore, this local optimization usually works directly in the search space $\mathbb{G}$. Here, algorithms such as greedy search hill climbing, Simulated Annealing, or Tabu Search can be utilized, but simply modifying the genotypes randomly and remembering the best results is also possible.

15.3 Baldwin Effect

The Baldwin effect[4], [1883, 2129, 2130] first proposed by Baldwin [123, 124], Morgan [1451, 1452], and Osborn [1586] in 1896, is a evolution theory which remains controversial until today [511, 1646]. Suzuki and Arita [1985] describe it as a "possible scenario of interactions between evolution and learning caused by balances between benefit and cost of learning" [2163]. Learning is a rather local phenomenon, normally involving only single individuals, whereas evolution usually takes place in the global scale of a population. The Baldwin effect combines both in two steps [2067]:

1. First, the lifetime learning gives the individuals the chance to adapt to their environment or even to change their phenotype. This *phenotypic plasticity*[5] may help the creatures to increase their fitness and, hence, their probability to produce more offspring. Different from Lamarckian evolution, the abilities attained this way do not influence the genotypes nor are inherited.

[3] http://en.wikipedia.org/wiki/Lamarckism [accessed 2008-09-10]
[4] http://en.wikipedia.org/wiki/Baldwin_effect [accessed 2008-09-10]
[5] http://en.wikipedia.org/wiki/Phenotypic_plasticity [accessed 2008-09-10]

2. In the second phase, evolution step by step generates individuals which can learn these abilities faster and easier and, finally, will have encoded them in their genome. Genotypical traits then replace the learning (or phenotypic adaptation) process and serve as an energy-saving shortcut to the beneficial traits. This process is called *genetic assimilation*[6] [2128, 2129, 2130, 2131].

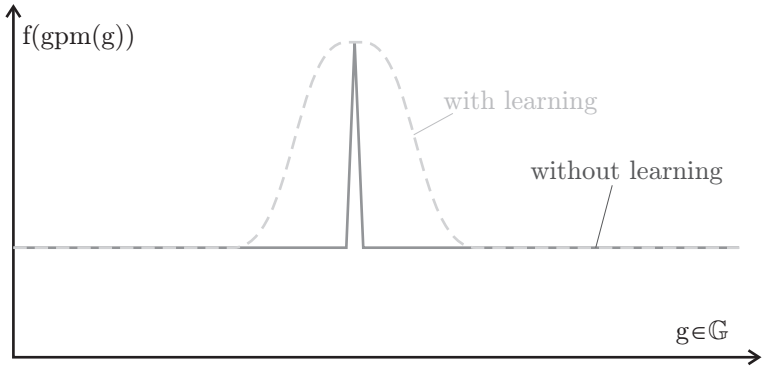


Fig. 15.1.a: The influence of learning capabilities of individuals on the fitness landscape according to [929, 930].

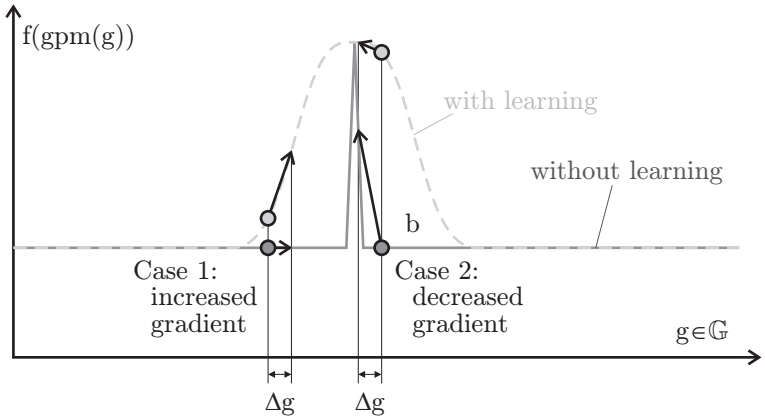


Fig. 15.1.b: The positive and negative influence of learning capabilities of individuals on the fitness landscape as in [1985].

Figure 15.1: The Baldwin effect.

Hinton and Nowlan [929, 930] were the first scientists performing experiments on the Baldwin effect with genetic algorithms [169, 904]. They found that the evolutionary interaction with learning smoothens the fitness landscape [864] and illustrated this effect on the example of a needle-in-a-haystack problem similar to Fig. 15.1.a. Mayley [1374] used experiments on Kauffman's NK fitness landscapes [1100] (see Section 21.2.1) to show that the Baldwin effect can also have negative influence: Whereas learning adds gradient information in regions of the search space which are distant from local or global optima (case 1 in Fig. 15.1.b), it decreases the information in their near proximity (called *hiding effect* [1374, 1062], case 2 in Fig. 15.1.b). One interpretation of this issue is that learning capabilities help individuals to survive in adverse conditions since they may find good abilities by learning and phenotypic adaptation. On the other hand, it makes not much of a difference

[6] http://en.wikipedia.org/wiki/Genetic_assimilation [accessed 2008-09-10]

whether an individual learns certain abilities or whether it was already born with them when it can exercise them at the same level of perfection. Thus, the selection pressure furthering the inclusion of good traits in the heredity information decreases if a life form can learn or adapt its phenotypes.

Suzuki and Arita [1985] found that the Baldwin effect decreases the evolution speed in their rugged experimental fitness landscape, but also led to significantly better results in the long term. By the way, Suzuki maintains a very nice bibliography on the Baldwin effect at http://www.alife.cs.is.nagoya-u.ac.jp/~reiji/baldwin/ [accessed 2008-09-10].

Like Lamarckian evolution, the Baldwin effect can also be added to evolutionary algorithms by performing a local search starting at each new offspring individual. Different from Lamarckism, the abilities and characteristics attained by this process only influence the objective values of an individual and are not coded back to the genotypes. Hence, it plays no role whether the search takes place in the search space $\mathbb{G}$ or in the problem space $\mathbb{X}$. The best objective values $F(p'.x)$ found in a search around individual p become its own objective values, but the modified variant p' of p actually scoring them is discarded [2215]. Nevertheless, the implementer will store these individuals somewhere else if they were the best solution candidates ever found. She must furthermore ensure that the user will be provided with the *correct* objective values of the final set of solution candidates resulting from the optimization process ($F(p.x)$, not $F(p'.x)$).

15.4 Summary on Lamarckian and Baldwinian Evolution

Whitley et al. [2215] showed that both, Lamarckian and Baldwinian evolution, can improve the performance of a genetic algorithm. In their experiments, the Lamarckian strategies were generally faster but the Baldwin effect could provide better solution in some cases. Sasaki and Tokoro [1808] furthermore showed that Lamarckian search is better if the environment (i.e., the objective functions) is static whereas Baldwinian evolution leads to better results in dynamic landscapes. This is only logical since in the Lamarckian case, the configurations with the best objective values are directly encoded in the genome and we have highly specialized genotypes. When applying the Baldwin effect, on the other hand, the genotypes can remain general and only the phenotypes are adapted. The work of Paenke et al. [1598] on the influence of phenotypic plasticity on the genotype diversity further substantiates the positive effects of the Baldwin effect in dynamic environments.

15.5 General Information

15.5.1 Areas Of Application

Some example areas of application of Memetic Algorithms are:

Application	References
Combinatorial Optimization	[220, 307, 1395, 901, 2043]
Engineering, Structural Optimization, and Design	[901]
Biochemistry	[901]
Networking and Communication	[2286, 1685] see Section 23.2 on page 401
Scheduling	[901]
Operations Research	[886]

Function Optimization	[2042]

15.5.2 Online Resources

Some general, online available ressources on Memetic Algorithms are:

http://www.densis.fee.unicamp.br/~moscato/memetic_home.html [accessed 2008-04-03]	
Last update:	2002-08-16
Description:	The Memetic Algorithms' Home Page by Pablo Moscato

15.5.3 Books

Some books about (or including significant information about) Memetic Algorithms are:

Hart, Krasnogor, and Smith [901]:	*Recent Advances in Memetic Algorithms*
Corne, Dorigo, Glover, Dasgupta, Moscato, Poli, and Price [448]:	*New Ideas in Optimisation*
Glover and Kochenberger [813]:	*Handbook of Metaheuristics*
Grosan, Abraham, and Ishibuchi [862]:	*Hybrid Evolutionary Algorithms*

16

Downhill Simplex (Nelder and Mead)

16.1 Introduction

The downhill simplex[1] (or Nelder-Mead method or amoeba algorithm[2]) published by Nelder and Mead [1517] in 1965 is an single-objective optimization approach for searching the space of n-dimensional real vectors ($\mathbb{G} \subseteq \mathbb{R}^n$) [1561, 1230]. Historically, it is closely related to the simplex extension by Spendley et al. [1941] to the Evolutionary Operation method mentioned in Section 2.1.6 on page 101 [1276]. Since it only uses the values of the objective functions without any derivative information (explicit or implicit), it falls into the general class of direct search methods [2260, 2054], as most of the optimization approaches discussed in this book do.

Downhill simplex optimization uses $n+1$ points in the $\mathbb{R}^n$. These points form a polytope[3], a generalization of a polygone, in the n-dimensional space – a line segment in $\mathbb{R}^1$, a triangle in $\mathbb{R}^2$, a tetrahedron in $\mathbb{R}^3$, and so on. Nondegenerated simplexes, i.e., those where the set of edges adjacent to any vertex form a basis in the $\mathbb{R}^n$, have one important festure: The result of replacing a vertex with its reflection through the opposite face is again, a nondegenerated simplex (see Fig. 16.1.a). The goal of downhill simplex optimization is to replace the best vertex of the simplex with an even better one or to ascertain that it is a candidate for the global optimum [1276]. Therefore, its other points are constantly flipped around in an intelligent manner as we will outline in Section 16.3.

Like hill climbing approaches, the downhill simplex may not converge to the global minimum and can get stuck at local optima [1230, 1383, 2046]. Random restarts (as in Hill Climbing with Random Restarts discussed in Section 10.5 on page 256) can be helpful here.

16.2 General Information

16.2.1 Areas Of Application

Some example areas of application of downhill simplex are:

[1] http://en.wikipedia.org/wiki/Nelder-Mead_method [accessed 2008-06-14]

[2] In the book *Numerical Recipes in C++* by Press et al. [1675], this optimization method is called "amoeba algorithm".

[3] http://en.wikipedia.org/wiki/Polytope [accessed 2008-06-14]

Application	References
Chemistry, Chemical Engineering	[2142, 1401, 2127, 145]
Robotics	[371]
Physics	[1604, 1493]
Biochemistry	[2293]
Data Mining and Data Analysis	[1812]

16.2.2 Online Resources

Some general, online available ressources on downhill simplex are:

`http://math.fullerton.edu/mathews/n2003/NelderMeadMod.html` [accessed 2008-06-14]
Last update: 2004-07-22
Description: Nelder-Mead Search for a Minimum
`http://www.boomer.org/c/p3/c11/c1106.html` [accessed 2008-06-14]
Last update: 2003
Description: Nelder-Mead (Simplex) Method

16.2.3 Books

Some books about (or including significant information about) downhill simplex are:

Avriel [89]: *Nonlinear Programming: Analysis and Methods*
Walters, Morgan, Parker, Jr., and Deming [2142]: *Sequential Simplex Optimization: A Technique for Improving Quality and Productivity in Research, Development, and Manufacturing*
Press, Teukolsky, Vettering, and Flannery [1675]: *Numerical Recipes in C++. Example Book. The Art of Scientific Computing*

16.3 The Downhill Simplex Algorithm

In Algorithm 16.1, we define the downhill simplex optimization approach. For simplification purposes we set both, the problem and the search space, to the n-dimensional real vectors, i. e., $\mathbb{X} \subseteq \mathbb{G} \subseteq \mathbb{R}^n$. In the actual implementation, we can use any set as problem space, given that a genotype-phenotype mapping $\text{gpm} : \mathbb{R}^n \mapsto \mathbb{X}$ is provided. Furthermore, notice that we optimize only a single objective function f. We can easily extend this algorithm for multi-objective optimization by using a comparator function cmp_F based on a set of objective functions F instead of comparing the values of f. In Algorithm 10.2, we have created a multi-objective hill climbing method with the same approach.

For visualization purposes, we apply the downhill simplex method exemplarily to an optimization problem with $\mathbb{G} = \mathbb{X} = \mathbb{R}^2$, where the simplex S consists of three points, in Figure 16.1.

The optimization process described by Algorithm 16.1 starts with creating a sample of $n+1$ random points in the search space in line 2. Here, the createPop operation must ensure that these samples form a nondegenerated simplex. Notice that apart from the creation of the initial simplex, all further steps are deterministic and do not involve random numbers.

In each search step, the points in the simplex S are arranged in ascending order according to their corresponding objective values (line 4). Hence, the best solution candidate is $S_{[0]}$ and the worst is $S_{[n]}$. We then compute the center $\mathbf{m}$ of the n best points in line 5 and then

Algorithm 16.1: $x^\star \longleftarrow$ downhillSimplex(f)

Input: f: the objective function subject to minimization
Input: [implicit] n: the dimension of the search space
Input: [implicit] $\alpha, \rho, \gamma, \sigma$: the reflection, the expansion, the contraction, and the shrink coefficent
Data: S: the simplex
Data: $\mathbf{m}$: the centroid of the simplex
Data: $\mathbf{r}$: the reflexion
Data: $\mathbf{e}$: the expansion
Data: $\mathbf{c}$: the contraction
Data: i: a counter variable
Output: $x^\star$: the best solution candidate found

```
1  begin
2      S ⟵ createPop(n + 1)
3      while ¬terminationCriterion() do
4          S ⟵ sortList_a(S, f)
5          m ⟵ (1/n) Σ_{i=0}^{n-1} S_[i]
           // Reflection: reflect the worst point over m
6          r ⟵ m + α (m − S_[n])
7          if f(S_[0]) < f(r) < f(S_[n]) then
8              S_[n] ⟵ r
9          else
10             if f(r) < f(S_[0]) then
                   // Expansion: try to search farther in this direction
11                 e ⟵ r + γ (r − m)
12                 if f(e) < f(r) then S_[n] ⟵ e
13                 else S_[n] ⟵ r
14             else
15                 b ⟵ true
16                 if f(r) ≥ f(S_[n−1]) then
                       // Contraction: a test point between r and m
17                     c ⟵ ρr + (1 − ρ)m
18                     if f(c) ≤ f(r) then
19                         S_[n] ⟵ c
20                         b ⟵ false
21                 if b then
                       // Shrink towards the best solution candidate S_[0]
22                     for i ⟵ n down to 1 do
23                         S_[i] ⟵ S_[0] + σ (S_[i] − S_[0])
24     return S_[0]
25 end
```

reflect the worst solution candidate $S_{[n]}$ through this point in line 6, obtaining the new point $\mathbf{r}$ as also illustrated in Fig. 16.1.a. The reflection parameter α is usually set to 1.

In the case that $\mathbf{r}$ is somewhere in between of the points in the current simplex, i. e., neither better than $S_{[0]}$ nor as worse as $S_{[n]}$, we directly replace $S_{[n]}$ with it. This simple move was already present in the first simplex algorithm defined by Spendley et al. [1941]. The contribution of Nelder and Mead [1517] was to turn the simplex search into an optimization algorithm by adding new options. These special operators were designed for speeding up the optimization process by deforming the simplex in way that they suggested would better adapt to the objective functions [1276].

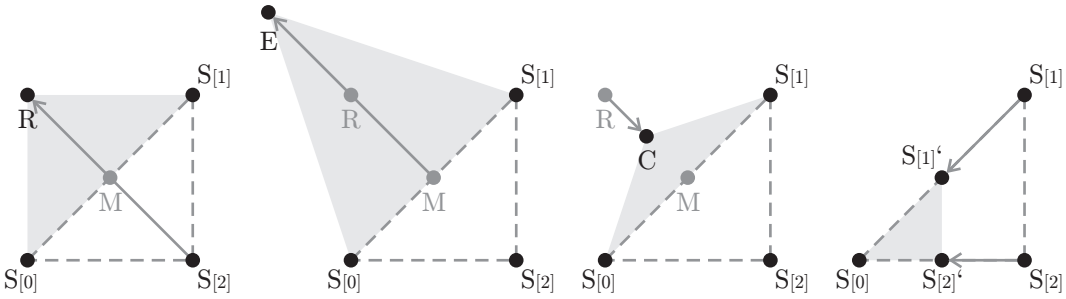


Fig. 16.1.a: Reflection

Fig. 16.1.b: Expansion

Fig. 16.1.c: Contraction

Fig. 16.1.d: Shrinking

Figure 16.1: One possible step of the downhill simplex algorithm applied to a problem in $\mathbb{R}^2$.

If $\mathbf{r}$ is better than the best solution candidate $S_{[0]}$, one of these operators is to expand the simplex further into this promising direction (line 11). As sketched in Fig. 16.1.b, we obtain the point $\mathbf{e}$ with the expansion parameter γ set to 1. We now choose the better one of these two points in order to replace $S_{[n]}$.

If $\mathbf{r}$ was no better than $S_{[n]}$, we the simplex is contracted by creating a point $\mathbf{c}$ somewhere in between of $\mathbf{r}$ and $\mathbf{m}$ in line 17. In Fig. 16.1.c, the contraction parameter ρ was set to $\frac{1}{2}$. We substitute $S_{[n]}$ with $\mathbf{c}$ only if $\mathbf{c}$ is better than $\mathbf{r}$,.

When everything else fails, we shrink the whole simplex in line 23 by moving all points (except $S_{[0]}$) into the direction of the current optimum $S_{[0]}$. The shrinking parameter σ normally has the value $\frac{1}{2}$, as is the case in the example outlined in Fig. 16.1.d.

16.4 Hybridizing with the Downhill Simplex

Interestingly, there are some similarities between evolutionary algorithms and the downhill simplex. Takahama and Sakai [1999], for instance, argue that the downhill simplex can be considered as an evolutionary algorithm with special selection and reproduction operators. Each search step of Nelder and Mead's algorithm could be regarded as an n-ary reproduction operation for search spaces that are subsets the $\mathbb{R}^n$. Also, there are vague similarities between search operations of Differential Evolution (described in Section 5.5 on page 229), Particle Swarm Optimization (introduced in Chapter 9 on page 249), and the reflection operator of downhill simplex. The joint work of Jakumeit et al. [1038] and Barth et al. [155], for example, goes into the direction of utilizing these similarities.

The research of Wang and Qiu [2144, 2146, 2145, 2147, 1680] focuses on the Hybrid Simplex Method PSO (HSMPSO), which, as the name says, is hybridization of Particle Swarm Optimization with Nelder and Mead's algorithm. In this approach, the downhill simplex operator is applied to each particle after a definite interval of iterations. Similar ideas of combining PSO with simplex methods are pursued by Fan et al. [643, 642].

Gao and Wang [770] emphasize the close similarities between the reproduction operators of Differential Evolution and the search step of the downhill simplex. Thus, it seems only logical to combine the two methods in form of a new Memetic Algorithm. The Low Dimensional Simplex Evolution (LDSE) of Luo and Yu [1333] incorporates the single search steps of the downhill simplex applied to a number m of points which is lower than the actual dimensionality of the problem n. Luo and Yu [1333, 1332] reported that this method is able to outperform Differential Evolution when applied to the test function set of Ali et al. [38].

There exist various other methods of hybridizing real-coded genetic algorithms with downhill simplex algorithm. Renders and Bersini [1719], for example, divide the population

into groups of $n + 1$ genotypes and allow the genetic algorithm to choose the Nelder-Mead simplex as additional reproduction operation besides crossover and simple averaging. The concurrent simplex of Yen et al. [2293, 2294], [2292] uses a probabilistic simplex method with $n + \Omega$ points instead of one, where the n best points are used to compute the centroid and the other $\Omega > 1$ points are reflected over it. They apply this idea to the top S individuals in the population, obtain $S - n$ children, and copy the best n individuals into the next generation. The remaining $(ps - S)$ genotypes (where ps is the population size) are created according to the conventional genetic algorithm reproduction scheme.

Barbosa et al. [145] also use the downhill simplex as reproduction operator in a real-coded genetic algorithm. Their operator performs up to a given number of simplex search steps (20 in their work) and leads to improved results. Again, this idea goes more into the direction of Memetic Algorithms. Further approaches for hybridizing genetic algorithms with the downhill simplex have been contributed by Musil et al. [1493], Zhang et al. [2314], [2313], and Satapathy et al. [1812].

The simplex crossover operator (SPX[4]) by Tsutsui et al. [2063], [925, 2061] also uses a simplex structure based on $n + 1$ real vectors for n dimensional problem spaces. It is, however, not directly related to the downhill simplex search.

[4] This abbreviation is also used for single-point crossover, see Section 3.4.4.

17

State Space Search

17.1 Introduction

State space search strategies[1] are not directly counted to the optimization algorithms. In global optimization, objective functions $f \in F$ are optimized. Normally, all elements of the problem space $\mathbb{X}$ are valid and only differ in their *utility* as solution. Optimization is about to find the elements with the best utility. State space search is instead based on a criterion $\text{isGoal} : \mathbb{X} \mapsto \mathbb{B}$ which determines whether an element of the problem space is a valid solution or not. The purpose of the search process is to find elements x from the solution space $\mathbb{S} \subseteq \mathbb{X}$, i. e., those for which $\text{isGoal}(x)$ is `true`. [1780, 446, 569]

Definition 17.1 (isGoal**).** The function $\text{isGoal} : \mathbb{X} \mapsto \mathbb{B}$ is the target predicate of state space search algorithms which states whether a given state $x \in \mathbb{X}$ is a valid solution (by returning `true`), i. e., the goal state, or not (by returning `false`). isGoal thus corresponds to membership in the solution space $\mathbb{S}$ defined in Definition 1.20 on page 42.

$$x \in \mathbb{S} \Leftrightarrow \text{isGoal}(x) = \texttt{true} \tag{17.1}$$

We will be able to apply state space search strategies for global optimization if we can define a threshold $\check{y}_i$ for each objective function $f_i \in F$. If we assume minimization, $\text{isGoal}(x)$ becomes `true` if and only if the values of all objective functions for x drop below given thresholds, and thus, we can define isGoal according to Equation 17.2.

$$\text{isGoal}(x) = \begin{cases} \texttt{true} \text{ if } f_i(x) \leq \check{y}_i \; \forall i \in 1..|F| \\ \texttt{false} \text{ otherwise} \end{cases} \tag{17.2}$$

In state space search algorithms, the search space $\mathbb{G}$ and the problem space $\mathbb{X}$ are often identical. Most state space search can only be applied if the search space is enumerable. One feature of the search algorithms introduced here is that they all are deterministic. This means that they will yield the same results in each run (when applied to the same problem, that is).

One additional operator needs to be defined for state space search algorithms: the "expand" function which helps enumerating the search space.

Definition 17.2 (expand**).** The operator $\text{expand} : \mathbb{G} \mapsto \mathcal{P}(\mathbb{G})$ receives one element g from the search space $\mathbb{G}$ as input and computes a set G of elements which can be reached from it.

"expand" is the exploration operation of state space search algorithms. Different from the mutation operator of evolutionary algorithms, it is strictly deterministic and returns a set instead of single individual. Applying it to the same element g values will thus always

[1] http://en.wikipedia.org/wiki/State_space_search [accessed 2007-08-06]

yield the same set G. We can consider expand(g) to return the set of all possible results that we could obtain with a unary search operation like mutation.

The realization of expand has severe impact on the performance of search algorithms. An efficient implementation, for example, should not include states that have already been visited in the returned set. If the same elements are returned, the same solution candidates and all of their children will be evaluated multiple times, which would be useless and time consuming. Another problem occurs if there are two elements $g_1, g_2 \in \mathbb{G}$ with $g_1 \in \text{expand}(g_2)$ and $g_2 \in \text{expand}(g_1)$ exist. Then, the search would get trapped in an endless loop. Thus, visiting a genotype twice should always be avoided. Often, it is possible to design the search operations in a way preventing this from the start. Otherwise, tabu lists should be used, as done in the previously discussed Tabu Search algorithm (see Chapter 14 on page 273).

Since we want to keep our algorithm definitions as general as possible, we will keep the notation of individuals p that encompass a genotype $p.g \in \mathbb{G}$ and a phenotype $p.x \in \mathbb{X}$. Therefore, we need to an expansion operator that returns a set of individuals P rather than a set G of elements of the search space. We therefore define the operation "expandToInd" in Algorithm 17.1.

Algorithm 17.1: $P \longleftarrow \text{expandToInd}(g)$

Input: g: the element of the search space to be expanded
Data: i: a counter variable
Data: p: an individual record
Output: P: the list of individual records resulting from the expansion

```
1 begin
2     G ⟵ expand(g)
3     P ⟵ ()
4     for i ⟵ 0 up to len(G) − 1 do
5         p.g ⟵ G[i]
          // Implicitly: p⋆.x ⟵ gpm(p⋆.g)
6         P ⟵ addListItem(P, p)
7     return P
8 end
```

For all state space search strategies, we can define four criteria that tell if they are suitable for a given problem or not.

1. *Completeness.* Does the search algorithm guarantee to find a solution (given that there exists one)? (Do not mix up with the completeness of search operations specified in Definition 1.27 on page 44.)
2. *Time Consumption.* How much time will the strategy need to find a solution?
3. *Memory Consumption.* How much memory will the algorithm need to store intermediate steps? Together with time consumption this property is closely related to complexity theory, as discussed in Section 30.1.3 on page 550.
4. *Optimiality.* Will the algorithm find an optimal solution if there exist multiple correct solutions?

Search algorithms can further be classified according to the following definitions:

Definition 17.3 (Local Search). Local search algorithms work on a single current state (instead of multiple solution candidates) and generally transcend only to neighbors of the current state [1780].

Local search algorithms are not systematic but have two major advantages: They use very little memory (normally only a constant amount) and are often able to find solutions

in large or infinite search spaces. These advantages come, of course, with large trade-offs in processing time.

We can consider local searches as special case of global searches which incorporate larger populations. If the previously mentioned requirements are met, global search, in turn, can be regarded as a special case of global optimization algorithms.

17.2 General Information

17.2.1 Areas Of Application

Some example areas of application of State space search are:

Application	References
Networking and Communication	[1918, 1637, 1952] see Section 23.2 on page 401

17.2.2 Books

Some books about (or including significant information about) State space search are:

Russell and Norvig [1780]: *Artificial Intelligence: A Modern Approach*
Bednorz [166]: *Advances in Greedy Algorithms*

17.3 Uninformed Search

The optimization algorithms that we have considered up to now always require some sort of utility measure. These measures, the objective functions, are normally real-valued and allow us to make fine distinctions between different individuals. Under some circumstances, however, only a criterion isGoal is given as a form of Boolean objective function. The methods previously discussed will then not be able to descend a gradient anymore and degenerate to random walks (see Section 17.3.5 on page 294).

Here, uninformed search strategies[2] are a viable alternative since they do not require or take into consideration any knowledge about the special nature of the problem (apart from the knowledge represented by the expand operation, of course). Such algorithms are very general and can be applied to a wide variety of problems. Their common drawback is that search spaces are often very large. Without the incorporation of information in form of heuristic functions, for example, the search may take very long and quickly becomes infeasible [1780, 446, 569].

17.3.1 Breadth-First Search

In breadth-first search[3] (BFS), we start with expanding the root solution candidate. Then all of the states derived from this expansion are visited, and then all their children, and so on. In general, we first expand all states in depth d before considering any state in depth $d + 1$.

It is complete, since it will always find a solution if there exists one. If so, it will also find the solution that can be reached from the root state with the least expansion steps. Hence,

Algorithm 17.2: $X^\star \longleftarrow \text{bfs}(r, \text{isGoal})$

Input: r: the root individual to start the expansion at
Input: isGoal: an operator that checks whether a state is a goal state or not
Data: p: the state currently processed
Data: P: the queue of states to explore
Output: $X^\star$: the solution states found, or $\emptyset$

```
1 begin
2     P ⟵ createList(1, r)
3     while len(P) > 0 do
4         p ⟵ deleteListItem(P, 0)
5         if isGoal(p.x) then return {p.x}
6         P ⟵ appendList(P, expandToInd(p.g))
7     return ∅
8 end
```

if the number of expansion steps needed from the origin to a state is a measure for the costs, BFS is also optimal.

Algorithm 17.2 illustrates how breadth-first search works. The algorithm is initialized with a root state $r \in \mathbb{G}$ which marks the starting point of the search. BFS uses a state list P which initially only contains this root individual. In a loop, the first element p is removed from this list. If the goal predicate isGoal($p.x$) evaluates to `true`, $p.x$ is a goal state and we can return a set $X^\star = \{p.x\}$ containing it as the solution. Otherwise, we expand $p.g$ and append the newly found individuals to the end of queue P. If no solution can be found, this process will continue until the whole accessible search space has been enumerated and P becomes empty. Then, an empty set is returned in place of $X^\star$, because there is no element x in the (accessible part of the) problem space $\mathbb{X}$ for which isGoal(x) becomes `true`.

In order to examine the space and time complexity of BFS, we assume a hypothetical state space $\mathbb{G}_h$ where the expansion of each state $g \in \mathbb{G}_h$ will return a set of $\text{len}(\text{expand}(g)) = b$ new states. In depth 0, we only have one state, the root state r. In depth 1, there are b states, and in depth 2 we can expand each of them to again, b new states which makes b^2, and so on. Up to depth d we have a number of states total of

$$1 + b + b^2 + \cdots + b^d = \frac{b^{d+1} + 1}{b - 1} \in \mathbf{O}\left(b^d\right) \tag{17.3}$$

We have both, a space and time complexity from $\mathbf{O}\left(b^d\right)$. In the worst case, all nodes in depth d need to be stored, in the best case only those of depth $d - 1$.

17.3.2 Depth-First Search

Depth-first search[4] (DFS) is very similar to BFS. From the algorithmic point of view, the only difference that it uses a stack instead of a queue as internal storage for states (compare line 4 in Algorithm 17.3 with line 4 in Algorithm 17.2). Here, always the last state element of the set of expanded states is considered next. Thus, instead of searching level for level in the breath as BFS does, DFS searches in depth (which – believe it or not – is the reason for its name). DFS advances in depth until the current state cannot further be expanded, i. e., $\text{expand}(p.g) = ()$. Then, the search steps again up one level. If the whole search space has been browsed and no solution is found, $\emptyset$ is returned.

The memory consumption of the DFS is linear, because in depth d, at most $d * b$ states are held in memory. If we assume a maximum depth m, the time complexity is b^m in the

[2] http://en.wikipedia.org/wiki/Uninformed_search [accessed 2007-08-07]
[3] http://en.wikipedia.org/wiki/Breadth-first_search [accessed 2007-08-06]
[4] http://en.wikipedia.org/wiki/Depth-first_search [accessed 2007-08-06]

Algorithm 17.3: $X^\star \longleftarrow \text{dfs}(r, \text{isGoal})$

Input: r: the root individual to start the expansion at
Input: isGoal: an operator that checks whether a state is a goal state or not
Data: p: the state currently processed
Data: P: the queue of states to explore
Output: $X^\star$: the solution states found, or $\emptyset$

```
1 begin
2     P ⟵ createList(1, r)
3     while len(P) > 0 do
4         p ⟵ deleteListItem(P, len(P) − 1)
5         if isGoal(p.x) then return {p.x}
6         P ⟵ appendList(P, expandToInd(p.g))
7     return ∅
8 end
```

worst case where the solution is the last child state in the path explored the last. If m is very large or infinite, a DFS may take very long to discover a solution or will not find it at all, since it may get stuck in a "wrong" branch of the state space. Hence, depth first search is neither complete nor optimal.

17.3.3 Depth-limited Search

The depth-limited search[5] [1780] is a depth-first search that only proceeds up to a given maximum depth d. In other words, it does not examine solution candidates that are more than d expand-operations away from the root state r, as outlined in Algorithm 17.4 in a recursive form. Analogously to the plain depth first search, the time complexity now becomes b^d and the memory complexity is in $\mathbf{O}(b * d)$. Of course, the depth-limited search can neither be complete nor optimal. If a maximum depth of the possible solutions however known, it may be sufficient.

Algorithm 17.4: $X^\star \longleftarrow \text{dl\_dfs}(r, \text{isGoal}, d)$

Input: r: the root individual to start the expansion at
Input: isGoal: an operator that checks whether a state is a goal state or not
Input: d: the (remaining) allowed depth steps
Data: p: the state currently processed
Output: $X^\star$: the solution states found, or $\emptyset$

```
1 begin
2     if isGoal(r.x) then return {r.x}
3     if d > 0 then
4         foreach p ∈ expandToIndr.g do
5             X⋆ ⟵ dl_dfs(p, isGoal, d − 1)
6             if len(X⋆) > 0 then return X⋆
7     return ∅
8 end
```

[5] http://en.wikipedia.org/wiki/Depth-limited_search [accessed 2007-08-07]

17.3.4 Iterative Deepening Depth-First Search

The iterative deepening depth-first search[6] (IDDFS, [1780]), defined in Algorithm 17.5, iteratively runs a depth-limited DFS with stepwise increasing maximum depths d. In each iteration, it visits the states in the according to the depth-first search. Since the maximum depth is always incremented by one, one new level in terms means of distance in expand-operations from the root is explored in each iteration. This effectively leads to some form of breadth-first search.

IDDFS thus unites the advantages of BFS and DFS: It is complete and optimal, but only has a linearly rising memory consumption in $\mathbf{O}(d * b)$. The time consumption, of course, is still in $\mathbf{O}\left(b^d\right)$. IDDFS is the best uninformed search strategy and can be applied to large search spaces with unknown depth of the solution.

The Algorithm 17.5 is intended for infinitely large search spaces. In real systems, there is a maximum $\hat{d}$ after which the whole space would be explored and the algorithm should return $\emptyset$ if no solution was found.

Algorithm 17.5: $X^\star \longleftarrow \text{iddfs}(r, \text{isGoal})$

Input: r: the root individual to start the expansion at
Input: isGoal: an operator that checks whether a state is a goal state or not
Data: d: the current depth limit
Output: $X^\star$: the solution states found, or $\emptyset$

```
1 begin
2 |  d ⟵ 0
3 |  repeat
4 |  |  X* ⟵ dl_dfs(r, isGoal, d)
5 |  |  d ⟵ d + 1
6 |  until len(X*) > 0
7 |  return X*
8 end
```

17.3.5 Random Walks

Random walks[7] (sometimes also called drunkard's walk) are a special case of undirected, local search. Instead of proceeding according to some schema like depth-first or breadth-first, the next solution candidate to be explored is always generated randomly from the currently investigated one. [974, 649] Under some special circumstances, random walks can be the search algorithms of choice. This for instance the case in

1. If we encounter a state explosion because there are too many states to which we can possible transcend to and methods like breadth-first search or iterative deepening depth-first search cannot be applied because they would consume too much memory.
2. In certain cases of online search it is not possible to apply systematic approaches like BFS or DFS. If the environment, for instance, is only partially observable and each state transition represents an immediate interaction with this environment, we are maybe not able to navigate to past states again. One example for such a case is discussed in the work of Skubch [1897, 1898] about reasoning agents.

[6] http://en.wikipedia.org/wiki/IDDFS [accessed 2007-08-08]
[7] http://en.wikipedia.org/wiki/Random_walk [accessed 2007-11-27]

Random walks are often used in optimization theory for determining features of a fitness landscape. Measures that can be derived mathematically from a walk model include estimates for the number of steps needed until a certain configuration is found, the chance to find a certain configuration at all, and the average difference of the objective values of two consecutive populations. From practically running random walks, some information about the search space can be extracted. Skubch [1897, 1898], for instance, uses the number of encounters of certain state transition during the walk in order to successively build a heuristic function.

17.4 Informed Search

In an informed search[8], a heuristic function h helps to decide which states are to be expanded next. If the heuristic is good, informed search algorithms may dramatically outperform uninformed strategies [1407, 1711, 1626].

As specified in Definition 1.2 on page 22, heuristic functions are problem domain dependent. In the context of an informed search, a heuristic function $h : \mathbb{X} \mapsto \mathbb{R}^+$ maps the states in the state space $\mathbb{G}$ to the positive real numbers $\mathbb{R}^+$. We further define that all heuristics will be zero for the elements which are part of the solution space $\mathbb{S}$, i. e.,

$$\forall x \in \mathbb{X} : \text{isGoal}(x) \Rightarrow h(x) = 0 \; \forall \textit{heuristics } h : \mathbb{X} \mapsto \mathbb{R}^+ \tag{17.4}$$

There are two possible meanings of the values returned by a heuristic function h:

1. In the above sense, the value of a heuristic function $h(p.x)$ for an individual p is the higher, the more expand-steps $p.g$ is *probably* (or *approximately*) away from finding a valid solution. Hence, the heuristic function represents the distance of an individual to a solution in *solution space*.
2. The heuristic function can also represent an objective function in some way. Suppose that we know the minimal value $\breve{y}$ for an objective function f or at least a value from where on all solutions are feasible. If this is the case, we could set $h(p.x) = \max\{0, f(p.x) - \breve{y}\}$, assuming that f is subject to minimization. Now the value of heuristic function will be the smaller, the closer an individual is to a possible correct solution and Equation 17.4 still holds. In other words, a heuristic function may also represent the distance to a solution in *objective space*.

Of course, both meanings are often closely related since states that are close to each other in problem space are probably also close to each other in objective space (the opposite does not necessarily hold).

A best-first search[9] [1626] is a search algorithm that incorporates such a heuristic function h in a way which ensures that promising individuals p with low estimation values $h(p.x)$ are evaluated before other states q that receive a higher values $h(q.x) > h(p.x)$.

17.4.1 Greedy Search

A greedy search[10] is a best-first search where the currently known solution candidate with the lowest heuristic value is investigated next. The greedy algorithm internal sorts the list of currently known states in *descending* order according to a heuristic function h. Thus, the elements with the best (lowest) heuristic value will be at the end of the list, which then

[8] http://en.wikipedia.org/wiki/Search_algorithms#Informed_search [accessed 2007-08-08]
[9] http://en.wikipedia.org/wiki/Best-first_search [accessed 2007-09-25]
[10] http://en.wikipedia.org/wiki/Greedy_search [accessed 2007-08-08]

can be used as a stack. The greedy search as specified in Algorithm 17.6 now works like a depth-first search on this stack and thus, also shares most of the properties of the DFS. It is neither complete nor optimal and its worst case time consumption is b^m. On the other hand, like breadth-first search, its worst-case memory consumption is also b^m.

Algorithm 17.6: $X^\star \longleftarrow \text{greedySearch}(r, \text{isGoal}, h)$

Input: r: the root individual to start the expansion at
Input: isGoal: an operator that checks whether a state is a goal state or not
Input: h: the heuristic function
Data: p: the state currently processed
Data: P: the queue of states to explore
Output: $X^\star$: the solution states found, or $\emptyset$

```
1 begin
2     P ⟵ createList(1, r)
3     while len(P) > 0 do
4         P ⟵ sortList_d(P, cmp(p_1, p_2) ≡ h(p_1.x) − h(p_2.x))
5         p ⟵ deleteListItem(P, len(P) − 1)
6         if isGoal(p.x) then return {p.x}
7         P ⟵ appendList(P, expandToInd(p.g))
8     return ∅
9 end
```

17.4.2 A* search

In A* search[11] is a best-first search that uses a estimation function $h^\star : \mathbb{X} \mapsto \mathbb{R}^+$ which is the sum of a heuristic function $h(x)$ that estimates the costs needed to get from x to a valid solution and a function $g(x)$ that computes the costs that are needed to get to x.

$$h^\star(x) = g(x) + h(x) \tag{17.5}$$

A* search proceeds exactly like the greedy search outlined in Algorithm 17.6, if $h^\star$ is used instead of plain h. An A* search will definitely find a solution if there exists one, i. e., it is complete.

Definition 17.4 (Admissible Heuristic Function). A heuristic function $h : \mathbb{X} \mapsto \mathbb{R}^+$ is admissible if it never overestimates the minimal costs for reaching a goal state.

Definition 17.5 (Monotonic Heuristic Function). A heuristic function $h : \mathbb{X} \mapsto \mathbb{R}^+$ is monotonic[12] if it never overestimates the costs for getting from one state to its successor.

$$h(p.x) \leq g(q.x) - g(p.x) + h(q.x) \;\; \forall q.g \in \text{expand}(p.g) \tag{17.6}$$

An A* search is optimal if the heuristic function h used is admissible. Optimal in this case means that there exists no search algorithm that can find the same solution as the A* search needing fewer expansion steps if using the same heuristic. If we implement expand in a way which prevents that a state is visited more than once, h also needs to be monotone in order for the search to be optimal.

[11] http://en.wikipedia.org/wiki/A%2A_search [accessed 2007-08-09]
[12] see Definition 27.28 on page 463

17.4.3 Adaptive Walks

An *adaptive walk* is a theoretical optimization method which, like a random walk, usually works on a population of size 1. It starts at a random location in the search space and proceeds by changing (or mutating) its single solution candidate. For this modification, three methods are available:

1. *One-mutant change*: The optimization process chooses a single new individual from the set of "one-mutant change" neighbors, i. e., a neighboring individual differing from the current solution candidate in only one property. If the new individual is better, it replaces its ancestor, otherwise it is discarded.
2. *Greedy dynamics*: The optimization process chooses a single new individual from the set of "one-mutant change" neighbors. If it is not better than the current solution candidate, the search continues until a better one has been found or all neighbors have been enumerated. The major difference to the previous form is the number of steps that are needed per improvement.
3. *Fitter Dynamics*: The optimization process enumerates all one-mutant neighbors of the current solution candidate and transcends to the best one.

From these elaborations, it becomes clear that adaptive walks are very similar to hill climbing and Random Optimization. The major difference is that an adaptive walk is a theoretical construct that, very much like random walks, helps us to determine properties of fitness landscapes whereas the other two are practical realizations of optimization algorithms.

Adaptive walks are a very common construct in evolutionary biology. Biological populations are running for a very long time and so their genetic compositions are assumed to be relatively converged [807, 44]. The dynamics of such populations in near-equilibrium states with low mutation rates can be approximated with one-mutant adaptive walks [1903, 807, 44].

18

Parallelization and Distribution

As already stated many times, global optimization problems are often computational intense. Up until now, we have only explored the structure and functionality of optimization algorithms without paying attention to their potential of parallelization or even distribution[1].

Roughly speaking, parallelization[2] means to search for pieces of code that can potentially run concurrently and letting them execute by different processors [1984, 184]. Take painting a fence for example. Here, the overall progress will be much faster if more than one painter applies the color to the wood. Distribution[3] is a special case of parallelization where the different processors are located on different machines in a network [146, 2010]. Imagine that each fence-painter would take a piece of the fence to his workshop where he can use a special airbrush which can color the whole piece at once. Distribution comes with the trade-off of additional communication costs for transporting the data, but has the benefit that it is more generic. At the current time, off-the-shelf PCs usually have not more than two CPUs. This limits the benefit of local parallelization. We can, however, connect arbitrarily many of such computers in a network for distributed processing.

18.1 Analysis

In order to understand which parts of an optimization algorithm can be parallelized, the first step is an analysis. We will do such an analysis for evolutionary algorithms as example for population-based optimizers.[4]. The parallelization and distribution of evolutionary algorithms has long been a subject to study and has been discussed by multiple researchers like Alba and Tomassini [34], Cant'u-Paz [329, 330], Tan et al. [2003], Tanese [2007], Mühlenbein [1478], and Bollini and Piastra [244].

There are two components of evolutionary algorithms whose performance potentially can remarkably be increased by parallelization: the evaluation and the reproduction stages. As sketched in Figure 18.1, evaluation is a per-individual process. The values of the objective functions are determined for each solution candidate independently from the rest of the population. Evaluating the individuals often involves complicated simulations and calculations and is thus usually the most time-consuming part of evolutionary algorithms.

During the fitness assignment process, it is normally required to compare solution candidates with the rest of the population, to compute special sets of individuals, or to update some data structures. This makes it very hard for parallelization to provide any speedup.

[1] Section 30.2 on page 553 gives a detailed introduction into distributed algorithms, their advantages and drawbacks.

[2] http://en.wikipedia.org/wiki/Parallelization [accessed 2007-07-03]

[3] http://en.wikipedia.org/wiki/Distributed_computing [accessed 2007-11-30]

[4] In Section 2.1.3 on page 98 you can find the basic evolutionary algorithm.

The selection phase may or may not require access to certain subsets of the population or data updates. Whether parallelization is possible or is beneficial thus depends on the selection scheme applied.

The reproduction phase, on the other hand, can very easily be parallelized. It involves creating a new individual by using (but not altering) the information from n existing ones, where $n = 0$ corresponds to the creation operation, $n = 1$ resembles mutation, and $n = 2$ means recombination. Thus, the making of each new genotype is an independent task.

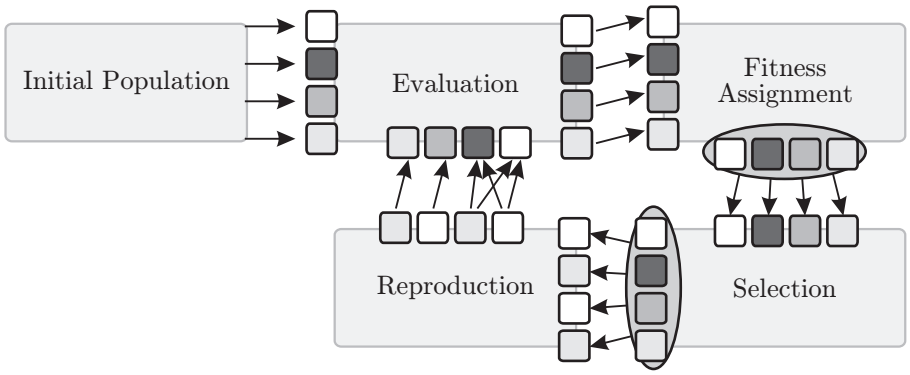


Figure 18.1: Parallelization potential in evolutionary algorithm.

Despite running an evolutionary algorithm in single a thread[5] of execution (see Figure 18.2), our analysis has shown that makes sense to have at least the evaluation and reproduction phase executed in parallel as illustrated in Figure 18.3. Usually, the population is larger than the number of available CPUs[6], so one thread could be created per processors that consecutively pulls individuals out of a queue and processes them. This approach even yields performance gains on off-the-shelf personal computers since these nowadays at least come with hyper-threading[7] technology [2064, 1564] or even dual-core[8] CPUs [1165, 1891].

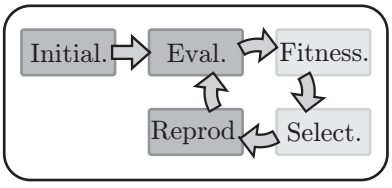


Figure 18.2: A sequentially proceeding evolutionary algorithm.

[5] http://en.wikipedia.org/wiki/Thread_%28computer_science%29 [accessed 2007-07-03]
[6] http://en.wikipedia.org/wiki/Cpu [accessed 2007-07-03]
[7] http://en.wikipedia.org/wiki/Hyper-threading [accessed 2007-07-03]
[8] http://en.wikipedia.org/wiki/Dual-core [accessed 2007-07-03]

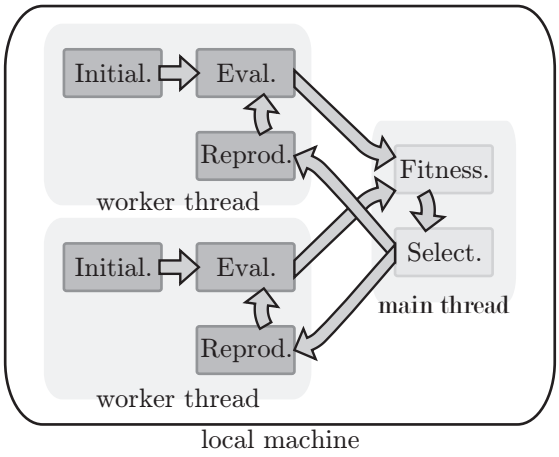


Figure 18.3: A parallel evolutionary algorithm with two worker threads.

Cant'u-Paz [328] divides parallel evolutionary algorithms into two main classes:

1. In *globally* parallelized EAs, each individual in the population can (possibly) always mate with any other.
2. In *coarse grained* approaches, the population is divided into several sub-populations where mating inside a sub-population is unrestricted but mating between individuals of different sub-populations may only take place occasionally according to some rule.

In ancient Greece, a *deme* was a district or township inhabited by a group that formed an independent community. They were the basic units of government in Attica as remodeled by Cleisthenes around 500 BC. In biology, a deme is a locally interbreeding group within a geographic population.

Definition 18.1 (Deme). In evolutionary algorithms, a *deme* is a distinct sub-population.

In the following, we are going to discuss some of the different parallelization methods from the viewpoint of distribution because of its greater generality.

18.2 Distribution

The distribution of an algorithm only pays off if the delay induced by the transmissions necessary for data exchange is much smaller than the time saved by distributing the computational load. Thus, in some cases distributing of optimization is useless. If searching for the root of a mathematical function for example, transmitting the parameter vector x to another computer will take much longer than computing the function $f(x)$ locally. In this section, we will investigate some basic means to distribute evolutionary algorithms that can as well as be applied to other optimization methods as outlined by Weise and Geihs [2177].

18.2.1 Client-Server

If the evaluation of the objective functions is time consuming, the easiest approach to distribute and evolutionary algorithm is the client-server scheme (also called master-slave).[9] Figure 18.4 illustrates how we can make use of this very basic, global distribution scheme. Here, the servers (slaves) receive the single tasks, process them, and return the results.

[9] A general discussing concerning the client-server architecture can be found in Section 30.2.2 on page 556

Such a task can, for example, be the reproduction of one or two individuals and the subsequent determination of the objective values of the offspring. The client (or master) just needs to distribute the parent individuals to the servers and receives their fully evaluated offspring in return. These offspring are then integrated into the population, where fitness assignment and selection is performed. Client-server-based distribution approaches for evolutionary algorithms have been discussed and tested by Van Veldhuizen et al. [2102], Xu et al. [2271], Dubreuil et al. [604] and were realized in general-purpose software packages by Cahon et al. [323], Luke et al. [1327], and Weise and Geihs [2177].

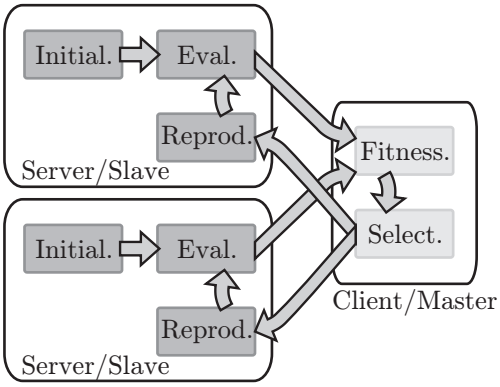


Figure 18.4: An EA distributed according to the client-server approach.

One practical realization of this approach can be to use a queue where all the selected individuals are pushed into as mating pool. Each server in the network is then represented by a thread on the client side. Such a thread pulls individuals from the queue, sends them to its corresponding server, and waits for the result to be returned. It places the individuals it receives into the new population and then starts over again. Servers may possess multiple processors, which can be taken into account by representing them by an appropriate number of threads.

18.2.2 Island Model

Under some circumstances, the client-server approach may not be optimal, especially if

1. Processing of the tasks is fast relatively to the amount of time needed for the data exchange between the client and the server. In other words, if messages that have to be exchanged travel longer than the work would take if performed locally, the client-server method would actually slow down the system.
2. Populations are required that cannot be held completely in the memory of a single computer. This can be the case either if the solution candidates are complex and memory consuming or the nature of the problem requires large populations.

In such cases, we can again learn from nature. Until now we only have imitated evolution on one large continent. All individuals in the population compete with each other and there are no barriers between the solution candidates. In reality, there occur obstacles like mountain ranges or oceans, separating parts of the population and creating isolated subpopulations. Another example for such a scenario is an archipelago like the Galapagos islands where Darwin [485], the father of the evolution theory, performed his studies. On the single islands, different species can evolve independently. From time to time, a few individuals from one isle *migrate* another one, maybe by traveling on a tree trunk over the water or by been blown there by a storm. If they are fit enough in their new environment, they can

compete with the local population and survive. Otherwise, they will be extruded by the native residents of the habitat. This way, the islands manage an approximately equal level of fitness of their individuals, while still preserving a large amount of diversity.

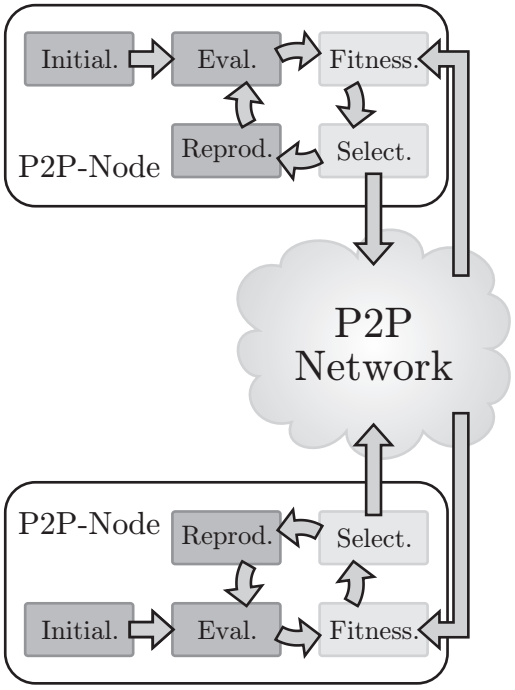


Figure 18.5: An evolutionary algorithm distributed in a P2P network.

We can easily copy this natural role model in evolutionary algorithms by using multiple sub-populations (demes) as discussed by Cohoon et al. [426], Martin et al. [1365], Skolicki and De Jong [1896, 1895], Gorges-Schleuter [836], Tanese [2007], and Toshine et al. [2048] and also realized in various software packages such as those created by Whitley and Starkweather [2213], Paechter et al. [1597], Tan et al. [2003], Arenas et al. [83], Chong and Langdon [398], Luke et al. [1327], Cahon et al. [323], and Weise and Geihs [2177]. Distributing the demes on n different nodes in a network of computers, each representing one island, is maybe the most popular form of coarse grained parallelization. Hence, both disadvantages of the original master/slave approach are circumvented: Communication between nodes is only needed when individuals *migrate* between them. This communication can be performed asynchronously to the n independently running evolutionary algorithms and does not slow down their performance. The migration rule can furthermore be chosen in a way that reduces the network traffic. By dividing the population, the number of solution candidates to be held on single machines also decreases, which helps to mitigate the memory consumption problem.

The island model can be realized by peer-to-peer networks[10] where each node runs an independent evolution, as illustrated in Figure 18.5. Here, we have modified the selection phase which now returns some additional individuals to be transmitted to another node in the system. Depending on the optimization problems, solution candidates migrating over the network can either enter the fitness assignment process on the receiving machine directly or may take part in the evaluation process first. If the latter is the case, different objective functions can applied on different nodes.

[10] P2P networks are discussed in Section 30.2.2 on page 557.

Driving this thought further, one will recognize that the peer-to-peer approach inherently allows mixing of different optimization technologies, as outlined by Weise and Geihs [2177]. On one node, for instance, the SPEA2-algorithm (see **??** on page ??) can be performed, whereas another node could optimize according to plain hill climbing as described in Chapter 10 on page 253. Such a system, illustrated in Figure 18.6, has a striking advantage. From the No Free Lunch Theorem discussed in Section 1.4.10 on page 76, we know that for optimization algorithms perform differently for different problems. If the problem is unimodal, i. e., has exactly one global optimum and no local optima, a hill climbing approach will outperform any other technique since it directly converges to this optimum. If the fitness landscape is rugged, one the other hand, methods like SPEA2 which have a very balanced exploration/exploitation proportion are able to yield better results and hill climbing algorithms will get stuck to local optima. In most cases, it is not possible to know beforehand which optimization strategy will perform best. Furthermore, the best approach may even change while the optimization proceeds. If a new, better individual evolves, i. e., a new optimum is approached, hill climbing will be fast in developing this solution candidate further until its best form is found, i. e., the bottom of the local optimum is reached. In other phases, an exploration of the solution space may be required since all known local optima have been tracked. A technology like Ant Colony Optimization could now come into play. A heterogeneous mixture of these algorithms that exchanges individuals from time to time will retain the good properties of the single algorithms and, in many cases, outperform a homogeneous search [1275, 1023, 24, 95, 2283]. Just remember how our discussion of Memetic Algorithms in Chapter 15 on page 277.

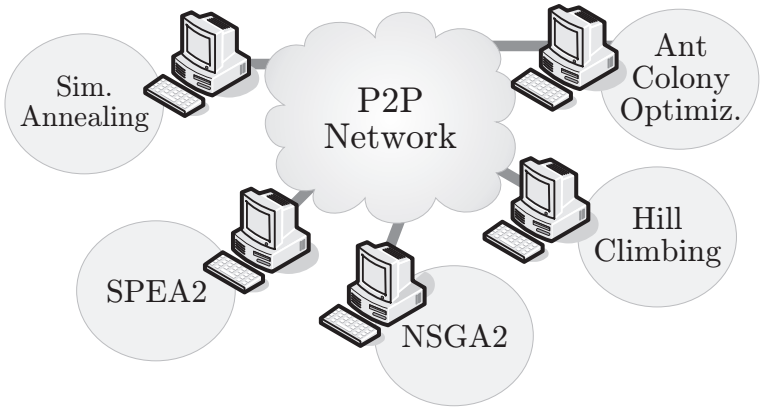


Figure 18.6: An example for a heterogeneous search.

The island model can also be applied locally by simply using disjoint local populations. Although this would not bring a performance gain, it could improve the convergence behavior of the optimization algorithm. Spieth et al. [1943], for instance, argue that the island model can be used to preserve the solution diversity. By doing so it decreases the probability of premature convergence (see Section 1.4.2 on page 58).

Broadcast-Distributed Parallel Evolutionary Algorithm

The Broadcast-Distributed Parallel Evolutionary Algorithm (BDP) defined by Johnson et al. [1060] extends the island model for large networks of wirelessly connected, resource-restricted devices such as sensor networks, amorphous and paintable computing systems. In the BDP, each node carries a separate population from which one individual is selected after each generation. This individual is broadcasted to the neighbors of the node. Every time a node

receives an individual, it appends it to an internal mate list. Whenever the length of this list exceeds a certain threshold, selection and subsequent crossover is performed on the joint set of the population and the individuals in the mate list.

18.2.3 Mixed Distribution

Of course, we can combine the both distribution approaches previously discussed by having a peer-to-peer network that also contains client-server systems, as sketched in Figure 18.7. Such a system will be especially powerful if we need large populations of individuals that take long to evaluate. Then, the single nodes in the peer-to-peer network together provide a larger virtual population, while speeding up their local evolutions by distributing the computational load to multiple servers.

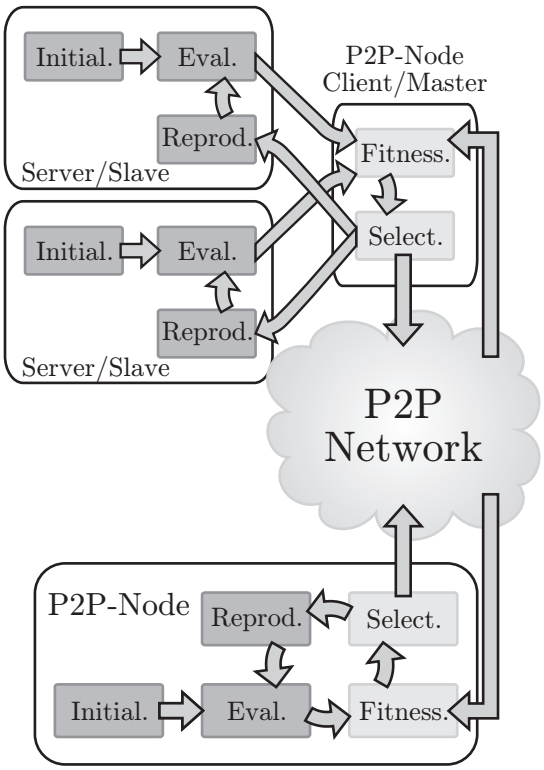


Figure 18.7: A mixed distributed evolutionary algorithms.

18.3 Cellular Genetic Algorithms

Cellular Genetic Algorithms [33] are a special family of parallelization models for genetic algorithms which has been studied by various researchers such as Whitley [2212, 2208], Manderick and Spiessens [1353], Hillis [928], and Davidor [490]. A good understanding of this model can be reached by starting with the basic architecture of the cellular system as described by Whitley [2208].

Assume we have a matrix m of $N \times N$ cells. Each cell has a processor and holds one individual of the population. It can communicate with its right, left, top, and bottom neighbor. Cells at the edge of the matrix are wired with cells in the same column/row at the

opposite edge. The cell $m_{I,j}$ can thus communicate with $m_{(i+1) \bmod N,j}$, $m_{(i-1) \bmod N,j}$, $m_{i,(j+1) \bmod N}$, and $m_{i,(j-1) \bmod N}$. It is also possible to extend this neighborhood to all cells in a given Manhattan distance, but let us stick with the easiest case.

Each cell can evaluate the individual it locally holds. For creating offspring, it can either mutate this individual or recombine it with one selected from of the four solution candidates on its neighbors. At the beginning, the matrix is initialized with random individuals. After some time, the spatial restriction of mating leads to the occurrence of local neighborhoods with similar solution candidates denoting local optima. These hoods begin to grow until they touch each other. Then, the regions better optima will "consume" worse ones and reduce the overall diversity.

Although there are no fixed mating restrictions like in the island model, regions that are about twenty or so moves away will virtually not influence each other. We can consider groups of cells that distant as separate sub-populations. This form of separation is called *isolation by distance* – again, a term that originally stems from biology (coined by Wright [2261]). [2208, 432, 1478, 836] For observing such effects, it is said that a certain minimum of cells is required – at least about 1000 according to Whitley [2208].

19

Maintaining the Optimal Set

Most multi-objective optimization algorithms return a set of optimal solutions $X^\star$ instead of a single individual $x^\star$. Many optimization techniques also internally keep track of the set of best solution candidates encountered during the search process. In Simulated Annealing, for instance, it is quite possible to discover an optimal element $\mathbf{x}^\star$ and subsequently depart from it to a local optimum $x_l^\star$. Therefore, optimizers normally carry a list of the non-prevailed solution candidates ever visited with them.

In scenarios where the search space $\mathbb{G}$ differs from the problem space $\mathbb{X}$ it often makes more sense to store the list of optimal individual records $P^\star$ instead of just keeping the optimal phenotypes $x^\star$. Since the elements of the search space are no longer required at the end of the optimization process, we define the simple operation extractPhenotypes which extracts them from a set of individuals P.

$$\forall x \in \text{extractPhenotypes}(P) \Rightarrow \exists p \in P : x = p.x \tag{19.1}$$

19.1 Updating the Optimal Set

Whenever a new individual p is created, the set of optimal individuals $P^\star$ may change. It is possible that the new solution candidate must be included in the optimal set or even prevails some of the phenotypes already contained therein which then must be removed.

Definition 19.1 (updateOptimalSet**).** The function updateOptimalSet updates a set of optimal elements $P^\star_{old}$ with the new solution candidate $p_{new}.x$. It uses implicit knowledge of the prevalence relation $\succ$ and the corresponding comparator function cmp_F.

$$\begin{aligned} &P^\star_{new} = \text{updateOptimalSet}(P^\star_{old}, p_{new})\,, \\ &P^\star_{old}, P^\star_{new} \subseteq \mathbb{G} \times \mathbb{X},\ p_{new} \in \mathbb{G} \times \mathbb{X} : \\ &\quad \forall p_1 \in P^\star_{old}\ \nexists p_2 \in P^\star_{old} : \quad p_2.x \succ p_1.x \quad \Rightarrow\ P^\star_{new} \subseteq P^\star_{old} \cup \{p_{new}\} \wedge \\ &\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \forall p_1 \in P^\star_{new}\ \nexists p_2 \in P^\star_{new} : \quad p_2.x \succ p_1.x \end{aligned} \tag{19.2}$$

We define two equivalent approaches in Algorithm 19.1 and Algorithm 19.2 which perform the necessary operations. Algorithm 19.1 creates a new, empty optimal set and successively inserts optimal elements whereas Algorithm 19.2 removes all elements which are prevailed by the new individual p_{new} from the old optimal set $P^\star_{old}$.

Especially in the case of evolutionary algorithms, not a single new element is created in each generation but a set P. Let us define the operation updateOptimalSetN for this purpose. This operation can easily be realized by iteratively applying updateOptimalSet, as shown in Algorithm 19.3.

Algorithm 19.1: $P^\star_{new} \longleftarrow$ updateOptimalSet$(P^\star_{old}, p_{new})$

Input: $P^\star_{old}$: the optimal set as known before the creation of p_{new}
Input: p_{new}: a new individual to be checked
Output: $P^\star_{new}$: the optimal set updated with the knowledge of p_{new}

```
1 begin
2     P*_new ⟵ ∅
3     foreach p_old ∈ P*_old do
4         if ¬(p_new.x ≻ p_old.x) then
5             P*_new ⟵ P*_new ∪ {p_old}
6             if p_old.x ≻ p_new.x then return P*_old
7     return P*_new ∪ {p_new}
8 end
```

Algorithm 19.2: $P^\star_{new} \longleftarrow$ updateOptimalSet$(P^\star_{old}, p_{new})$ *(2nd Version)*

Input: $P^\star_{old}$: the optimal set as known before the creation of p_{new}
Input: p_{new}: a new individual to be checked
Output: $P^\star_{new}$: the optimal set updated with the knowledge of p_{new}

```
1 begin
2     P*_new ⟵ P*_old
3     foreach p_old ∈ P*_old do
4         if p_new.x ≻ p_old.x then
5             P*_new ⟵ P*_new \ {p_old}
6         else if p_old.x ≻ p_new.x then
7             return P*_old
8     return P*_new ∪ {p_new}
9 end
```

Algorithm 19.3: $P^\star_{new} \longleftarrow$ updateOptimalSetN$(P^\star_{old}, P)$

Input: $P^\star_{old}$: the old optimal set
Input: P: the set of new individuals to be checked for optimality
Data: p: an individual from P
Output: $P^\star_{new}$: the updated optimal set

```
1 begin
2     P*_new ⟵ P*_old
3     foreach p ∈ P do P*_new ⟵ updateOptimalSet(P*_new, p)
4     return P*_new
5 end
```

19.2 Obtaining Optimal Elements

The function updateOptimalSet helps an optimizer to build and maintain a list of optimal individuals. When the optimization process finishes, the extractPhenotypes can then be used to obtain the optimal elements of the problem space and return them to the user. However, not all optimization methods maintain an optimal set all the time. When they terminate, they have to extracts all optimal elements the set of individuals *Pop* currently known.

Definition 19.2 (extractOptimalSet**).** The function extractOptimalSet function extracts a set $P^\star$ of optimal (non-prevailed) individuals from any given set of individuals *Pop*.

$$\forall P^\star \subseteq Pop \subseteq \mathbb{G} \times \mathbb{X}, P^\star = \text{extractOptimalSet}(Pop) \Rightarrow \forall p_1 \in P^\star \nexists p_2 \in Pop : p_2.x \succ p_1.x \tag{19.3}$$

Algorithm 19.4 defines one possible realization of extractOptimalSet. By the way, this approach could also be used for updating an optimal set.

$$\text{updateOptimalSet}(P^\star_{old}, p_{new}) \equiv \text{extractOptimalSet}(P^\star_{old} \cup p_{new}) \tag{19.4}$$

Algorithm 19.4: $P^\star \longleftarrow$ extractOptimalSet(Pop)

Input: Pop: the list to extract the optimal individuals from
Data: p_{any}, p_{chk}: solution candidates tested for supremacy
Data: i, j: counter variables
Output: $P^\star$: the optimal subset extracted from Pop

```
1  begin
2      P* ⟵ Pop
3      for i ⟵ len(P*) − 1 down to 1 do
4          for j ⟵ i − 1 down to 0 do
5              if P*[i] ≻ P*[j] then
6                  P* ⟵ deleteListItem(P*, j)
7                  i ⟵ i − 1
8              else if P*[j] ≻ P*[i] then
9                  P* ⟵ deleteListItem(P*, i)
10     return listToSet(P*)
11 end
```

19.3 Pruning the Optimal Set

In some optimization problems, there may be very many if not infinite many optimal individuals. The set of $X^\star$ optimal solution candidates computed by the optimization algorithms, however, cannot grow infinitely because we only have limited memory. Therefore, we need to perform an action called *pruning* which reduces the size of the optimal set to a given limit k [1466, 1993, 427].

There exists a variety of possible pruning operations. Morse [1466] and Taboada and Coit [1992], for instance, suggest to use clustering algorithms[1] for this purpose. In principle, also any combination of the fitness assignment and selection schemes discussed in Chapter 2 would do. It is very important that the loss of generality during a pruning operation is minimized. Fieldsend et al. [667], for instance, point out that if the extreme elements of the optimal frontier are lost, the resulting set may only represent a small fraction of the optimal that could have been found without pruning. Instead of working on the set of optimal solution candidates $X^\star$, we again base our approach on a set of optimal individuals $P^\star$ and we define:

Definition 19.3 (pruneOptimalSet**).** The pruning operation pruneOptimalSet reduces the size of a set $P^\star_{old}$ of individuals to fit to a given upper boundary k.

$$\forall P^\star_{new} \subseteq P^\star_{old} \subseteq \mathbb{G} \times \mathbb{X},\ k \in \mathbb{N} : P^\star_{new} = \text{pruneOptimalSet}(P^\star_{old}, k) \Rightarrow |P^\star_{new}| \leq k \tag{19.5}$$

19.3.1 Pruning via Clustering

Algorithm 19.5 uses clustering to provide the functionality specified in this definition and thereby realizes the idea of Morse [1466] and Taboada and Coit [1992]. Basically, any given clustering algorithm could be used as replacement for cluster – see Chapter 29 on page 535 for more information on clustering.

[1] You can find a discussion of clustering algorithms in Section 29.2.

Algorithm 19.5: $P^\star_{new} \longleftarrow \text{pruneOptimalSet}_c(P^\star_{old}, k)$

Input: $P^\star_{old}$: the optimal set to be pruned
Input: k: the maximum size allowed for the optimal set ($k > 0$)
Input: [implicit] cluster: the clustering algorithm to be used
Input: [implicit] nucleus: the function used to determine the nuclei of the clusters
Data: B: the set of clusters obtained by the clustering algorithm
Data: b: a single cluster $b \in B$
Output: $P^\star_{new}$: the pruned optimal set

```
1 begin
    // obtain k clusters
2   B ⟵ cluster(P*_old)
3   P*_new ⟵ ∅
4   foreach b ∈ B do P*_new ⟵ P*_new ∪ nucleus(b)
5   return P*_new
6 end
```

19.3.2 Adaptive Grid Archiving

Let us discuss the adaptive grid archiving algorithm (AGA) as example for a more sophisticated approach to prune the optimal set. AGA has been introduced for the evolutionary algorithm PAES[2] by Knowles and Corne [1154] and uses the objective values (computed by the set of objective functions F) directly. Hence, it can treat the individuals as $|F|$-dimensional vectors where each dimension corresponds to one objective function $f \in F$. This $|F|$-dimensional objective space $\mathbb{Y}$ is divided in a grid with d divisions in each dimension. Its span in each dimension is defined by the corresponding minimum and maximum objective values. The individuals with the minimum/maximum values are always preserved. This circumvents the phenomenon of narrowing down the optimal set described by Fieldsend et al. [667] and distinguishes the AGA approach from clustering-based methods. Hence, it is not possible to define maximum optimal set sizes k which are smaller than $2|F|$. If individuals need to be removed from the set because it became too large, the AGA approach removes those that reside in regions which are the most crowded.

The original sources outline the algorithm basically with with descriptions and definitions. Here, we introduce a more or less trivial specification in Algorithm 19.6 on the facing page and Algorithm 19.7 on page 312. The function agaDivide is internally used to perform the grid division. It transforms the objective values of each individual to grid coordinates stored in the array *lst*. Furthermore, agaDivide also counts the number of individuals that reside in the same coordinates for each individual and makes it available in *cnt*. It ensures the preservation of border individuals by assigning a negative *cnt* value to them. This basically disables their later disposal by the pruning algorithm $\text{pruneOptimalSet}_{aga}$ since $\text{pruneOptimalSet}_{aga}$ deletes the individuals from the set $P^\star_{old}$ that have largest *cnt* values first.

[2] PAES is discussed in **??** on page ??

Algorithm 19.6: $(P_l, lst, cnt) \longleftarrow \text{agaDivide}(P_{old}, d)$

Input: P_{old}: the optimal set to be pruned
Input: d: the number of divisions to be performed per dimension
Input: [implicit] F: the set of objective functions
Data: i, j: counter variables
Data: $mini, maxi, mul$: temporary stores
Output: (P_l, lst, cnt): a tuple containing the list representation P_l of P_{old}, a list lst assigning grid coordinates to the elements of P_l and a list cnt containing the number of elements in those grid locations

```
1  begin
2      mini ⟵ createList(|F|, ∞)
3      maxi ⟵ createList(|F|, −∞)
4      for i ⟵ |F| − 1 down to 0 do
5          mini[i−1] ⟵ min {f_i(p.x) ∀p ∈ P_old}
6          maxi[i−1] ⟵ max {f_i(p.x) ∀p ∈ P_old}
7      mul ⟵ createList(|F|, 0)
8      for i ⟵ |F| − 1 down to 0 do
9          if maxi[i] ≠ mini[i] then
10             mul[i] ⟵ d / (maxi[i] − mini[i])
11         else
12             maxi[i] ⟵ maxi[i] + 1
13             mini[i] ⟵ mini[i] − 1
14     P_l ⟵ setToList(P_old)
15     lst ⟵ createList(len(P_l), ∅)
16     cnt ⟵ createList(len(P_l), 0)
17     for i ⟵ len(P_l) − 1 down to 0 do
18         lst[i] ⟵ createList(|F|, 0)
19         for j ⟵ 1 up to |F| do
20             if (f_j(P_l[i]) ≤ mini[j−1]) ∨ (f_j(P_l[i]) ≥ maxi[j−1]) then
21                 cnt[i] ⟵ cnt[i] − 2
22             lst[i][j−1] ⟵ ⌊(f_j(P_l[i]) − mini[j−1]) ∗ mul[j−1]⌋
23         if cnt[i] > 0 then
24             for j ⟵ i + 1 up to len(P_l) − 1 do
25                 if lst[i] = lst[j] then
26                     cnt[i] ⟵ cnt[i] + 1
27                     if cnt[j] > 0 then cnt[j] ⟵ cnt[j] + 1
28     return (P_l, lst, cnt)
29 end
```

Algorithm 19.7: $P^{\star}_{new} \longleftarrow \text{pruneOptimalSet}_{aga}(P^{\star}_{old}, d, k)$

Input: $P^{\star}old$: the optimal set to be pruned
Input: d: the number of divisions to be performed per dimension
Input: k: the maximum size allowed for the optimal set ($k \geq 2|F|$)
Input: [implicit] F: the set of objective functions
Data: i: a counter variable
Data: P_l: the list representation of $P^{\star}_{old}$
Data: lst: a list assigning grid coordinates to the elements of P_l
Data: cnt: the number of elements in the grid locations defined in lst
Output: $P^{\star}_{new}$: the pruned optimal set

```
1  begin
2      if len(P*_old) ≤ k then return P*_old
3      (P_l, lst, cnt) ⟵ agaDivide(P*_old, d)
4      while len(P_l) > k do
5          idx ⟵ 0
6          for i ⟵ len(P_l) − 1 down to 1 do
7              if cnt[i] > cnt[idx] then idx ⟵ i
8          for i ⟵ len(P_l) − 1 down to 0 do
9              if (lst[i] = lst[idx]) ∧ (cnt[i] > 0) then cnt[i] ⟵ cnt[i] − 1
10         P_l ⟵ deleteListItem(P_l, idx)
11         cnt ⟵ deleteListItem(cnt, idx)
12         lst ⟵ deleteListItem(lst, idx)
13     return listToSet(P_l)
14 end
```

Part II

Applications

20

Experimental Settings, Measures, and Evaluations

In this chapter we will discuss the possible experimental settings, the things that we can measure during experiments, and what information we can extract from these measurements. We will also define some suitable shortcuts which we use later in descriptions of experiments in order to save space in tables and graphics.

20.1 Settings

Experiments can only be reproduced and understood if their setup is precisely described. Especially in the area of global optimization algorithms, there is a wide variety of possible parameter settings. Incompletely documenting an experiment may lead to misunderstandings. Other researchers repeating the tests will use default settings for all parameters not specified (or any settings that they find neat) and possibly obtain totally different results. Here, we will try to list many possible parameters of experiments (without claiming completeness). Of course, not all of them are relevant in a given experiment. Instead, this list is to be understood as a hint of what to consider when specifying a test series. In many tables and graphics, we will use shortcuts of the parameter names in order to save space.

20.1.1 The Optimization Problem

As stated in Definition 1.34 on page 46, an optimization problem is a five-tuple $(\mathbb{X}, F, \mathbb{G}, Op, \text{gpm})$ specifying the problem space $\mathbb{X}$, the objective functions F, the search space $\mathbb{G}$, the set of search operations Op, and the genotype-phenotype mapping gpm. Specifying the elements of this tuple is the most important prerequisite for any experiment. Table 20.1 gives an example for this structure.

Parameter	Short	Description
Problem Space	$\mathbb{X}$	The space of possible solution candidates. (see Section 1.3.1) *Example:* The variable length natural vectors $x = (x_0, x_1, \dots)^T$, $x_i \in \mathbb{N}_0 \; \forall i \in [0, \text{len}(x) - 1]$, $0 < \text{len}(x) \leq 500 \; \forall x \in \mathbb{X}$
Objective Functions	$\boldsymbol{F}$	The objective functions which measure the utility of the solution candidates. If nothing else is stated, *minimization* is assumed. (see Definition 1.1) *Example:* $F = \{f_1, f_2\} : f_1(x) = \sum_{i=0}^{\text{len}(x)-1} x_i$, $f_2(x) = \sum_{i=0}^{\text{len}(x)-1} \sin x_i$

Search Space	$\mathbb{G}$	The space of the elements where the search operations are applied on. (see Section 1.3.1) *Example:* The variable length bit strings $g : 0 < \text{len}(g) \leq 2000$.
Search Operations	***Op***	The search operations available for the optimizer. Here it is also important to note the way in which they are applied. (see Section 1.3.1) *Example:* creation: uniform in length and values $os = 1 \rightarrow mr = 10\%$ mutation, $cr = 90\%$ multi-point crossover $os = 2 \rightarrow mr = 25\%$ mutation, $cr = 80\%$ multi-point crossover $os = 3 \rightarrow mr = 45\%$ mutation, $cr = 65\%$ multi-point crossover The results from crossover may be mutated (non-exclusive search operations).
GPM	**gpm**	The genotype-phenotype mapping translates points from the search space $\mathbb{G}$ to points in the problem space $\mathbb{X}$. (see Definition 1.30) *Example:* $x = \text{gpm}(g) \Rightarrow x_i = \sum_{j=4i}^{4i+3} g_j$, $\text{len}(x) = \left\lfloor \frac{1}{4}\text{len}(g) \right\rfloor$

Table 20.1: The basic optimization problem settings.

In this table we have defined a simple example optimization problem which has a search space composed of vectors with between 1 and 500 elements (natural numbers). These vectors are encoded as variable length bit strings, where groups of four bits stand for one vector element. As objective functions, two simple sums over the vector elements are applied. When needed, new strings with uniformly distributed length and contents are created with the creation operation. Sometimes, a test series involves tests with different settings. This is the case in this example too, where three configurations for the reproduction operations are given. In a table containing the results of the experiments, there could be a column "*os*" may contain the values from 1 to 3 corresponding to these settings. In our example, elements resulting from the crossover may be mutated (which is meant by "non-exclusive"). Therefore, the percentages in which the operators are applied do not necessarily need to sum up to 100%. With this definition, the problem can be reproduced easily and it is also possible to apply different global optimization algorithms and to compare the results.

20.1.2 The Optimization Algorithm Applied

The performance of an optimization algorithm strongly depends on its configuration. Table 20.1 lists some of the parameters most commonly involved in this book and gives examples how they could be configured.

Parameter	Short	Description
Optimization Algorithm	***alg***	The optimization algorithm used to solve the problem. (see Definition 1.39) *Example:* $alg = 0 \rightarrow$ (Parallel) Random Walks $alg = 1 \rightarrow$ evolutionary algorithm $alg = 2 \rightarrow$ Memetic Algorithm $alg = 3 \rightarrow$ Simulated Annealing $alg = 4 \rightarrow$ downhill simplex

Comparison Operator	***cm***	In multi-objective optimization, individuals are often compared according to Pareto or prevalence schemes. This parameter states which scheme was used, if any. (see Section 1.2.4) *Example:* $cm = 0 \rightarrow$ weighted sum $cm = 1 \rightarrow$ Pareto comparison
Steady-State	***ss***	Are the parent individuals in the population simply discarded (generational) or do they compete with their offspring (steady-state). This parameter is usually only valid in the context of evolutionary algorithms or other population-oriented optimizers. (see Section 2.1.6) *Example:* $ss = 0 \rightarrow$ generational $ss = 1 \rightarrow$ steady-state
Population Size	***ps***	The population size (only valid for population-oriented algorithms). *Example:* $ps \in \{10, 100, 1000, 10\,000\}$
Elitism	***el***	Are the best solution candidates preserved (elitism) or not (no elitism)? (see Definition 2.4) *Example:* $el = 0 \rightarrow$ no elitism is used $el = 1 \rightarrow$ elitism is used
Maximum Archive Size	***as***	The size of the archive with the best known individuals (only valid if elitism is used). Notice: An archive size of zero corresponds to no elitism. (see Definition 2.4) *Example:* $as \in \{0, 10, 20, 40, 80\}$
Fitness Assignment Algorithm	***fa***	The fitness assignment algorithm used. (see Section 2.3) *Example:* $fa = 0 \rightarrow$ weighted sum fitness assignment $fa = 1 \rightarrow$ Pareto ranking $fa = 2 \rightarrow$ Variety Preserving Ranking
Selection Algorithm	***sel***	The selection algorithm used. (see Section 2.4) *Example:* $sel = 0 \rightarrow$ Fitness proportionate selection $sel = 1 \rightarrow$ Tournament selection $sel = 2 \rightarrow$ Truncation Selection
Tournament Size	***k***	The number of individuals competing in tournaments in tournament selection (only valid for $sel = 1$). (see Section 2.4.8) *Example:* $k \in \{2, 3, 4, 5\}$
Convergence Prevention	***cp***	Is the simple convergence prevention method used? (see Section 2.4.8) *Example:* $cp \in \{0, 0.1, 0.2, 0.3, 0.4\}$

Table 20.2: Some basic optimization algorithm settings.

Such a table describes a set of experiments if at least one of the parameters has more than one value. If several parameters can be configured differently, the number of experiments multiplies accordingly. Then, (full) *factorial experiments*[1] [263, 2288, 681] where all possible parameter combinations are tested separately (multiple times) can be performed. Factorial experiments are one basic *design of experiments*[2] (DoE) [682, 1149, 263, 460, 2288]. Since many parameter settings of evolutionary algorithms have influence each other [648] (for example elitism and steady state and selection and fitness assignment, mutation and crossover rate, etc.), it is insufficient to test the influence of each parameter separately. Instead, DoE designs are recommended in order to determine the effect of these factors with efficient experiments.

20.1.3 Other Run Parameters

In Table 20.3, some additional parameters describing how the optimization algorithm was applied and how the experiments were carried out.

[1] http://en.wikipedia.org/wiki/Factorial_experiment [accessed 2008-08-07]
[2] http://en.wikipedia.org/wiki/Design_of_experiments [accessed 2008-10-14]

Parameter	Short	Description
Number of Training Cases	tc	The number of training cases used for evaluating the objective functions. *Example:* $tc \in \{1, 10, 20\}$
Training Case Change Policy	ct	The policy according to which the training cases are changed. (see Definition 1.39) *Example:* $ct = 0 \rightarrow$ The training cases do not change. $ct = 1 \rightarrow$ The training cases change each generation. $ct = 2 \rightarrow$ Training cases change after each evaluation.
Evaluation Limit	$mx\tau$	The maximum number of individual evaluations τ that each run is allowed to perform. (see Definition 1.42) *Example:* $mx\tau = 45\,000$
Generation Limit	mxt	The maximum number of iterations/generations t that each run is allowed to perform. (see Definition 1.43) *Example:* $mxt = 1000$
Maximum Number of Runs	mxr	The (maximum) number of runs to perform. This threshold can be combined with time constrainints which may lead to fewer runs being performed. *Example:* $mxr = 40$
Maximum Time per Run	$mxrT$	The (maximum) amount of time granted per run. *Example:* $mxrT = 40\text{h}$
Maximum Total Time	mxT	The (maximum) total time granted. *Example:* $mxT = 40\text{d}\,5\text{h}$
System Configuration	Cfg	Especially in cases where time constraints are imposed, the configuration of the system on which the experiments run becomes important. *Example:* $Cfg = 0 \rightarrow$ one 9 GHz two-core PC, 3 GiB RAM, Windows XP, Java 1.4 $Cfg = 1 \rightarrow$ one 9 GHz two-core PC, 3 GiB RAM, Windows XP, Java 1.6

Table 20.3: Some additional parameters of experiments.

20.2 Measures

In Table 20.4 we list some basic measurements that easily can be taken from each test series *of a single configuration*. From these basic results, more meaningful metrics can be computed.

Measure	Short	Description
Number of Comp. Runs	$\#r$	The total number of completed runs with the specified configuration. *Example:* $\#r = 100$
Success Evaluations	$s\tau_i$	The number of individual evaluations τ performed in run i until the first individual with optimal functional objective values occured. This individual may have non-optimal non-functional objective values or be overfitted. (see Definition 1.42) *Example:* $s\tau_i = 100 \rightarrow 1^{st}$ successful individual in evaluation 100 $s\tau_i = \emptyset \rightarrow$ no successful individual was found

Success Generations	st_i	The number of iterations/generations t performed in run i until the first individual with optimal functional objective values occurred. This individual may have non-optimal non-functional objective values or be overfitted. (see Definition 1.43) *Example:* $st_i = 800 \rightarrow 1^{st}$ successful individual in generation 800 $st_i = \emptyset \rightarrow$ no successful individual was found
Perfection Evaluation	$p\tau_i$	The number of individual evaluations τ performed in run i until a "perfect" individual with optimal functional objective values occurred. Depending on the context, this may be an individual where all objective values are optimal, a non-overfitted individual with optimal functional objectives, or both. Thus, $pt_i \geq st_i$ always holds (see Definition 1.42) *Example:* $p\tau_i = 100 \rightarrow 1^{st}$ perfect individual in evaluation 100 $p\tau_i = \emptyset \rightarrow$ no perfect individual was found
Perfection Generation	pt_i	The number of iterations/generations t performed in run i until a "perfect" individual with optimal functional objective values occurred. Depending on the context, this may be an individual where all objective values are optimal, a non-overfitted individual with optimal functional objectives, or both. Thus, $pt_i \geq st_i$ always holds (see Definition 1.43) *Example:* $pt_i = 800 \rightarrow 1^{st}$ perfect individual in generation 800 $pt_i = \emptyset \rightarrow$ no perfect individual was found
Solution Set	$X_i^\star$	The set $X_i^\star \subseteq \mathbb{X}$ of solutions returned by run i. *Example:* $X_i^\star = \left\{(0,1,2)^T, (4,5,6)^T\right\}$
Runtime	rT_i	The total time needed by run i. *Example:* $rT_i = 312\text{s}$

Table 20.4: Some basic measures that can be obtained from experiments.

The distinction between successful and perfect individuals may seem confusing at first glance and is not necessary in many experiments. Often though, such a distinction is useful. Assume, for instance, that we want to optimize a schedule for a transportation or manufacturing company. A successful schedule, in this case, would be one that allows the company to process all orders. This does not necessarily mean that it is optimal. Perfect would stand for optimal in this context while successful would translate to feasible. If we evolve programs with Genetic Programming and use training cases for their evaluation, we can consider a run as successful if it finds a program which works correctly on all training cases. This, however, could also be caused by overfitting. Then, a perfect program would be one that either also works correctly on another, larger set of test cases not used during the optimization process or whose correctness has been proven. An experimental run is called "successful" ("perfect") if it found a successful ("perfect") individual. The concrete definition of successful and perfect is problem specific and must be stated whenever using these predicates. Furthermore notice that we use the sign $\emptyset$ in order to denote runs where no successful (or perfect) solution candidate was discovered.

20.3 Evaluation

20.3.1 Simple Evaluation Measures

After a series of experiments has been carried out and the measures from Section 20.2 have been collected, we can use them to compute some first, simple metrics which can serve as basis for deriving more comprehensive statistics. The simple metrics basically cover everything mentioned in Section 28.3: For a quantity q measured in multiple experimental

runs, we can compute the number $\#\mathrm{q}$ of experiments that fulfilled the predicates attached to q and the estimators of the minimum $\check{q}$, mean $\overline{q}$, maximum $\hat{q}$, median $\mathrm{med}(q)$, and the standard deviation $s\,[q]$, and so on. Obviously, not all of them are needed or carry a meaning in every experiment. Table 20.5 lists some of these first metrics.

Measure	Short	Description
Number of Successful Runs	$\#\boldsymbol{s}$	The number of runs where successful individuals were discovered. $\#s = \lvert\{i : (s\tau_i \neq \emptyset) \wedge (0 \leq i < \#r)\}\rvert$ (20.1)
Success Fraction	$\boldsymbol{s}_{/r}$	The fraction of experimental runs that turned out successful. $s_{/r} = \frac{\#s}{\#r}$ (20.2)
Minimum Success Evaluation	$\breve{\boldsymbol{s\tau}}$	The number of evaluations τ needed by the fastest (successful) experimental run to find a successful individual. (or $\emptyset$ if no run was successful) $\breve{s\tau} = \begin{cases} \min\{s\tau_i \neq \emptyset\} & \text{if } \exists s\tau_i \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases}$ (20.3)
Mean Success Evaluation	$\overline{\boldsymbol{s\tau}}$	The average number of evaluations τ needed by the (successful) experimental runs to find a successful individual. (or $\emptyset$ if no run was successful) $\overline{s\tau} = \begin{cases} \frac{\sum_{s\tau_i \neq \emptyset} s\tau_i}{\lvert\{s\tau_i \neq \emptyset\}\rvert} & \text{if } \exists s\tau_i \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases}$ (20.4)
Maximum Success Evaluation	$\widehat{\boldsymbol{s\tau}}$	The number of evaluations τ needed by the slowest (successful) experimental run to find a successful individual. (or $\emptyset$ if no run was successful) $\widehat{s\tau} = \begin{cases} \max\{s\tau_i \neq \emptyset\} & \text{if } \exists s\tau_i \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases}$ (20.5)
Minimum Success Generation	$\breve{\boldsymbol{st}}$	The number of generations/iterations t needed by the fastest (successful) experimental run to find a successful individual. (or $\emptyset$ if no run was successful) $\breve{st} = \begin{cases} \min\{st_i \neq \emptyset\} & \text{if } \exists st_i \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases}$ (20.6)
Mean Success Generation	$\overline{\boldsymbol{st}}$	The average number of generations/iterations t needed by the (successful) experimental runs to find a successful individual. (or $\emptyset$ if no run was successful) $\overline{st} = \begin{cases} \frac{\sum_{st_i \neq \emptyset} st_i}{\lvert\{st_i \neq \emptyset\}\rvert} & \text{if } \exists st_i \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases}$ (20.7)
Maximum Success Generation	$\widehat{\boldsymbol{st}}$	The number of Generations generations/iterations t needed by the slowest (successful) experimental run to find a successful individual. (or $\emptyset$ if no run was successful) $\widehat{st} = \begin{cases} \max\{st_i \neq \emptyset\} & \text{if } \exists st_i \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases}$ (20.8)
Number of Perfect Runs	$\#\boldsymbol{p}$	The number of runs where perfect individuals were discovered. $\#p = \lvert\{i : (p\tau_i \neq \emptyset) \wedge (0 \leq i < \#r)\}\rvert$ (20.9)
Perfection Fraction	$\boldsymbol{p}_{/r}$	The fraction of experimental runs that found perfect individuals. $p_{/r} = \frac{\#p}{\#r}$ (20.10)
Minimum Perfection Evaluation	$\breve{\boldsymbol{p\tau}}$	The number of evaluations τ needed by the fastest (perfect) experimental run to find a perfect individual. (or $\emptyset$ if no run was found one) $\breve{p\tau} = \begin{cases} \min\{p\tau_i \neq \emptyset\} & \text{if } \exists p\tau_i \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases}$ (20.11)
Mean Perfection Evaluation	$\overline{\boldsymbol{p\tau}}$	The average number of evaluations τ needed by the (perfect) experimental runs to find a perfect individual. (or $\emptyset$ if no run was found one) $\overline{p\tau} = \begin{cases} \frac{\sum_{p\tau_i \neq \emptyset} p\tau_i}{\lvert\{p\tau_i \neq \emptyset\}\rvert} & \text{if } \exists p\tau_i \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases}$ (20.12)

Maximum Perfection Evaluation	$\widehat{p\tau}$	The number of evaluations τ needed by the slowest (perfect) experimental run to find a perfect individual. (or $\emptyset$ if no run found one) $$\widehat{p\tau} = \begin{cases} \max\{p\tau_i \neq \emptyset\} & \text{if } \exists p\tau_i \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases} \quad (20.13)$$
Minimum Perfection Generation	$\widecheck{pt}$	The number of generations/iterations t needed by the fastest (perfect) experimental run to find a perfect individual. (or $\emptyset$ if no run was found one) $$\widecheck{pt} = \begin{cases} \min\{pt_i \neq \emptyset\} & \text{if } \exists pt_i \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases} \quad (20.14)$$
Mean Perfection Generation	$\overline{pt}$	The average number of generations/iterations t needed by the (perfect) experimental runs to find a perfect individual. (or $\emptyset$ if no run was found one) $$\overline{pt} = \begin{cases} \frac{\sum_{pt_i \neq \emptyset} pt_i}{\lvert\{pt_i \neq \emptyset\}\rvert} & \text{if } \exists pt_i \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases} \quad (20.15)$$
Maximum Perfection Generation	$\widehat{pt}$	The number of generations/iterations t needed by the slowest (perfect) experimental run to find a perfect individual. (or $\emptyset$ if no run found one) $$\widehat{pt} = \begin{cases} \max\{pt_i \neq \emptyset\} & \text{if } \exists pt_i \neq \emptyset \\ \emptyset & \text{otherwise} \end{cases} \quad (20.16)$$
Mean Runtime	$\overline{rT}$	The arithmetic mean of the runtime consumed by the single runs. $$\overline{rT} = \frac{1}{\#r}\sum_{i=0}^{\#r-1} rT_i \quad (20.17)$$

Table 20.5: Simple evaluation results.

20.3.2 Sophisticated Estimates

The average generation of success $\overline{st}$ is an estimate of the expected number of iterations needed by the optimizer to find a solution to the optimization problem specified. From the measurements taken during the experiments, however, we can also extract some more sophisticated estimates which give us more information about the optimization process.

Cumulative Probability of Success etc.

One of these estimate is the *cumulative probability of success* $CPs(ps, t')$ introduced by Koza [1196]. It approximates the probability that a population-based optimization algorithm with a population size ps solves a given problem until iteration (generation) t'. Basically, it can easily be estimated from the experimental data as follows:

$$CPs(ps, t') = \frac{\lvert\{st_i : st_i \leq t'\}\rvert}{\#r} \quad (20.18)$$

The probability of solving the problem until the t'^{th} iteration at least once in st_n independent runs then becomes approximately $1 - (1 - CPs(ps, t'))^{st_n}$. If we want to find out how many runs with t' iterations we need to solve the problem with probability z, we have to solve $z = 1 - (1 - CPs(ps, t'))^{st_n}$. This equation can be solved and we obtain the function $st_n(z, ps, t')$:

$$st_n(z, ps, t') = \begin{cases} \frac{\log(1-z)}{\log(1-CPs(ps,t'))} & \text{if } 0 < CPs(ps, t') < z \\ 1 & \text{if } CPs(ps, t') \geq z \\ +\infty & \text{otherwise} \end{cases} \quad (20.19)$$

From this value, we can directly compute an estimate of the number of objective function evaluations $st_n(z, ps, t')$ needed (i. e., the individuals to be processed) to find a solution with probability z if $st_n(z, ps, t')$ independent runs proceed up to t' iterations. If we have a generational evolutionary algorithm (i. e., $ss = 1$), this would be $st_n(z, ps, t') * ps * t'$. In

a steady-state algorithm where all ps parents compete with a total of ps offspring and tc training cases are used that change each generation ($ct = 1$), we would require $2 * ps * tc * st_n(z, ps, t') * t'$ evaluations. If the training cases where constant, we would not need to re-evaluate the parents in order to determine their objective values, and $st_n(z, ps, t')$ would become $ps * tc * st_n(z, ps, t') * t'$.

Obviously, how the value of $st_n(z, ps, t')$ is computed strongly depends on the optimization algorithm applied and may be different from case to case. For the sake of simplicity, we assume that even if we have $|F| > 1$ objective functions, all of them can be evaluated in one single evaluation step together if not explicitly stated otherwise.

Since we often distinguish successful and perfect solutions in this book, we can easily derive the estimates CPp, pt_n, and $p\tau_n$ analogously to CPs, st_n, and $s\tau_n$:

$$CPp(ps, t') = \frac{|\{pt_i : pt_i \leq t'\}|}{\#r} \tag{20.20}$$

$$pt_n(z, ps, t') = \begin{cases} \frac{\log(1-z)}{\log(1-CPp(ps,t'))} & \text{if } 0 < CPp(ps, t') < z \\ 1 & \text{if } CPp(ps, t') \geq z \\ +\infty & \text{otherwise} \end{cases} \tag{20.21}$$

20.4 Verification

When obtaining measures like the mean number of individual evaluations $\overline{s\tau}$ needed to solve a given problem for multiple optimizers or for several configurations of the same optimization algorithm, one would tend to say that an algorithm/configuration A with $\overline{s\tau}(A) < \overline{s\tau}(B)$ is better than an algorithm/configuration B. Such a statement should never be made without further discussion and statistical foundation. Never forget that measures or evaluation results obtained from experiments are always *estimates*[3], i. e., guesses on the real parameter of an unknown probability distribution driving the process (optimization algorithm) which we have sampled with our measurements. An estimate should never be considered to be more than a direction, a pointer to an area, where the real values of the parameters are.

20.4.1 Confidence Intervals or Statistical Tests

So, instead of defining the mean number of evaluations to success as a single number, we could instead compute a confidence interval (see Section 28.7.3). A confidence interval specifies boundaries inside which the true value of the estimated quantity is located with a certain probability. Using a bit more math, we could derive an interval like $P(1000 \leq E[s\tau(A)] \leq 3000) \geq 90\%$, which is far more meaningful than just stating that $\overline{s\tau}(A) = 2000$. If the upper limit of the confidence interval of $E[s\tau(A)]$ is below the lower limit of the confidence interval for $E[s\tau(B)]$, it would indeed be justified to say that algorithm A performs better than B.

Computing the conventional confidence intervals discussed in Section 28.7.3 has a drawback when it comes to experiments. If you look up the examples there, you will find that all equations there assume that the measured quantity has one of the well-known probability distributions. In other words, for deriving the aforementioned interval for A, we would have to assume that the $s\tau(A)$ are often distributed, for instance. Of course we do not know if this is the case, and normally we *cannot* know. More often, we even have strong evidence that such an assumption would be rank nonsense. A normal distribution is a continuous distribution which stretches to infinity in both directions. Even if we ignore that $s\tau$ surely is not a continuous but discrete quantity, it will definitely never be negative. Furthermore, if

[3] Some introduction on estimation theory can be found in Section 28.7, by the way.

the optimization algorithm used for the experiments is population-oriented, $s\tau$ is often considered to be a multiple of the population size (see Section 20.3.2, for example). Computing a confidence interval using obviously wrong or even unverified/unverifiable assumptions is useless, wrong, and misleading.

This brings us back to square one, to the quantities which we have derived with the previously discussed evaluation methods. But there exists another way to check whether $\overline{s\tau}(A) < \overline{s\tau}(B)$ is significant or if this result is rather likely to be coincidence: statistical testing. Statistical tests are briefly discussed in Section 28.8 in this book. The idea of testing is that first, a so-called null hypothesis H_0 is defined which states "$E[s\tau(A)]$ and $E[s\tau(B)]$ are equal." The alternative hypothesis H_1 would be "$E[s\tau(A)]$ and $E[s\tau(B)]$ are *not* equal." The goal is to find out whether or not there is enough statistical evidence to reject H_0 and to accept H_1 with a certain, small probability p of error. Of course, most of the hypothesis tests have the same problem than the conventional confidence intervals from Section 28.7.3, they assume certain probability distributions driving the measurements. The set of *non-parametric* tests discussed in Section 28.8.1 works without such assumptions.

Thus, whenever we have insufficient information about the distribution of the samples, these tests are the method of choice for checking if experimental results indeed carry some meaning or not. Nevertheless, it is very important to realize that even these tests have certain requirements which must not be ignored.

Interestingly, many statistical tests can be inverted and used to compute confidence intervals as noted in Section 28.7.3 which closes the circle of this section.

20.4.2 Factorial Experiments

If we have an experiment where multiple parameter configurations are tested, i.e., a factorial experiment [263, 2288, 681], we often want to find two things:

1. the best possible configuration, and
2. which settings of one parameter are (in average) good and which are bad.

Notice that a configuration consisting only of the worst possible settings of all parameters can still be the best configuration possible – if the parameter settings interact. In Section 20.1 we have already mentioned that this is often the case in optimization algorithms such as evolutionary algorithms. On the other hand, knowing general trends for certain parameters is valuable too. Obviously, the best observed parameter configuration is the one with the best mean or median performance. If it is significantly better than the others needs to be tested.

Finding out whether a certain parameter configuration is good or not is relatively easy in factorial experiments. Assume we have run an experiment with the five parameters population size (either $ps = 512$ or $ps = 1024$), convergence prevention ($cp = 0$ or $cp = 0.3$), steady-state or generational populations ($ss = 1$ or $ss = 0$), mutation rates $mr = 3\%$ and $mr = 15\%$, and crossover rates $cr = 60\%$ and $cr = 80\%$. In the case of a full factorial experiment, we would thus test $2^5 = 32$ different configurations. For each configuration, mxr experimental runs are performed. Assume furthermore that we are considered in the estimate $p_{/r}$ of the probability of finding a perfect solution in a single run and the expected number of individual evaluations $pt_n(z, ps, t')$ needed to find such a perfect solution with probability z (if runs of the evolutionary algorithm were performed with population size ps up to t' iterations).

After all $32 * mxr$ runs, we would compute these measures for each single configuration. The influence of the two population size settings on the perfection rate $p_{/r}$ can then be estimated by dividing the testing configurations into two groups, those with $ps = 1024$ and those with $ps = 512$. For both groups, the arithmetic mean $\overline{p_{/r}}$ and the median $\text{med}(p_{/r})$ are computed separately and compared. In Table 20.6, we see that the mean and the median of the configurations with 1024 individuals in the population are higher than for those with $ps = 512$. As one would expect, the larger population has a higher chance of completing

a run with finding a correct solution than the smaller population algorithm. Now we test this trend with the mentioned non-parametric tests[4] by pairwise grouping the results of the runs which have exactly the same configurations except for their population size settings. In the example Table 20.6, we have 16 such pairs and find that all three tests agree on the significance of the result.

$ps = 1024$ vs. $ps = 512$ (based on 16 samples)	
Test according to $p_{/r}$ (higher is better)	
Sign test: (see Section 28.8.1)	$\text{med}(p_{/r})\|_{ps=1024} = 0.19$, $\text{med}(p_{/r})\|_{ps=512} = 0.09$, $\alpha \approx 0.0063 \Rightarrow$ *significant* at level $\alpha = 0.01$
Randomization test: (see Section 28.8.1)	$\overline{p_{/r}}\|_{ps=1024} = 0.06$, $\overline{p_{/r}}\|_{ps=512} = 0.0$, $\alpha \approx 0.0024 \Rightarrow$ *significant* at level $\alpha = 0.01$
Signed rankt test: (see Section 28.8.1)	$R(p_{/r})\|_{ps:1024-512} = 114.0$, $\alpha \approx 0.019 \Rightarrow$ *not significant* at level $\alpha = 0.01$
Test according to $p\tau_n$ (lower is better)	
Sign test: (see Section 28.8.1)	$\text{med}(p\tau_n)\|_{ps=1024} = 1.66 \cdot 10^8$, $\text{med}(p\tau_n)\|_{ps=512} = +\infty$, $\alpha \approx 0.1940 \Rightarrow$ *not significant* at level $\alpha = 0.01$
Randomization test: (see Section 28.8.1)	$\overline{p\tau_n}\|_{ps=1024} = +\infty$, $\overline{p\tau_n}\|_{ps=512} = +\infty$, *could not be applied*
Signed rankt test: (see Section 28.8.1)	$R(p\tau_n)\|_{ps:1024-512} = -94.0$, $\alpha \approx 0.0601 \Rightarrow$ *not significant* at level $\alpha = 0.01$

Table 20.6: $ps = 1024$ vs. $ps = 512$ (based on 16 samples)

For $p\tau_n$, we can also find differences between the two groups. However, as it (could have) turned out, multiple configurations were not able to yield a solution in any of the runs (i. e., have $p_{/r} = 0$) and thus, their $p\tau_n$ becomes infinite. Due to this configuration, the randomization test could not be applied. Besides the numerical problems here, another reason why some of tests cannot be used would be if we had too many samples, for instance (see the discussion of the randomization test in Section 28.8.1). Although the larger population size again seems to better in the sample, the tests show that there is not enough evidence to support this expectations and that the result could have shown up coincidentally. A more thorough example for this approach can be found in Section 21.3.2 on page 366.

[4] We can assume that both $p_{/r}$ and $p\tau_n$ are continuous quantities for large mxr.

21

Benchmarks and Toy Problems

In this chapter, we discuss some benchmark and toy problems which are used to demonstrate the utility of global optimization algorithms. Such problems have usually no direct real-world application but are well understood, widely researched, and can be used

1. to measure the speed and the ability of genetic algorithm algorithms to find good solutions, as done for example by Luke and Panait [1319], Borgulya [250], and Liu et al. [1295],
2. as benchmark for comparing different optimization approaches, as done by Zitzler et al. [2330] and Purshouse and Fleming [1679], for instance,
3. to derive theoretical results since they are normally well understood in a mathematical sense, as done for example by Jansen and Wegener [1039],
4. as basis to verify theories, as used for instance by Burke et al. [308] and Langdon and Poli [1241],
5. as playground to test new ideas, research, and developments,
6. as easy-to-understand examples to discuss features and problems of optimization (as done here in Section 1.2.2 on page 27),
7. for demonstration purposes, since they normally are interesting, funny, and can be visualized in a nice manner.

21.1 Real Problem Spaces

Mathematical benchmark functions are especially interesting for testing and comparing techniques based on real vectors ($\mathbb{X} = \mathbb{R}^n$) like plain Evolution Strategy (see Chapter 5 on page 227), Differential Evolution (see Section 5.5 on page 229), and Particle Swarm Optimization (see Chapter 9 on page 249). However, they only require such vectors as solution candidates, i. e., elements of the problem space $\mathbb{X}$. Hence, techniques with different search spaces $\mathbb{G}$, like genetic algorithms, can also be applied to them, given that a genotype-phenotype mapping is provided accordingly.

The optima or the Pareto frontier of benchmark functions has already been determined theoretically. When applying an optimization algorithm to the functions, we are interested in the number of solution candidates which they need to process in order to find the optima and how close we can get to them. They also give us a great opportunity to find out about the influence of parameters like population size, the choice of the selection algorithm, or the efficiency of reproduction operations.

21.1.1 Single-Objective Optimization

In this section, we list some of the most important benchmark functions for scenarios involving only a single optimization criterion. This, however, does not mean that the search

space has only a single dimension – even a single-objective optimization can take place in n-dimensional space $\mathbb{R}^n$.

Sphere

The sphere function listed by Suganthan et al. [1979] (or F_1 by De Jong [512]) and defined here in Table 21.1 is a very simple measure of efficiency of optimization methods. They have, for instance, been used by Rechenberg [1713] for testing his Evolution Strategy-approach.

function	$f_{sphere}(x) = \sum_{i=1}^{n} x_i^2$	(21.1)
domain	$\mathbb{X} \subset \mathbb{R}^n,\ X_i \in [-10, 10]$	(21.2)
optimum	$\mathbf{x}^\star = (0, 0, .., 0)^T$	(21.3)
separable	yes	
multimodal	no	

Table 21.1: The Sphere function.

TODO

21.1.2 Multi-Objective Optimization

In this section, we list some of the most important benchmark functions for scenarios involving multiple objectives (see Section 1.2.2 on page 27). A comprehensive review on such problems is given by Huband et al. [972]. Other multi-objective problems can be found in [546].

TODO

21.1.3 Dynamic Fitness Landscapes

The moving peaks benchmarks independently developed by Branke [277, 278] and Morrison and De Jong [1465] in order to illustrate the behavior of dynamic environments as discussed in Section 1.4.9 on page 76. Figure 21.1 shows an example of this benchmark for a two-dimensional real parameter setting (the third dimension is the fitness).

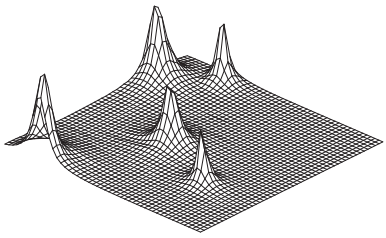


Fig. 21.1.a: $t = 0$

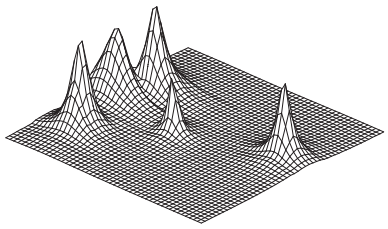


Fig. 21.1.b: $t = 1$

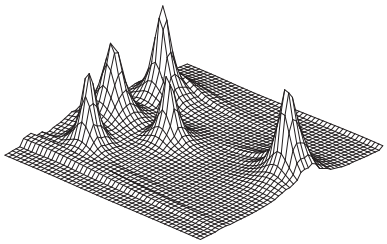


Fig. 21.1.c: $t = 2$

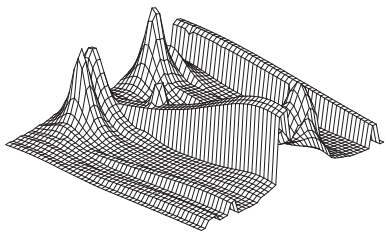


Fig. 21.1.d: $t = 3$

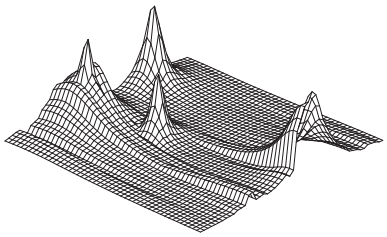


Fig. 21.1.e: $t = 4$

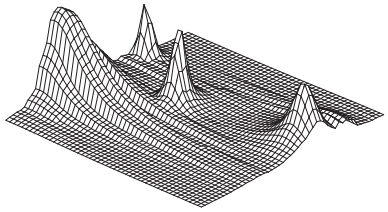


Fig. 21.1.f: $t = 6$

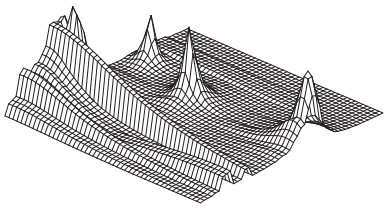


Fig. 21.1.g: $t = 7$

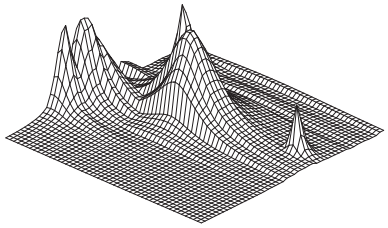


Fig. 21.1.h: $t = 13$

Figure 21.1: An example for the moving peaks benchmark of Branke [277, 278]

21.2 Binary Problem Spaces

21.2.1 Kauffman's NK Fitness Landscapes

The ideas of fitness landscapes[1] and epistasis[2] came originally from evolutionary biology and later were adopted by evolutionary computation theorists. It is thus not surprising that biologists also contributed much to the research of both. In the late 1980s, Kauffman [1098] defined the *NK fitness landscape* [1100, 1098, 1101], a family of objective functions with tunable epistasis, in an effort to investigate the links between epistasis and ruggedness.

The problem space and also the search space of this problem are bit strings of the length N, i. e., $\mathbb{G} = \mathbb{X} = \mathbb{B}^N$. Only one single objective function is used and referred to as fitness

[1] Fitness landscapes have been introduced in Section 1.3.2 on page 47.
[2] Epistasis is discussed in Section 1.4.6 on page 68.

function $F_{N,K} : \mathbb{B}^N \mapsto \mathbb{R}^+$. Each gene x_i contributes one value $f_i : \mathbb{B}^{K+1} \mapsto [0,1] \subset \mathbb{R}^+$ to the fitness function which is defined as the average of all of these N contributions. The fitness f_i of a gene x_i is determined by its allele and the alleles at K other loci $x_{i_1}, x_{i_2}, .., x_{i_K}$ with $i_{1..K} \in [0, N-1] \setminus \{i\} \subset \mathbb{N}_0$, called its *neighbors*.

$$F_{N,K}(x) = \frac{1}{N} \sum_{i=0}^{N-1} f_i \left(x_i, x_{i_1}, x_{i_2}, .., x_{i_K}\right) \tag{21.4}$$

Whenever the value of a gene changes, all the fitness values of the genes to whose neighbor set it belongs will change too – to values uncorrelated to their previous state. While N describes the basic problem complexity, the intensity of this epistatic effect can be controlled with the parameter $K \in 0..N$: If $K = 0$, there is no epistasis at all, but for $K = N-1$ the epistasis is maximized and the fitness contribution of each gene depends on all other genes. Two different models are defined for choosing the K neighbors: *adjacent neighbors*, where the K nearest other genes influence the fitness of a gene or *random neighbors* where K other genes are therefore randomly chosen.

The single functions f_i can be implemented by a table of length 2^{K+1} which is indexed by the (binary encoded) number represented by the gene x_i and its neighbors. These tables contain a fitness value for each possible value of a gene and its neighbors. They can be filled by sampling an uniform distribution in $[0, 1)$ (or any other random distribution).

We may also consider the f_i to be single objective functions that are combined to a fitness value $F_{N,K}$ by a weighted sum approach, as discussed in Section 1.2.2. Then, the nature of NK problems will probably lead to another well known aspect of multi-objective optimization: conflicting criteria. An improvement in one objective may very well lead to degeneration in another one.

The properties of the NK landscapes have intensely been studied in the past and the most significant results from Kauffman [1099], Weinberger [2170], and Fontana et al. [721] will be discussed here. We therefore borrow from the summaries provided by Altenberg [44] and Defoin Platel et al. [549]. Further information can be found in [2258, 769, 393, 392]. An analysis of the behavior of estimation of distribution algorithms and genetic algorithms in NK landscapes has been provided by Pelikan [1632].

$K = 0$

For $K = 0$, the fitness function is not epistatic. Hence, all genes can be optimized separately and we have the classical additive multi-locus model.

1. There is a single optimum $\mathbf{x}^\star$ which is globally attractive, i. e., which can and will be found by any (reasonable) optimization process regardless of the initial configuration.
2. For each individual $x \neq \mathbf{x}^\star$, there exists a fitter neighbor.
3. An adaptive walk[3] from any point in the search space will proceed by reducing the Hamming distance to the global optimum by 1 in each step (if each mutation only affects one single gene). The number of better neighbors equals the Hamming distance to the global optimum. Hence, the estimated number of steps of such a walk is $\frac{N}{2}$.
4. The fitness of direct neighbors is highly correlated since it shares $N-1$ components.

$K = N - 1$

For $K = N-1$, the fitness function equals a random assignment of fitness to each point of the search space.

1. The probability that a genotype is a local optimum is $\frac{1}{N-1}$.
2. The expected total number of local optima is thus $\frac{2^N}{N+1}$.

[3] See Section 17.4.3 on page 297 for a discussion of adaptive walks.

3. The average distance between local optima is approximately $2\ln(N-1)$.
4. The expected length of adaptive walks is approximately $\ln(N-1)$.
5. The expected number of mutants to be tested in an adaptive walk before reaching a local optimum is $\sum_{i=0}^{\log_2(N-1)-1} 2^i$.
6. With increasing N, the expected fitness of local optima reached by an adaptive from a random initial configuration decreases towards the mean fitness $\overline{F_{N,K}} = \frac{1}{2}$ of the search space. This is called the *complexity catastrophe* [1099].

For $K = N-1$, the work of Flyvbjerg and Lautrup [692] is of further interest.

Intermediate K

1. For small K, the best local optima share many common alleles. As K increases, this correlation diminishes. This degeneration proceeds faster for the random neighbors method than for the nearest neighbors approach.
2. For larger K, the fitness of the local optima approach a normal distribution with mean m and variance s approximately

$$m = \mu + \sigma\sqrt{2\ln(K+1)}K + 1 \tag{21.5}$$

$$s = \frac{(K+1)\sigma^2}{N(K+1+2(K+2)\ln(K+1))} \tag{21.6}$$

   where μ is the expected value of the f_i and σ^2 is their variance.
3. The mean distance between local optima, roughly twice the length of an adaptive walk, is approximately $\frac{N\log_2(K+1)}{2(K+1)}$.
4. The autocorrelation function[4] $\rho(k, F_{N,K})$ and the correlation length τ are:

$$\rho(k, F_{N,K}) = \left(1 - \frac{K+1}{N}\right)^k \tag{21.7}$$

$$\tau = \frac{-1}{\ln\left(1 - \frac{K+1}{N}\right)} \tag{21.8}$$

Computational Complexity

Altenberg [44] nicely summarizes the four most important theorems about the computational complexity of optimization of NK fitness landscapes. These theorems have been proven using different algorithms introduced by Weinberger [2171] and Thompson and Wright [2040].

1. The NK optimization problem with adjacent neighbors is solvable in $\mathbf{O}\left(2^K N\right)$ steps and thus in $\mathcal{P}$ [2171].
2. The NK optimization problem with random neighbors is $\mathcal{NP}$-complete for $K \geq 2$ [2171, 2040].
3. The NK optimization problem with random neighbors and $K = 1$ is solvable in polynomial time. [2040].

Adding Neutrality – NKp, NKq, and Technological Landscapes

As we have discussed in Section 1.4.5, natural genomes exhibit a certain degree of neutrality. Therefore, researchers have proposed extensions for the NK landscape which introduce neutrality, too [776, 777]. Two of them, the NKp and NKq landscapes, achieve this by altering the contributions f_i of the single genes to the total fitness. In the following, assume that there are N tables, each with 2^K entries representing these contributions.

[4] See Definition 1.48 on page 63 for more information on autocorrelation.

The NKp landscapes devised by Barnett [149] achieves neutrality by setting a certain number of entries in each table to zero. Hence, the corresponding allele combinations do not contribute to the overall fitness of an individual. If a mutation leads to a transition from one such zero configuration to another one, it is effectively neutral. The parameter p denotes the probability that a certain allele combination does not contribute to the fitness. As proven by Reidys and Stadler [1718], the ruggedness of the NKp landscape does not vary for different values of p. Barnett [148] proved that the degree of neutrality in this landscape depends on p.

Newman and Engelhardt [1521] follow a similar approach with their NKq model. Here, the fitness contributions f_i are integers drawn from the range $[0, q)$ and the total fitness of a solution candidate is normalized by multiplying it with $1/q-1$. A mutation is neutral when the new allelic combination resulting from it leads to the same contribution than the old one. In NKq landscapes, the neutrality decreases with rising values of q. In [777], you can find a thorough discussion of the NK, the NKp, and the NKq fitness landscape.

With their technological landscapes, Lobo et al. [1300] follow the same approach from the other side: the discretize the continuous total fitness function $F_{N,K}$. The parameter M of their technological landscapes corresponds to a number of bins $[0, 1/M)$, $[1/M, 2/M)$, ..., into which the fitness values are sorted and put away.

21.2.2 The p-Spin Model

Motivated by the wish of researching the models for the origin of biological information by Anderson [51, 1747] and Tsallis and Ferreira [2056], Amitrano et al. [48] developed the p-spin model. This model is an alternative to the NK fitness landscape for tunable ruggedness [2172]. Other than the previous models, it includes a complete definition for all genetic operations which will be discussed in this section.

The p-spin model works with a fixed population size ps of individuals of an also fixed length N. There is no distinction between genotypes and phenotype, in other words, $\mathbb{G} = \mathbb{X}$. Each gene of an individual x is a binary variable which can take on the values -1 and 1.

$$\mathbb{G} = \{-1, 1\}^N, \quad x_{i[j]} \in \{-1, 1\} \ \forall i \in [1..ps], j \in [0..N-1] \tag{21.9}$$

On the 2^N possible genotypic configurations, a space with the topology of an N-dimensional hypercube is defined where neighboring individuals differ in exactly one element. On this genome, the Hamming distance dist_{Ham} can be defined as

$$\text{dist}_{Ham}(x_1, x_2) = \frac{1}{2} \sum_{i=0}^{N-1} (1 - x_{1[i]} x_{2[i]}) \tag{21.10}$$

Two configurations are said to be ν mutations away from each other if they have the Hamming distance ν. Mutation in this model is applied to a fraction of the $N * ps$ genes in the population. These genes are chosen randomly and their state is changed, i. e., $x_{[i]} \to -x_{[i]}$.

The objective function f_K (which is called *fitness function* in this context) is subject to maximization, i. e., individuals with a higher value of f_K are less likely to be removed from the population. For very subset z of exactly K genes of a genotype, one contribution $A(z)$ is added to the fitness.

$$f_K(x) = \sum_{\forall z \in \mathcal{P}([0..K]) \wedge |z| = K} A(x_{[z[0]]}, x_{[z[1]]}, \ldots, x_{[z[K-1]]}) \tag{21.11}$$

$A(z) = a_z * z[0] * z[1] * \cdots * z[K-1]$ is the product of an evenly distributed random number a_z and the elements of z. For $K = 2$, f_2 can be written as $f_2(x) = \sum_{i=0}^{K-1} \sum_{j=0}^{k-1} a_{ij} x_{[i]} x_{[j]}$, which corresponds to the *spin-glass* [208, 1402] function first mentioned by Anderson [51] in this context. With rising values of K, this fitness landscape becomes more rugged. Its correlation length τ is approximately $N/2K$, as discussed thoroughly by Weinberger and Stadler [2172].

For selection, Amitrano et al. [48] suggest to use the measure $P_D(x)$ defined by Rokhsar et al. [1747] as follows:

$$P_D(x) = \frac{1}{1 + e^{\beta(f_K(x) - H_0)}} \tag{21.12}$$

where the coefficient β is a sharpness parameter and H_0 is a threshold value. For $\beta \to \infty$, all individuals x with $f_K(x) < H_0$ will die and for $\beta = 0$, the death probability is always $\frac{1}{2}$. The individuals which have died are then replaced with copies of the survivors.

21.2.3 The ND Family of Fitness Landscapes

The ND family of fitness landscape has been developed by Beaudoin et al. [161] in order to provide a model problem with tunable neutrality.

The degree of neutrality ν is defined as the number (or, better, the fraction of) neutral neighbors (i.e., those with same fitness) of a solution candidate, as specified in Equation 1.42 on page 64. The populations of optimization processes residing on a neutral network (see Section 1.4.5 on page 66) tend to converge into the direction of the individual which has the highest degree of neutrality on it. Therefore, Beaudoin et al. [161] create a landscape with a predefined neutral degree distribution.

The search space is again the set of all binary strings of the length N, $\mathbb{G} = \mathbb{X} = \mathbb{B}^N$. Thus, a genotype has minimally 0 and at most N neighbors with Hamming distance 1 that have the same fitness. The array D has the length $N + 1$ and the element $D_{[i]}$ represents the fraction of genotypes in the population that have i neutral neighbors.

Beaudoin et al. [161] provide an algorithm that divides the search space into neutral networks according to the values in D. Since this approach cannot exactly realize the distribution defined by D, the degrees of neutrality of the single individuals are subsequently refined with a Simulated Annealing algorithm. The objective (fitness) function is created in form of a complete table mapping $\mathbb{X} \mapsto \mathbb{R}$. All members of a neutral network then receive the same, random fitness.

If it is ensured that all members in a neutral network always have the same fitness, its actual value can be modified without changing the topology of the network. Tunable deceptiveness is achieved by setting the fitness values according to the *Trap* Functions [540, 12, 1069].

Trap Functions

Trap functions $f_{b,r,\mathbf{x}^\star} : \mathbb{B}^N \mapsto \mathbb{R}$ are subject to maximization based on the Hamming distance to a pre-defined global optimum $\mathbf{x}^\star$. They build a second, local optimum in form of a hill with a gradient pointing away from the global optimum. This trap is parameterized with two values, b and r, where b corresponds to the width of the attractive basins and r to their relative importance.

$$f_{b,r,\mathbf{x}^\star}(x) = \begin{cases} 1 - \frac{\text{dist}_{Ham}(x,\mathbf{x}^\star)}{Nb} & \text{if } N * \text{dist}_{Ham}(x,\mathbf{x}^\star) < b \\ \frac{r\left(\frac{1}{N}\text{dist}_{Ham}(x,\mathbf{x}^\star) - b\right)}{1-b} & \text{otherwise} \end{cases} \tag{21.13}$$

Equation 21.14 shows a similar "Trap" function defined by Ackley [12] where $u(x)$ is the number of ones in the bit string x of length n and $z = \lfloor 3n/4 \rfloor$ [1069]. The objective function $f(x)$ is subject to maximization is sketched in Figure 21.2.

$$f(x) = \begin{cases} (8n/z)\,(z - u(x)) & \text{if } u(x) \leq z \\ (10n/\,(n - z))\,(u(x) - z) & \text{otherwise} \end{cases} \tag{21.14}$$

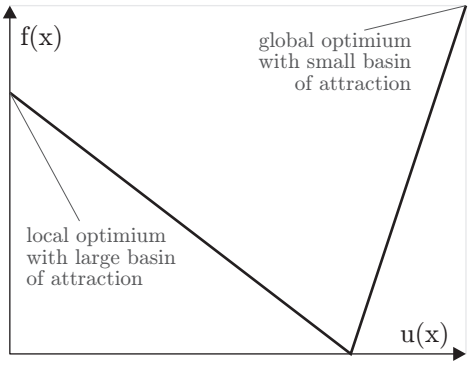


Figure 21.2: Ackley's "Trap" function [12, 1069].

21.2.4 The Royal Road

The Royal Road functions developed by Mitchell et al. [1432] and presented first at the Fifth International Conference on Genetic Algorithms in July 1993 are a set of special fitness landscapes for genetic algorithms [1067, 1432, 731, 1682, 2098]. Their problem space $\mathbb{X}$ and search space $\mathbb{G}$ are fixed-length bit strings. The Royal Road functions are closely related to the Schema Theorem[5] and the Building Block Hypothesis[6] and were used to study the way in which highly fit schemas are discovered. They therefore define a set of schemas $S = s_1, s_2, \ldots, s_n$ and an objective function (here referred to as fitness function), subject to maximization, as

$$f(x) = \sum_{\forall s \in S} c(s)\sigma(s,x) \tag{21.15}$$

where $x \in \mathbb{X}$ is a bit string, $c(s)$ is a value assigned to the schema s and $\sigma(s,x)$ is defined as

$$\sigma(s,x) = \begin{cases} 1 & \text{if } x \textit{ is an instance of } s \\ 1 & \text{otherwise} \end{cases} \tag{21.16}$$

In the original version, $c(s)$ is the order of the schema s, i. e., $c(s) \equiv \text{order}(s)$, and S is specified as follows (where $*$ stands for the *don't care* symbol as usual).

```
1  s1  = 11111111************************************************************; c(s1 ) = 8
2  s2  = ********11111111********************************************************; c(s2 ) = 8
3  s3  = ****************11111111****************************************************; c(s3 ) = 8
4  s4  = ************************11111111************************************************; c(s4 ) = 8
5  s5  = ********************************11111111********************************; c(s5 ) = 8
6  s6  = ****************************************11111111****************************; c(s6 ) = 8
7  s7  = ************************************************11111111********************; c(s7 ) = 8
8  s8  = ********************************************************11111111; c(s8 ) = 8
9  s9  = 1111111111111111************************************************************; c(s9 ) = 16
10 s10 = ****************1111111111111111********************************************; c(s10) = 16
11 s11 = ********************************1111111111111111****************************; c(s11) = 16
12 s12 = ************************************************1111111111111111; c(s12) = 16
13 s13 = 11111111111111111111111111111111************************************************; c(s13) = 32
14 s14 = ********************************11111111111111111111111111111111; c(s14) = 32
15 s15 = 1111111111111111111111111111111111111111111111111111111111111111; c(s15) = 64
```

Listing 21.1: An example Royal Road function.

[5] See Section 3.6 on page 150 for more details.
[6] The Building Block Hypothesis is elaborated on in Section 3.6.5 on page 152

By the way, from this example, we can easily see that a fraction of all mutation and crossover operations applied to most of the solution candidates will fall into the don't care areas. Such modifications will not yield any fitness change and therefore are neutral.

The Royal Road functions provide certain, predefined stepping stones (i. e., building blocks) which (theoretically) can be combined by the genetic algorithm to successively create schemas of higher fitness and order. Mitchell et al. [1432] performed several tests with their Royal Road functions. These tests revealed or confirmed that

1. Crossover is a useful reproduction operation in this scenario. Genetic algorithms which apply this operation clearly outperform hill climbing approaches solely based on mutation.
2. In the spirit of the Building Block Hypothesis, one would expect that the intermediate steps (for instance order 32 and 16) of the Royal Road functions would help the genetic algorithm to reach the optimum. The experiments of Mitchell et al. [1432] showed the exact opposite: leaving them away speeds up the evolution significantly. The reason is the fitness difference between the intermediate steps and the low-order schemas is high enough that the first instance of them will lead the GA to converge to it and wipe out the low-order schemas. The other parts of this intermediate solution play no role and may allow many zeros to *hitchhike* along.

Especially this last point gives us another insight on how we should construct genomes: the fitness of combinations of good low-order schemas should not be too high so other good low-order schemas do not extinct when they emerge. Otherwise, the phenomenon of *domino convergence* researched by Rudnick [1773] and outlined in Section 1.4.2 and Section 21.2.5 may occur.

Variable-Length Representation

The original Royal Road problems can be defined for binary string genomes of any given length n, as long as n is fixed. A Royal Road benchmark for variable-length genomes has been defined by Defoin Platel et al. [548].

The problem space $\mathbb{X}_\Sigma$ of the VLR (variable-length representation) Royal Road problem is based on an alphabet Σ with $N = |\Sigma|$ letters. The fitness of an individual $x \in \mathbb{X}_\Sigma$ is determined by whether or not consecutive *building blocks* of the length b of the letters $l \in \Sigma$ are present. This presence can be defined as

$$B_b(x,l) = \begin{cases} 1 \text{ if } \exists i : (0 \le i < (\text{len}(x) - b)) \wedge (x_{[i+j]} = l \; \forall j : 0 \le j < (b-1)) \\ 0 \text{ otherwise} \end{cases} \tag{21.17}$$

1. Where $b \geq 1$ is the length of the building blocks,
2. Σ is the alphabet with $N = |\Sigma|$ letters,
3. l is a letter in Σ,
4. $x \in \mathbb{X}_\Sigma$ is a solution candidate, and
5. $x_{[k]}$ is the k^{th} locus of x.

$B_b(x,l)$ is 1 if a building block, an uninterrupted sequence of the letter l, of at least length b, is present in x. Of course, if $\text{len}(x) < b$ this cannot be the case and $B_b(x,l)$ will be zero.

We can now define the functional objective function $f_{\Sigma b} : \mathbb{X}_\Sigma \mapsto [0,1]$ which is subject to maximization as

$$f_{\Sigma b}(x) = \frac{1}{N} \sum_{i=1}^{N} B_b(x, \Sigma_{[i]}) \tag{21.18}$$

An optimal individual $\mathbf{x}^\star$ solving the VLR Royal Road problem is thus a string that includes building blocks of length b for all letters $l \in \Sigma$. Notice that the position of these blocks plays no role. The set $\mathbf{X}^\star_\mathbf{b}$ of all such optima with $f_{\Sigma b}(\mathbf{x}^\star) = 1$ is then

$$\mathbf{X}_\mathbf{b}^\star \equiv \{\mathbf{x}^\star \in \mathbb{X}_\Sigma : B_b(\mathbf{x}^\star, l) = 1 \; \forall l \in \Sigma\} \tag{21.19}$$

Such an optimum $\mathbf{x}^\star$ for $b = 3$ and $\Sigma = \{A, T, G, C\}$ is

$$\mathbf{x}^\star = \mathbf{AAA}\,GT\mathbf{GGG}\,TAA\mathbf{TTT}\,T\mathbf{CCC}\,TCCC \tag{21.20}$$

The relevant building blocks of $\mathbf{x}^\star$ are written in bold face. As it can easily be seen, their location plays no role, only their presence is important. Furthermore, multiple occurrences of building blocks (like the second CCC) do not contribute to the fitness. The fitness landscape has been designed in a way ensuring that fitness degeneration by crossover can only occur if the crossover points are located inside building blocks and not by block translocation or concatenation. In other words, there is no inter-block epistasis.

Epistatic Road

Defoin Platel et al. [549] combined their previous work on the VLR Royal Road with Kauffman's NK landscapes and introduced the Epistatic Road. The original NK landscape works on binary representation of the fixed length N. To each locus i in the representation, one fitness function f_i is assigned denoting its contribution to the overall fitness. f_i however is not exclusively computed using the allele at the i^{th} locus but also depends on the alleles of K other loci, its neighbors.

The VLR Royal Road uses a genome based on the alphabet Σ with $N = \text{len}(\Sigma)$ letters. It defines the function $B_b(x, l)$ which returns 1 if a building block of length b containing only the character l is present in x and 0 otherwise. Because of the fixed size of the alphabet Σ, there exist exactly N such functions. Hence, the variable-length representation can be translated to a fixed-length, binary one by simply concatenating them:

$$B_b(x, \Sigma_{[0]})\; B_b(x, \Sigma_{[1]}) \ldots B_b(x, \Sigma_{[N-1]}) \tag{21.21}$$

Now we can define a NK landscape for the Epistatic Road by substituting the $B_b(x, l)$ into Equation 21.4 on page 330:

$$F_{N,K,b}(x) = \frac{1}{N} \sum_{i=0}^{N-1} f_i(B_b(x, \Sigma_{[i]}), B_b(x, \Sigma_{[i_1]}), \ldots, B_b(x, \Sigma_{[i_K]})) \tag{21.22}$$

The only thing left is to ensure that the end of the road, i. e., the presence of all N building blocks, also is the optimum of $F_{N,K,b}$. This is done by exhaustively searching the space $\mathbb{B}^N$ and defining the f_i in a way that $B_b(x, l) = 1 \; \forall \, l \in \Sigma \Rightarrow F_{N,K,b}(x) = 1$.

Royal Trees

An analogue of the Royal Road for Genetic Programming has been specified by Punch et al. [1678]. This *Royal Tree* problem specifies a series of functions A, B, C, ... with increasing arity, i. e., A has one argument, B has two arguments, C has three, and so on. Additionally, a set of terminal nodes x, y, z is defined.

For the first free levels, the perfect trees are shown Figure 21.3. An optimal A-level tree consists of an A node with an x leaf attached to it. The perfect level-B tree has a B as root with two perfect level-A trees as children. A node labeled with C having three children which all are optimal B-level trees is the optimum at C-level, and so on.

The objective function, subject to maximization, is computed recursively. The raw fitness of a node is the weighted sum of the fitness of its children. If the child is a perfect tree at the appropriate level, a perfect C tree beneath a D-node, for instance, its fitness is multiplied with the constant $FullBonus$, which normally has the value 2. If the child is not a perfect tree, but has the correct root, the weight is $PartialBonus$ (usually 1). If it is otherwise

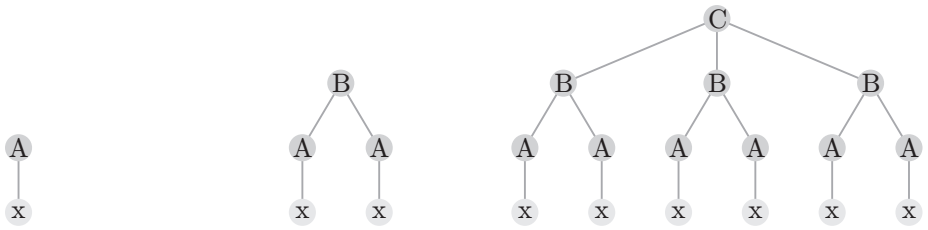


Fig. 21.3.a: Perfect A-level Fig. 21.3.b: Perfect B-level Fig. 21.3.c: Perfect C-level

Figure 21.3: The perfect Royal Trees.

incorrect, its fitness is multiplied with $Penalty$, which is $\frac{1}{3}$ per default. If the whole tree is a perfect tree, its raw fitness is finally multiplied with $CompleteBonus$ which normally is also 2. The value of a x leaf is 1.

From Punch et al. [1678], we can furthermore borrow three examples for this fitness assignment and outline them in Figure 21.4. A tree which represents a perfect A level has the score of $CompleteBonus * FullBonus * 1 = 2 * 2 * 1 = 4$. A complete and perfect tree at level B receives $CompleteBonus(FullBonus * 4 + FullBonus * 4) = 2 * (2 * 4 + 2 * 4) = 32$. At level C, this makes $CompleteBonus(FullBonus * 32 + FullBonus * 32 + FullBonus * 32) = 2(2 * 32 + 2 * 32 + 2 * 32) = 384$.

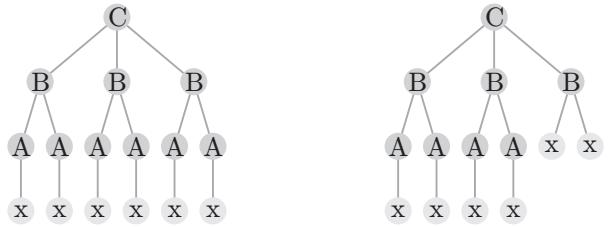


Fig. 21.4.a: $2(2*32+2*32+2*32) = 384$ Fig. 21.4.b: $2(2*32+2*32+\frac{2}{3}*1) = 128\frac{2}{3}$ Fig. 21.4.c: $2(2*32+\frac{1}{3}*1+\frac{1}{3}*1) = 64\frac{2}{3}$

Figure 21.4: Example fitness evaluation of Royal Trees

Other Derived Problems

Storch and Wegener [1968, 1969, 1970] used their Real Royal Road for showing that there exist problems where crossover helps improving the performance of evolutionary algorithms. Naudts et al. [1505] have contributed generalized Royal Road functions functions in order to study epistasis.

21.2.5 OneMax and BinInt

The OneMax and BinInt are two very simple model problems for measuring the convergence of genetic algorithms.

The OneMax Problem

The task in the OneMax (or BitCount) problem is to find a binary string of length n consisting of all ones. The search and problem space are both the fixed-length bit strings

$\mathbb{G} = \mathbb{X} = \mathbb{B}^n$. Each gene (bit) has two alleles 0 and 1 which also contribute exactly this value to the total fitness, i. e.,

$$f(x) = \sum_{i=0}^{n-1} x_{[i]}, \quad \forall x \in \mathbb{X} \tag{21.23}$$

For the OneMax problem, an extensive body of research has been provided by Ackley [12], Mühlenbein and Schlierkamp-Voosen [1481], Thierens and Goldberg [2035], Miller and Goldberg [1416], Bäck [97], Blickle and Thiele [230], and Wilson and Kaur [2230].

The BinInt Problem

The BinInt problem devised by Rudnick [1773] also uses the bit strings of the length n as search and problem space ($\mathbb{G} = \mathbb{X} = \mathbb{B}^n$). It is something like a perverted version of the OneMax problem, with the objective function defined as

$$f(x) = \sum_{i=0}^{n-1} 2^{n-i-1} x_{[i]}, \quad x_{[i]} \in \{0,1\} \ \forall i \in [0..n-1] \tag{21.24}$$

Since the bit at index i has a higher contribution to the fitness than all other bit at higher indices together, the comparison between two solution candidates x_1 and x_2 is won by the lexicographically bigger one. Thierens et al. [2036] give the example $x_1 = (1, 1, \underline{1}, 1, 0, 0, 0, 0)$ and $x_2 = (1, 1, \underline{0}, 0, 1, 1, 0, 0)$, where the first deviating bit (underlined, at index 2) fully determines the outcome of the comparison of the two.

We can expect that the bits with high contribution (high salience) will converge quickly whereas the other genes with lower salience are only pressured by selection when all others have already been fully converged. Rudnick [1773] called this sequential convergence phenomenon *domino convergence* due to its resemblance with a row of falling domino stones [2036] (see Section 1.4.2). Generally, he showed that first, the highly salient genes converge (i. e., take on the correct values in the majority of the population). Then, step by step, the building blocks of lower significance can converge, too. Another result of Rudnick's work is that mutation may stall the convergence because it can disturb the highly significant genes, which then counters the effect of the selection pressure on the less salient ones. Then, it becomes very less likely that the majority of the population will have the best alleles in these genes. This somehow dovetails with the idea of error thresholds from theoretical biology [625, 1552] which we have mentioned in Section 1.4.3. It is also explains some the experimental results obtained with the Royal Road problem from Section 21.2.4. The BinInt problem was used in the studies of Sastry and Goldberg [1810, 1811].

One of the maybe most important conclusions from the behavior of GAs applied to the BinInt problem is that applying a genetic algorithm to solve a numerical problem ($\mathbb{X} \subseteq \mathbb{R}^n$) whilst encoding the solution candidates binary ($\mathbb{G} \subseteq \mathbb{B}^n$) in a straightforward manner will like produce suboptimal solutions. Schraudolph and Belew [1836], for instance, recognized this problem and suggested a *Dynamic Parameter Encoding* (DPE) where the values genes of the bit string are readjusted over time: Initially, optimization takes place on a rather coarse grained scale and after the optimum on this scale is approximated, the focus is shifted to a finer interval and the genotypes are re-encoded to fit into this interval. In their experiments, this method works better as the direct encoding.

21.2.6 Long Path Problems

The long path problems have been designed by Horn et al. [958] in order to construct a unimodal, non-deceptive problem without noise which hill climbing algorithms still can only solve in exponential time. The idea is to wind a path with increasing fitness through the search space so that any two adjacent points on the path are no further away than one

search step and any two points not adjacent on the path are away at least two search steps. All points which are not on the path should guide the search to its origin.

The problem space and search space in their concrete realization is the space of the binary strings $\mathbb{G} = \mathbb{X} = \mathbb{B}^l$ of the fixed, *odd* length l. The objective function $f_{lp}(x)$ is subject to maximization. It is furthermore assumed that the search operations in hill climbing algorithms alter at most one bit per search step, from which we can follow that two adjacent points on the path have a Hamming distance of one and two non-adjacent points differ in at least two bits.

The simplest instance of the long path problems that Horn et al. [958] define is the *Root2path* P_l. Paths of this type are constructed by iteratively increasing the search space dimension. Starting with $P_1 = (\texttt{0}, \texttt{1})$, the path P_{l+2} is constructed from two copies $P_l^a = P_l^b$ of the path P_l as follows. First, we prepend `00` to all elements of the path P_l^a and `11` to all elements of the path P_l^b. For $l = 1$ this makes $P_1^a = (\texttt{000}, \texttt{001})$ and $P_l^b = (\texttt{110}, \texttt{111})$. Obviously, two elements on P_l^a or on P_l^b still have a Hamming distance of one whereas each element from P_l^a differs at least two bits from each element on P_l^b. Then, a bridge element B_l is created that equals the last element of P_l^a, but has `01` as the first two bits, i. e., $B_1 = \texttt{011}$. Now the sequence of the elements in P_l^b is reversed and P_l^a, B_l, and the reversed P_l^b are concatenated. Hence, $P_3 = (\texttt{000}, \texttt{001}, \texttt{011}, \texttt{111}, \texttt{110})$. Due to this recursive structure of the path construction, the path length increases exponentially with l (for odd l):

$$\text{len}(P_{l+2}) = 2 * \text{len}(P_l) + 1 \tag{21.25}$$

$$\text{len}(P_l) = 3 * 2^{\frac{l-1}{2}} - 1 \tag{21.26}$$

The basic fitness of a solution candidate is 0 if it is not on the path and its (zero-based) position on the path plus one if it is part of the path. The total number of points in the space is $\mathbb{B}^l$ is 2^l and thus, the fraction occupied by the path is approximately $3{*}2^{-\frac{l+1}{2}}$, i.e., decreases exponentially. In order to avoid that the long path problem becomes a needle-in-a-haystack problem[7], Horn et al. [958] assign a fitness that leads the search algorithm to the path's origin to all off-path points x_o. Since the first point of the path is always the string `00...0` containing only zeros, subtracting the number of ones from l, i. e., $f_{lp}(x_o) = l - \text{countOccurences}(1, x_o)$, is the method of choice. To all points on the path, l is added to the basic fitness, making them superior to all other solution candidates.

Some examples for the construction of Root2paths can be found in Table 21.2 and the path for $l = 3$ is illustrated in Figure 21.5. In Algorithm 21.1 we try to outline how the objective value of a solution candidate x can be computed online. Here, please notice two things: First, this algorithm deviates from the one introduced by Horn et al. [958] – we tried to resolve the tail recursion and also added some minor changes. Another algorithm for determining f_{lp} is given by Rudolph [1774]. The second thing to realize is that for small l, we would not use the algorithm during the individual evaluation but rather a lookup table. Each solution candidate could directly be used as index for this table which contains the objective values. For $l = 20$, for example, a table with entries of the size of 4B would consume 4MiB which is acceptable on today's computers.

The experiments of Horn et al. [958] showed that hill climbing methods that only concentrate on sampling the neighborhood of the currently known best solution perform very poor on long path problems whereas genetic algorithms which combine different solution candidates via crossover easily find the correct solution. Rudolph [1774] shows that it does so in polynomial expected time. He also extends this idea long k-paths in [1775]. Droste et al. [598] and Garnier and Kallel [774] analyze this path and find that also (1+1)-EAs can have exponential expected runtime on such unimodal functions.

It should be mentioned that the Root2paths constructed according to the method described in this section here do not have the maximum length possible for long paths. Horn

[7] See Section 1.4.5 for more information on needle-in-a-haystack problems.

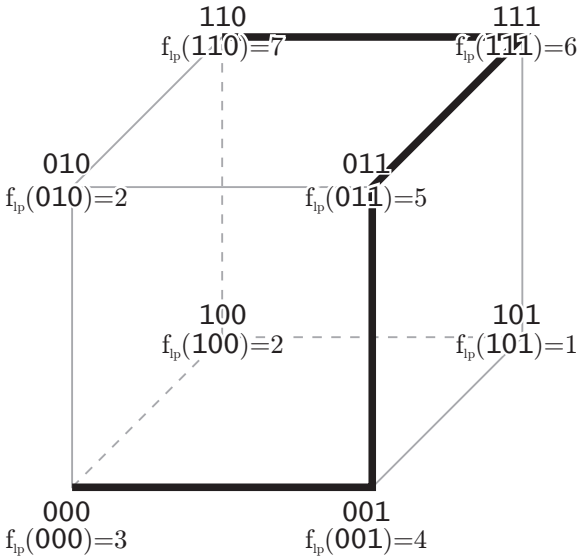


Figure 21.5: The root2path for $l = 3$.

P_1 = (0, 1)
P_3 = (000, 001, 011, 111, 110)
P_5 = (00000, 00001, 00011, 00111, 00110, 01110, 11110, 11111, 11011, 11001, 11000)
P_7 = (0000000, 0000001, 0000011, 0000111, 0000110, 0001110, 0011110, 0011111, 0011011, 0011001, 0011000, 0111000, 1111000, 1111001, 1111011, 1111111, 1111110, 1101110, 1100110, 1100111, 1100011, 1100001, 1100000)
P_9 = (000000000, 000000001, 000000011, 000000111, 000000110, 000001110, 000011110, 000011111, 000011011, 000011001, 000011000, 000111000, 001111000, 001111001, 001111011, 001111111, 001111110, 001101110, 001100110, 001100111, 001100011, 001100001, 001100000, 011100000, 111100000, 111100001, 111100011, 111100111, 111100110, 111101110, 111111110, 111111111, 111111011, 111111001, 111111000, 110111000, 110011000, 110011001, 110011011, 110011111, 110011110, 110001110, 110000110, 110000111, 110000011, 110000001, 110000000)
P_{11} = (00000000000, 00000000001, 00000000011, 00000000111, 00000000110, 00000001110, 00000011110, 00000011111, 00000011011, 00000011001, 00000011000, 00000111000, 00001111000, 00001111001, 00001111011, 00001111111, 00001111110, 00001101110, 00001100110, 00001100111, 00001100011, 00001100001, 00001100000, 00011100000, 00111100000, 00111100001, 00111100011, 00111100111, 00111100110, 00111101110, 00111111110, 00111111111, 00111111011, 00111111001, 00111111000, 00110111000, 00110011000, 00110011001, 00110011011, 00110011111, 00110011110, 00110001110, 00110000110, 00110000111, 00110000011, 00110000001, 00110000000, 01110000000, 11110000000, 11110000001, 11110000011, 11110000111, 11110000110, 11110001110, 11110011110, 11110011111, 11110011011, 11110011001, 11110011000, 11110111000, 11111111000, 11111111001, 11111111011, 11111111111, 11111111110, 11111101110, 11111100110, 11111100111, 11111100011, 11111100001, 11111100000, 11011100000, 11001100000, 11001100001, 11001100011, 11001100111, 11001100110, 11001101110, 11001111110, 11001111111, 11001111011, 11001111001, 11001111000, 11000111000, 11000011000, 11000011001, 11000011011, 11000011111, 11000011110, 11000001110, 11000000110, 11000000111, 11000000011, 11000000001, 11000000000)

Table 21.2: Some long Root2paths for l from 1 to 11 with underlined bridge elements.

Algorithm 21.1: $r \longleftarrow f_{lp}(x)$

Input: x: the solution candidate with an *odd* length
Data: s: the current position in x
Data: $sign$: the sign of the next position
Data: $isOnPath$: `true` if and only if x is on the path
Output: r: the objective value

```
1  begin
2    sign ⟵ 1
3    s ⟵ len(x) − 1
4    r ⟵ 0
5    isOnPath ⟵ true
6    while (s ≥ 0) ∧ isOnPath do
7      if s = 0 then
8        if x[0] = 1 then r ⟵ r + sign
9      sub ⟵ subList(x, s − 2, 2)
10     if sub = 11 then
11       r ⟵ r + sign ∗ (3 ∗ 2^(s/2) − 2)
12       sign ⟵ −sign
13     else
14       if sub ≠ 00 then
15         if (x[s] = 0) ∧ (x[s−1] = 1) ∧ (x[s−2] = 1) then
16           if (s = 2) ∨ [(x[s−3] = 1)∧ (countOccurences(1, subList(x, 0, s − 3)) = 0)] then
               r ⟵ r + sign ∗ (3 ∗ 2^(s/2−1) − 1)
17           else else isOnPath ⟵ false
18         else isOnPath ⟵ false
19     s ⟵ s − 2
20   if isOnPath then r ⟵ r + len(x)
21   else r ⟵ len(x) − countOccurences(1, x) − 1
22 end
```

et al. [958] also introduce *Fibonacci paths* which are longer than the Root2paths. The problem of finding maximum length paths in a l-dimensional hypercube is known as the *snake-in-the-box*[8] problem [1104, 483] which was first described by Kautz [1104] in the late 1950s. It is a very hard problem suffering from combinatorial explosion and currently, maximum snake lengths are only known for small values of l.

21.2.7 Tunable Model for Problematic Phenomena

What is a good model problem? Which model fits best to our purposes? These questions should be asked whenever we apply a benchmark, whenever we want to use something for testing the ability of a global optimization approach. The mathematical functions introduced in Section 21.1, for instance, are good for testing special mathematical reproduction operations like used in Evolution Strategies and for testing the capability of an evolutionary algorithm for estimating the Pareto frontier in multi-objective optimization. Kauffman's NK fitness landscape (discussed in Section 21.2.1) was intended to be a tool for exploring the relation of ruggedness and epistasis in fitness landscapes but can prove very useful for finding out how capable an global optimization algorithm is to deal with problems exhibiting these phenomena. In Section 21.2.4, we outlined the Royal Road functions, which were used to

[8] http://en.wikipedia.org/wiki/Snake-in-the-box [accessed 2008-08-13]

investigate the ability of genetic algorithms to combine different useful formae and to test the Building Block Hypothesis. The Artificial Ant (Section 21.3.1) and the GCD problem from Section 21.3.2 are tests for the ability of Genetic Programming of learning algorithms.

All these benchmarks and toy problems focus on specific aspects of global optimization and will exhibit different degrees of the problematic properties of optimization problems to which we had devoted Section 1.4:

1. premature convergence and multimodality (Section 1.4.2),
2. ruggedness (Section 1.4.3),
3. deceptiveness (Section 1.4.4),
4. neutrality and redundancy (Section 1.4.5),
5. overfitting and oversimplification (Section 1.4.8), and
6. dynamically changing objective functions (Section 1.4.9).

With the exception of the NK fitness landscape, it remains unclear to which degrees these phenomena occur in the test problem. How much intrinsic epistasis does the Artificial Ant or the GCD problem emit? What is the quantity of neutrality inherent in Royal Road for variable-length representations? Are the mathematical test functions rugged and, if so, to which degree? All the problems are useful test instances for global optimization. They have not been designed to give us answers to questions like: Which fitness assignment process can be useful when an optimization problem exhibits weak causality and thus has a rugged fitness landscape? How does a certain selection algorithm influence the ability of a genetic algorithm to deal with neutrality? Only Kauffman's NK landscape provides such answers, but only for epistatis. By fine-tuning its N and K parameters, we can generate problems with different degrees of epistatis. Applying a genetic algorithm to these problems then allows us to draw conclusions on its expected performance when being fed with high or low epistatic real-world problems.

In this section, a new model problem is defined that exhibits ruggedness, epistasis, neutrality, multi-objectivity, overfitting, and oversimplification features in a controllable manner Weise et al. [2185], Niemczyk [1533]. Each of them is introduced as a distinct filter component which can separately be activated, deactivated, and fine-tuned. This model provides a perfect test bed for optimization algorithms and their configuration settings. Based on a rough estimation of the structure of the fitness landscape of a given problem, tests can be run very fast using the model as a benchmark for the settings of an optimization algorithm. Thus, we could, for instance, determine a priori whether increasing the population size of an evolutionary algorithm over an approximated limit is likely to provide a gain.

With it, we also can evaluate the behavior of an optimization method in the presence of various problematic aspects, like epistasis or neutrality. This way, strengths and weaknesses of different evolutionary approaches could be explored in a systematic manner. Additionally, it is also well suited for theoretical analysis because of its simplicity. The layers of the model, sketched using an example in Figure 21.6, are specified in the following.

Model Definition

The basic optimization task in this model is to find a binary string $\mathbf{x}^\star$ of a predefined length $n = \text{len}(\mathbf{x}^\star)$ consisting of alternating zeros and ones in the space of all possible binary strings $\mathbb{X} = \mathbb{B}^*$. The tuning parameter for the problem size is $n \in \mathbb{N}$.

$$\mathbf{x}^\star = 0101010101010\ldots01 \tag{21.27}$$

Overfitting and Oversimplification

Searching this optimal string could be done by comparing each genotype g with $\mathbf{x}^\star$. Therefore we would use the Hamming distance[9] [882] $\text{dist}_{Ham}(a, b)$, which defines the difference

[9] Definition 29.6 on page 537 includes the specification of the Hamming distance.

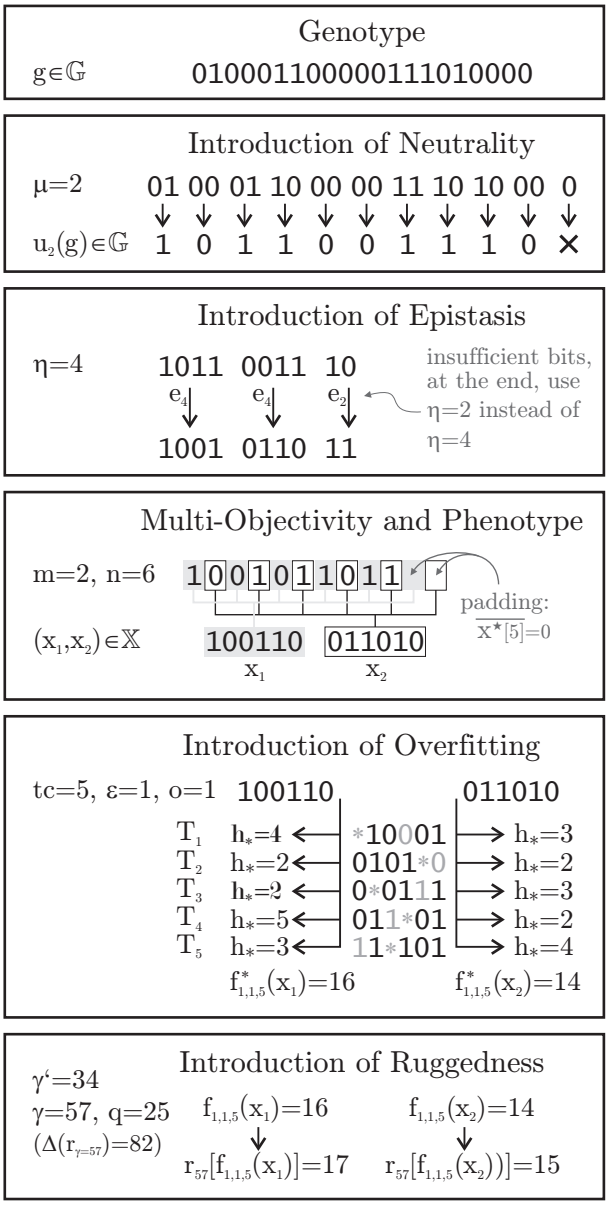


Figure 21.6: An example for the fitness landscape model.

between two binary strings a and b of equal length as the number of bits in which they differ (see Equation 29.10).

Instead of doing this directly, we test the solution candidate against tc training samples $T_1, T_2, .., T_{tc}$. These samples are modified versions of the perfect string $\mathbf{x}^\star$.

As outlined in Section 1.4.8 on page 72, we can distinguish between overfitting and oversimplification. The latter is often caused by incompleteness of the training cases and the former can originate from noise in the training data. Both forms can be expressed in terms of our model by the objective function $f_{\varepsilon,o,tc}$ (based on a slightly extended version of the Hamming distance dist^*_{Ham}) which is subject to minimization.

$$\text{dist}^*_{Ham}(a,b) = |\{\forall i : (a_{[i]} \neq b_{[i]}) \wedge (b_{[i]} \neq *) \wedge (0 \leq i < |a|)\}| \tag{21.28}$$

$$f_{\varepsilon,o,tc}(x) = \sum_{i=1}^{tc} \text{dist}^*_{Ham}(x, T_i)\,, \quad f_{\varepsilon,o,tc}(x) \in [0, \hat{f}]\ \forall x \in \mathbb{X} \tag{21.29}$$

In the case of oversimplification, the perfect solution $\mathbf{x}^\star$ will always reach a perfect score in all training cases. There may be incorrect solutions reaching this value in some cases too, because some of the facets of the problem are hidden. We take this into consideration by placing *o don't care* symbols ($*$) uniformly distributed into the training cases. The values of the solution candidates at their loci have no influence on the fitness.

When overfitting is enabled, the perfect solution will not reach the optimal score in any training case because of the noise present. Incorrect solutions may score better in some cases and even outperform the real solution if the noise level is high. Noise is introduced in the training cases by toggling ε of the remaining $n - o$ bits, again following a uniform distribution. An optimization algorithm can find a correct solution only if there are more training samples with correctly defined values for each locus than with wrong or don't care values.

The optimal objective value is zero and the maximum $\hat{f}$ of $f_{\varepsilon,o,tc}$ is limited by the upper boundary $\hat{f} \leq (n-o)tc$. Its exact value depends on the training cases. For each bit index i, we have to take into account whether a zero or a one in the phenotype would create larger errors:

$$count(i, val) = |\{j \in 1..n : T_{j[i]} = val\}| \tag{21.30}$$

$$e(i) = \begin{cases} count(i,1) & \text{if } count(i,1) \geq count(i,0) \\ count(i,0) & \text{otherwise} \end{cases} \tag{21.31}$$

$$\hat{f} = \sum_{i=1}^{tc} e(i) \tag{21.32}$$

Neutrality

We can create a well-defined amount of neutrality during the genotype-phenotype mapping by applying a transformation u_μ that shortens the solution candidates by a factor μ. The i^{th} bit in $u_\mu(g)$ is defined as 0 if and only if the majority of the μ bits starting at locus $i * \mu$ in g is also 0, and as 1 otherwise. The default value 1 set in draw situations has (in average) no effect on the fitness since the target solution $\mathbf{x}^\star$ is defined as a sequence of alternating zeros and ones. If the length of a genotype g is not a multiple of μ, the remaining $\text{len}(g) \bmod \mu$ bits are ignored. The tunable parameter for the neutrality in our model is μ. If μ is set to 1, no additional neutrality is modeled.

Epistasis

Epistasis in general means that a slight change in one gene of a genotype influences some other genes. We can introduce epistasis in our model as part of the genotype mapping and apply it after the neutrality transformation. We therefore define a bijective function e_η that translates a binary string z of length η to a binary string $e_\eta(z)$ of the same length. Assume we have two binary strings z_1 and z_2 which differ only in one single locus, i.e., their Hamming distance is one. $e_\eta(i)$ntroduces epistasis by exhibiting the following property:

$$\text{dist}_{Ham}(z_1, z_2) = 1 \Rightarrow \text{dist}_{Ham}(e_\eta(z_1), e_\eta(z_2)) \geq \eta - 1\ \forall z_1, z_2 \in \mathbb{B}^\eta \tag{21.33}$$

The meaning of Equation 21.33 is that a change of one bit in a genotype g leads to the change of at least $\eta - 1$ bits in the corresponding mapping $e_\eta(x)$. This, as well as the demand for bijectivity, is provided if we define e_η as follows:

$$e_\eta(z)_{[i]} = \begin{cases} \bigotimes_{\substack{\forall j: 0 \le j < \eta \wedge \\ j \neq (i-1) \bmod \eta}} z_{[j]} & \text{if } 0 \le z < 2^{\eta-1}, i.\,e.,\ z_{[\eta-1]} = 0 \\ \overline{e_\eta(z - 2^{\eta-1})_{[i]}} & \text{otherwise} \end{cases} \tag{21.34}$$

In other words, for all strings $z \in \mathbb{B}^\eta$ which have the most significant bit (MSB) not set, the e_η transformation is performed bitwise. The i^{th} bit in $e_\eta(z)$ equals the exclusive or combination of all but one bit in z. Hence, each bit in z influences the value of $\eta - 1$ bits in $e_\eta(z)$. For all z with 1 in the MSB, $e_\eta(z)$ is simply set to the negated e_η transformation of z with the MSB cleared (the value of the MSB is $2^{\eta-1}$). This division in e is needed in order to ensure its bijectiveness. This and the compliance with Equation 21.33 can be shown with a rather lengthy proof omitted here.

In order to introduce this model of epistasis in genotypes of arbitrary length, we divide them into blocks of the length η and transform each of them separately with e_η. If the length of a given genotype g is no multiple of η, the remaining $\text{len}(g) \bmod \eta$ bits at the end will be transformed with the function $e_{\text{len}(g) \bmod \eta}$ instead of e_η, as outlined in Figure 21.6. It may also be an interesting fact that the e_η transformations are a special case of the NK landscape discussed in Section 21.2.1 with $N = \eta$ and $K \approx \eta - 2$.

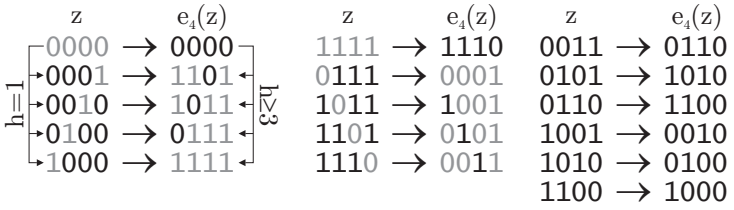


Figure 21.7: An example for the epistasis mapping $z \to e_4(z)$.

The tunable parameter η for the epistasis ranges from 2 to $n * m$, the product of the basic problem length n and the number of objectives m (see next section). If it is set to a value smaller than 3, no additional epistasis is introduced. Figure 21.7 outlines the mapping for $\eta = 4$.

Multi-Objectivity

A multi-objective problem with m criteria can easily be created by interleaving m instances of the benchmark problem with each other and introducing separate objective functions for each of them. Instead of just dividing the genotype g in m blocks, each standing for one objective, we scatter the objectives as illustrated in Figure 21.6. The bits for the first objective function comprise $x_1 = (g_{[0]}, g_{[m]}, g_{[2m]}, \dots)$, those used by the second objective function are $x_2 = (g_{[1]}, g_{[m+1]}, g_{[2m+1]}, \dots)$. Notice that no bit in g is used by more than one objective function. Superfluous bits (beyond index $nm - 1$) are ignored. If g is too short, the missing bits in the phenotypes are replaced with the complement from $\mathbf{x}^\star$, i. e., if one objective misses the last bit (index $n - 1$), it is padded by $\overline{\mathbf{x}^\star{}_{[n-1]}}$ which will worsen the objective by 1 on average.

Because of the interleaving, the objectives will begin to conflict if epistasis ($\eta > 2$) is applied, similar to NK landscapes. Changing one bit in the genotype will change the outcome of at most $\min\{\eta, m\}$ objectives. Some of them may improve while others may worsen.

A non-functional objective function minimizing the length of the genotypes is added if variable-length genomes are used during the evolution. If fixed-length genomes are used, they can be designed in a way that the blocks for the single objectives have always the right length.

Ruggedness

In an optimization problem, there can be at least two (possibly interacting) sources of ruggedness of the fitness landscape. The first one, epistasis, has already been modeled and discussed. The other source concerns the objective functions themselves, the nature of the problem. We will introduce this type of ruggedness *a posteriori* by artificially lowering the causality of the problem space. We therefore shuffle the objective values with a permutation $r : [0, \hat{f}] \mapsto [0, \hat{f}]$, where $\hat{f}$ the abbreviation for the maximum possible objective value, as defined in Equation 21.32.

Before we do that, let us shortly outline what makes a function *rugged*. Ruggedness is obviously the opposite of smoothness and causality. In a smooth objective function, the objective values of the solution candidates neighboring in problem space are also neighboring. In our original problem with $o = 0$, $\varepsilon = 0$, and $tc = 1$ for instance, two individuals differing in one bit will also differ by one in their objective values. We can write down the list of objective values the solution candidates will take on when they are stepwise improved from the worst to the best possible configuration as $\left(\hat{f}, \hat{f}-1, .., 2, 1, 0\right)$. If we exchange two of the values in this list, we will create some artificial ruggedness. A measure for the ruggedness of such a permutation r is $\Delta(r)$:

$$\Delta(r) = \sum_{i=0}^{\hat{f}-1} |r_{[i]} - r_{[i+1]}| \tag{21.35}$$

The original sequence of objective values has the minimum value $\check{\Delta} = \hat{f}$ and the maximum possible value is $\hat{\Delta} = \frac{\hat{f}(\hat{f}+1)}{2}$. There exists at least one permutation for each Δ value in $\check{\Delta}..\hat{\Delta}$. We can hence define the permutation r_γ which is applied after the objective values are computed and which has the following features:

1. It is bijective (since it is a permutation).
2. It must preserve the optimal value, i. e., $r_{\gamma[0]} = 0$.
3. $\Delta(r_\gamma) = \check{\Delta} + \gamma$.

With $\gamma \in \left[0, \hat{\Delta} - \check{\Delta}\right]$, we can fine-tune the ruggedness. For $\gamma = 0$, no ruggedness is introduced. For a given $\hat{f}$, we can compute the permutations r_γ with the procedure buildRPermutation$\left(\gamma, \hat{f}\right)$ defined in Algorithm 21.2.

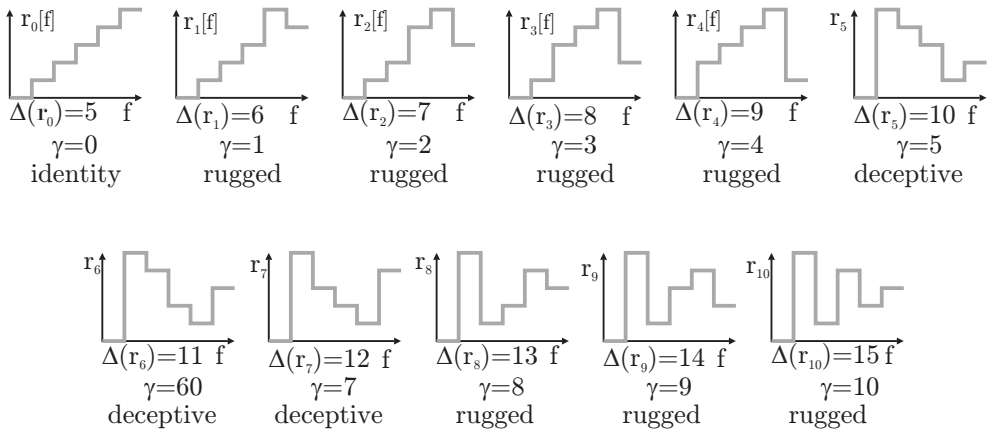


Figure 21.8: An example for r_γ with $\gamma = 0..10$ and $\hat{f} = 5$.

Algorithm 21.2: $r_\gamma \longleftarrow \text{buildRPermutation}\left(\gamma, \hat{f}\right)$

Input: γ: the γ value
Input: $\hat{f}$: the maximum objective value
Data: i, j, d, tmp: temporary variables
Data: $k, start, r$: parameters of the subalgorithm
Output: r_γ: the permutation r_γ

```
1  begin
2      Subalgorithm r ⟵ permutate(k, r, start)
3      begin
4          if k > 0 then
5              if k ≤ (f̂ − 1) then
6                  r ⟵ permutate(k − 1, r, start)
7                  tmp ⟵ r[f̂]
8                  r[f̂] ⟵ r[f̂−k]
9                  r[f̂−k] ⟵ tmp
10             else
11                 i ⟵ ⌊(start+1)/2⌋
12                 if (start mod 2) = 0 then
13                     i ⟵ f̂ + 1 − i
14                     d ⟵ −1
15                 else
16                     d ⟵ 1
17                 for j ⟵ start up to f̂ do
18                     r[j] ⟵ i
19                     i ⟵ i + d
20                 r ⟵ permutate(k − f̂ + start, r, start + 1)
21     end
22     r ⟵ (0, 1, 2, .., f̂ − 1, f̂)
23     return permutate(γ, r, 1)
24 end
```

Figure 21.8 outlines all ruggedness permutations r_γ for an objective function which can range from 0 to $\hat{f} = 5$. As can be seen, the permutations scramble the objective function more and more with rising γ and reduce its gradient information.

Experimental Validation

In this section, we will use a selection of the experimental results obtained with our model in order to validate the correctness of the approach.[10] Table 21.3 states the configuration of the evolutionary algorithm used for our experiments. For each of the experiment-specific settings discussed later, at least 50 runs have been performed.

Parameter	Short	Description
Problem Space	$\mathbb{X}$	The variable-length bit strings consisting of between 1 and 8000 bits. (see Section 3.5)
Objective Functions	$\boldsymbol{F}$	$F = \{f_{\varepsilon,o,tc}, f_{nf}\}$, where f_{nf} is the non-functional length criterion $f_{nf}(x) = \text{len}(x)$ (see Equation 21.29)

[10] More experimental results and more elaborate discussions can be found in the bachelor's thesis of Niemczyk [1533].

Search Space	$\mathbb{G}$	$\mathbb{G} = \mathbb{X}$
Search Operations	***Op***	$cr = 80\%$ single-point crossover, $mr = 20\%$ single-bit mutation
GPM	**gpm**	(see Section 21.2.7)
Optimization Algorithm	***alg***	plain genetic algorithm (see Chapter 3)
Comparison Operator	***cm***	Pareto comparison (see Section 1.2.2)
Population Size	***ps***	$ps = 1000$
Steady-State	***ss***	The algorithms were generational (not steady-state) ($ss = 0$). (see Section 2.1.6)
Fitness Assignment Algorithm	***fa***	For fitness assignment in the evolutionary algorithm, Pareto ranking was used. (see Section 2.3.3)
Selection Algorithm	***sel***	A tournament selection with tournament size $k = 5$ was applied. (see Section 2.4.4)
Convergence Prevention	***cp***	No additional means for convergence prevention were used, i. e., $cp = 0$. (see Section 2.4.8)
Generation Limit	***mxt***	The maximum number of generations that each run is allowed to perform. (see Definition 1.43) $mxt = 1001$

Table 21.3: The settings of the experiments with the benchmark model.

Basic Complexity

In the experiments, we distinguish between *success* and *perfection*. Success means finding individuals x of optimal *functional* fitness, i.e., $f_{\varepsilon,o,tc}(x) = 0$. Multiple such *successful* strings may exist, since superfluous bits at the end of genotypes do not influence their functional objective. We will refer to the number of generations needed to find a successful individual as *success generations*. The perfect string $\mathbf{x}^\star$ has no useless bits, it is the shortest possible solution with $f_{\varepsilon,o,tc} = 0$ and, hence, also optimal in the non-functional length criterion. In our experiments, we measure:

Measure	Short	Description
Success Fraction	$s_{/r}$	The fraction of experimental runs that turned out successful. (see Section 20.3.1)
Minimum Success Generation	$\breve{st}$	The number of generations needed by the fastest (successful) experimental run to find a successful individual. (see Section 20.3.1)
Mean Success Generation	$\overline{st}$	The average number of generations needed by the (successful) experimental runs to find a successful individual. (see Section 20.3.1)
Maximum Success Generation	$\widehat{st}$	The number of Generations generations needed by the slowest (successful) experimental run to find a successful individual. (see Section 20.3.1)
Mean Perfection Generation	$\overline{pt}$	The average number of generations needed by the (perfect) experimental runs to find a perfect individual. equ:experimentPerfAvgGen

Table 21.4: First-level evaluation results of the experiments with the model benchmark.

In Figure 21.9, we have computed the minimum, average, and maximum number of the success generations ($\breve{st}$, $\overline{st}$, and $\widehat{st}$) for values of n ranging from 8 to 800. As illustrated, the problem hardness increases steadily with rising string length n. Trimming down the solution strings to the perfect length becomes more and more complicated with growing n. This is

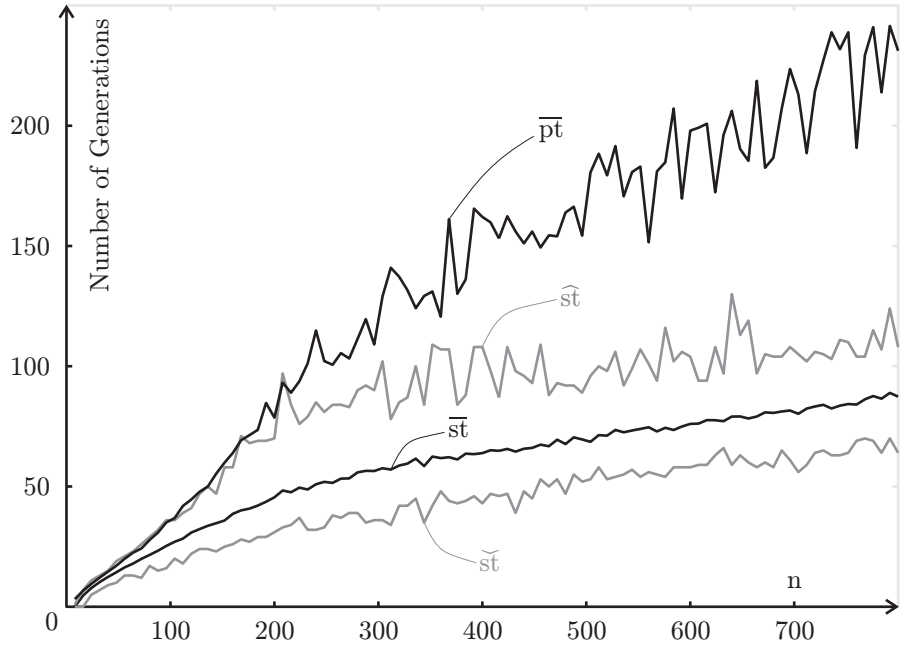


Figure 21.9: The basic problem hardness.

likely because the fraction at the end of the strings where the trimming is to be performed will shrinks in comparison with its overall length.

Ruggedness

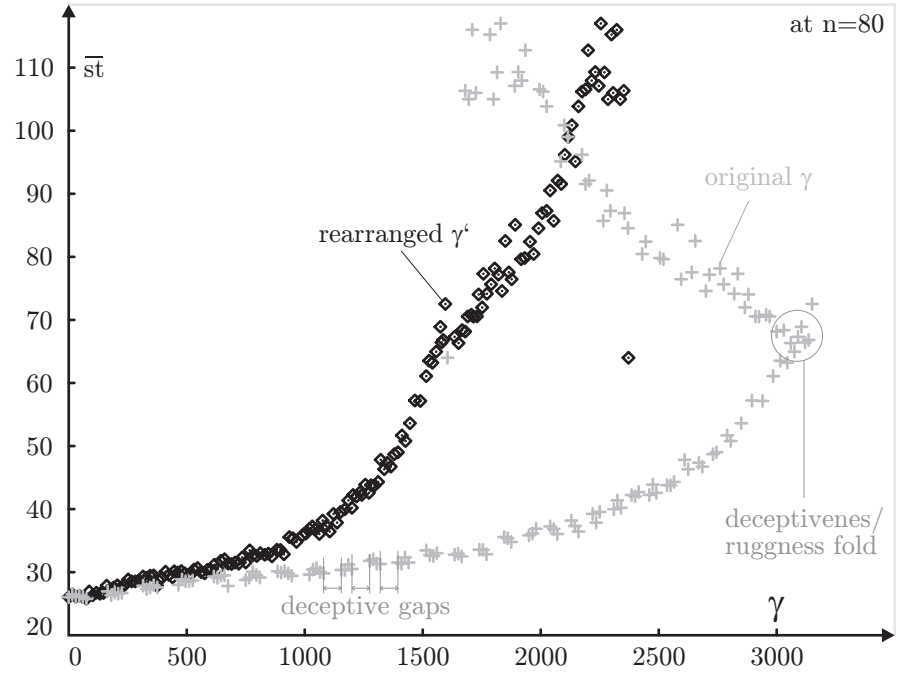


Figure 21.10: Experimental results for the ruggedness.

In Figure 21.10, we plotted the average success generations $\overline{st}$ with $n = 80$ and different ruggedness settings γ. Interestingly, the gray original curve behaves very strangely. It is divided into alternating solvable and unsolvable[11] problems. The unsolvable ranges of γ correspond to gaps in the curve. With rising γ, the solvable problems require more and more generations until they are solved. After a certain (earlier) γ threshold value, the unsolvable sections become solvable. From there on, they become simpler with rising γ. At some point, the two parts of the curve meet.

Algorithm 21.3: $\gamma \longleftarrow \text{translate}\left(\gamma', \hat{f}\right)$

Input: γ': the raw γ value
Input: $\hat{f}$: the maximum value of $f_{\varepsilon,o,tc}$
Data: i, j, k, l: some temporary variables
Output: γ: the translated γ value

```
1  begin
2      l ⟵ f̂(f̂−1)/2
3      i ⟵ ⌊f̂/2⌋ * ⌊(f̂+1)/2⌋
4      if γ ≤ f̂i then
5          j ⟵ ⌊(f̂+2)/2 − √(f̂²/4 + 1 − γ)⌋
6          k ⟵ γ − j(f̂ + 2) + j² + f̂
7          return k + 2(j(f̂ + 2) − j² − f̂) − j
8      else
9          j ⟵ ⌊((f̂ mod 2)+1)/2 + √((1−(f̂ mod 2))/4 + γ − 1 − i)⌋
10         k ⟵ γ − (j − (f̂ mod 2))(j − 1) − 1 − i
11         return l − k − 2j² + j − (f̂ mod 2)(−2j + 1)
12 end
```

The reason for this behavior is rooted in the way that we construct the ruggedness mapping r and illustrates the close relation between ruggedness and deceptiveness. Algorithm 21.2 is a greedy algorithm which alternates between creating groups of mappings that are mainly rugged and such that are mainly deceptive. In Figure 21.8 for instance, from $\gamma = 5$ to $\gamma = 7$, the permutations exhibit a high degree of deceptiveness whilst just being rugged before and after that range. Thus, it seems to be a good idea to rearrange these sections of the ruggedness mapping. The identity mapping should still come first, followed by the purely rugged mappings ordered by their Δ-values. Then, the permutations should gradually change from rugged to deceptive and the last mapping should be the most deceptive one ($\gamma = 10$ in Figure 21.8). The black curve in Figure 21.10 depicts the results of rearranging the γ-values with Algorithm 21.3. This algorithm maps deceptive gaps to higher γ-values and, by doing so, makes the resulting curve continuous.[12]

Fig. 21.11.a sketches the average success generations for the rearranged ruggedness problem for multiple values of n and γ'. Depending on the basic problem size n, the problem hardness increases steeply with rising values of γ'.

In Algorithm 21.2 and Algorithm 21.3, we use the maximum value of the functional objective function (abbreviated with $\hat{f}$) in order to build and to rearrange the ruggedness permutations r. Since this value depends on the basic problem length n, the number of

[11] We call a problem unsolvable if it has not been solved within 1000 generations.
[12] This is a deviation from our original idea, but this idea did not consider deceptiveness.

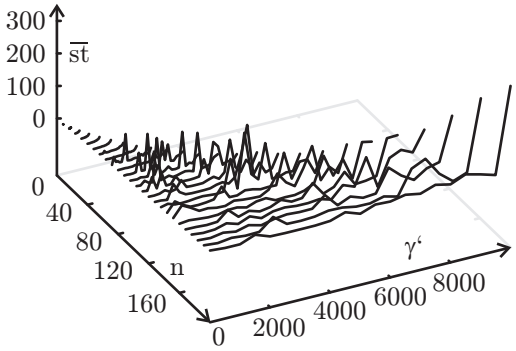


Fig. 21.11.a: Unscaled ruggedness $\overline{st}$ plot.

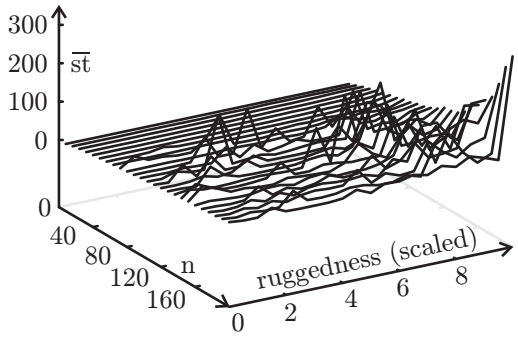


Fig. 21.11.b: Scaled ruggedness $\overline{st}$ plot.

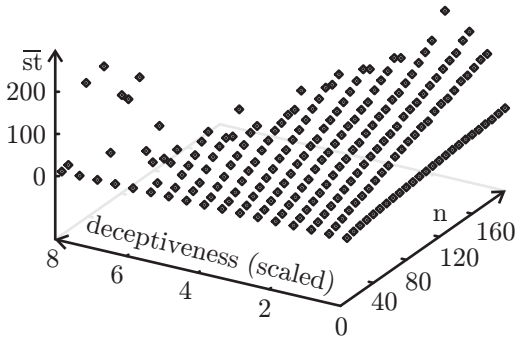


Fig. 21.11.c: Scaled deceptiveness – average success generations $\overline{st}$

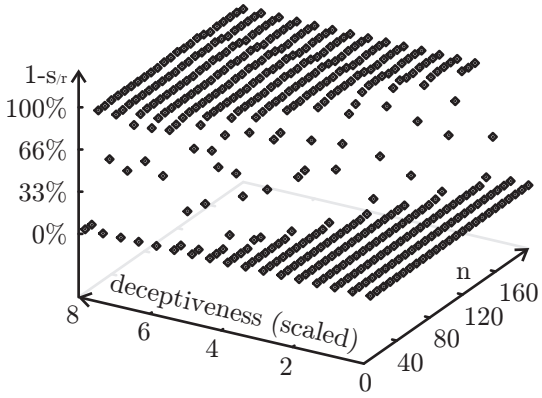


Fig. 21.11.d: Scaled deceptiveness – failed runs

Figure 21.11: Experiments with ruggedness and deceptiveness.

different permutations and thus, the range of the γ' values will too. The length of the lines in direction of the γ' axis in Fig. 21.11.a thus increases with n. We introduce two additional scaling functions for ruggedness and deceptiveness with a parameter g spanning from zero to ten, regardless of n. Only one of these functions can be used at a time, depending on whether experiments should be run for rugged (Equation 21.36) or for deceptive (Equation 21.37) problems. For scaling, we use the highest γ' value which maps to rugged mappings $\gamma'_r = \left\lfloor 0.5\hat{f} \right\rfloor * \left\lceil 0.5\hat{f} \right\rceil$, and the minimum and maximum ruggedness values according to Equation 21.35.

$$\textit{rugged:}\quad \gamma' = \text{round}(0.1g * \gamma'_r) \tag{21.36}$$

$$\textit{deceptive:}\quad \gamma' = \begin{cases} 0 \text{ if } g \leq 0 \\ \gamma'_r + \text{round}\Big(0.1g * \Big(\hat{\Delta} - \check{\Delta} - \gamma'_r\Big)\Big) \text{ otherwise} \end{cases} \tag{21.37}$$

When using this scaling mechanism, the curves resulting from experiments with different n-values can be compared more easily: Fig. 21.11.b based on the scale from Equation 21.36, for instance, shows much clearer how the problem difficulty rises with increasing ruggedness than Fig. 21.11.a. We also can spot some irregularities which always occur at about the same degree of ruggedness, near $g \approx 9.5$, and that we will investigate in future.

The experiments with the deceptiveness scale Equation 21.37 show the tremendous effect of deceptiveness in the fitness landscape. Not only does the problem hardness rise steeply

with g (Fig. 21.11.c), after certain threshold, the evolutionary algorithm becomes unable to solve the model problem at all (in 1000 generations), and the fraction of failed experiments in Fig. 21.11.d jumps to 100% (since the fraction $s_{/r}$ of solved ones goes to zero).

Epistasis

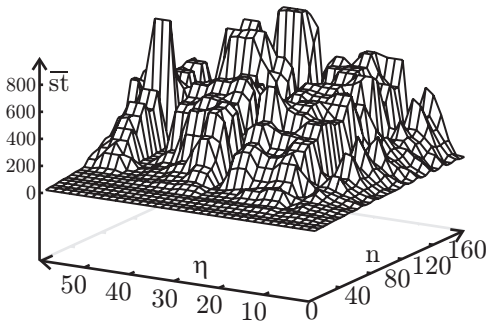


Fig. 21.12.a: Epistasis η and problem length n.

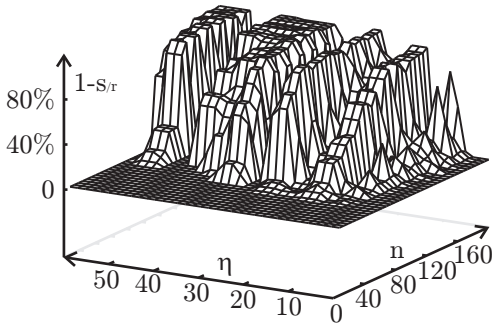


Fig. 21.12.b: Epistasis η and problem length n: failed runs.

Figure 21.12: Experiments with epistasis.

Fig. 21.12.a illustrates the relation between problem size n, the epistasis factor η, and the average success generations. Although rising epistasis makes the problems harder, the complexity does not rise as smoothly as in the previous experiments. The cause for this is likely the presence of crossover – if mutation was allowed solely, the impact of epistasis would most likely be more intense. Another interesting fact is that experimental settings with odd values of η tend to be much more complex than those with even ones. This relation becomes even more obvious in Fig. 21.12.b, where the proportion of failed runs, i. e., those which were not able to solve problem in less than 1000 generations, is plotted. A high plateau for greater values of η is cut by deep valleys at positions where $\eta = 2 + 2i \; \forall i \in \mathbb{N}$. This phenomenon has thoroughly been discussed by Niemczyk [1533] and can be excluded from the experiments by applying the scaling mechanism with parameter $y \in [0, 10]$ as defined in Equation 21.38:

$$\textit{epistasis: } \eta = \begin{cases} y \leq 0 & \text{if } 2 \\ y \geq 10 & \text{if } 41 \\ 2\lfloor 2y \rfloor + 1 & \text{otherwise} \end{cases} \tag{21.38}$$

Neutrality

Figure 21.13 illustrates the average number of generations $\overline{st}$ needed to grow an individual with optimal functional fitness for different values of the neutrality parameter μ. Until $\mu \approx 10$, the problem hardness increases rapidly. For larger degrees of redundancy, only minor increments in $\overline{st}$ can be observed. The reason for this strange behavior seems to be the crossover operation. Niemczyk [1533] shows that a lower crossover rate makes experiments involving the neutrality filter of the model problem very hard. We recommend using only μ-values in the range from zero to eleven for testing the capability of optimization methods of dealing with neutrality.

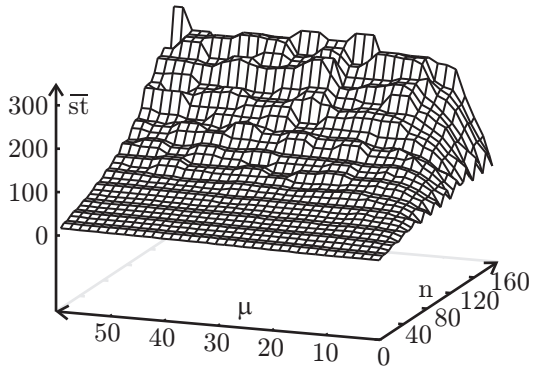


Figure 21.13: The results from experiments with neutrality.

Epistasis and Neutrality

Our model problem consists of independent filters for the properties that may influence the hardness of an optimization task. It is especially interesting to find out whether these filters can be combined arbitrarily, i. e., if they are indeed free of interaction. In the ideal case, $\overline{st}$ of an experiment with $n = 80$, $\mu = 8$, and $\eta = 0$ added to $\overline{st}$ of an experiment for $n = 80$, $\mu = 0$, and $\eta = 4$ should roughly equal to $\overline{st}$ of an experiment with $n = 80$, $\mu = 8$, and $\eta = 4$. In Figure 21.14, we have sketched these expected values (Fig. 21.14.a) and the results of the corresponding real experiments (Fig. 21.14.b). In fact, these two diagrams are very similar. The small valleys caused by the "easier" values of η (see Section 21.2.7) occur in both charts. The only difference is a stronger influence of the degree of neutrality.

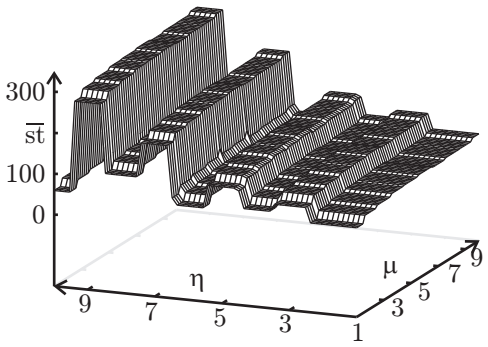


Fig. 21.14.a: The expected results.

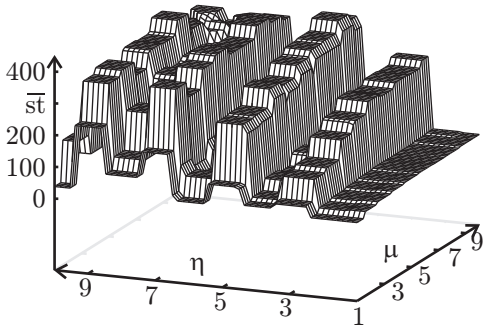


Fig. 21.14.b: The experimentally obtained results.

Figure 21.14: Expection and reality: Experiments involving both, epistasis and neutrality

Ruggedness and Epistasis

It is a well-known fact that epistasis leads to ruggedness, since it violates the causality as discussed in Section 1.4.6. Combining the ruggedness and the epistasis filter therefore leads to stronger interactions. In Fig. 21.15.b, the influence of ruggedness seems to be amplified by the presence of epistasis when compared with the estimated results shown in Fig. 21.15.a. Apart from this increase in problem hardness, the model problem behaves as expected. The characteristic valleys stemming from the epistasis filter are clearly visible, for instance.

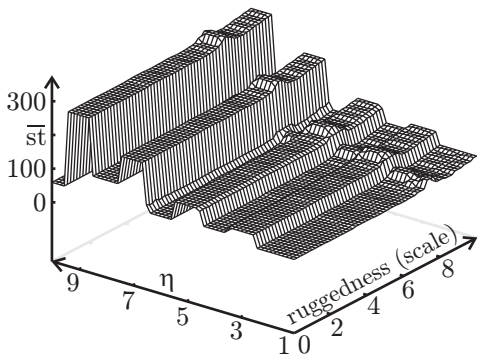


Fig. 21.15.a: The expected results.

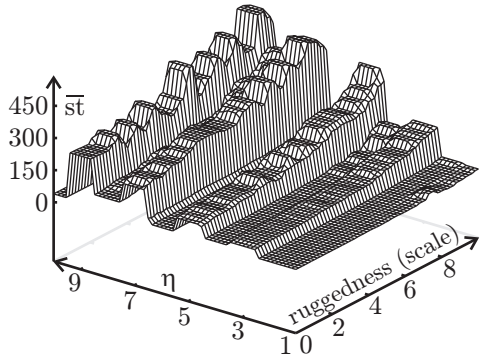


Fig. 21.15.b: The experimentally obtained results.

Figure 21.15: Expection and reality: Experiments involving both, ruggedness and epistasis

Summary

In summary, this model problem has proven to be a viable approach for simulating problematic phenomena in optimization. It is

1. functional, i. e., allows us to simulate many problematic features,
2. tunable – each filter can be tuned independently,
3. easy to understand,
4. allows for very fast fitness computation,
5. easily extensible – each filter can be replaced with other approaches for simulating the same feature.

Niemczyk [1533] has written a stand-alone Java class implementing the model problem which is provided at `http://www.sigoa.org/documents/` [accessed 2008-05-17] and `http://www.it-weise.de/documents/files/TunableModel.java` [accessed 2008-05-17]. This class allows setting the parameters discussed in this section and provides methods for determining the objective values of individuals in the form of `byte` arrays. In the future, some strange behaviors (like the irregularities in the ruggedness filter and the gaps in epistasis) of the model need to be revisited, explained, and, if possible, removed.

21.3 Genetic Programming Problems

21.3.1 Artificial Ant

We already have discussed parts of the Artificial Ant problem in Section 1.2.2 on page 27 – here we are going to investigate it more thoroughly. The goal of the original problem defined by Collins and Jefferson [431, 1046, 433] was to find a program that controls an artificial ant in a simulated environment. Such environments usually have the following features:

1. It is divided in a toroidal grid generating rectangular cells in the plane making the positions of coordinates of all objects discrete.
2. There exists exactly one ant in the environment.
3. The ant will always be inside one cell at one time.
4. A cell can either contain one piece of food or not.

The ant is a very simple life form. It always faces in one of the four directions north, east, south, or west. Furthermore, it can sense if there is food in the next cell in the direction it faces. It cannot sense if there is food on any other cell in the map.

Like space, the time in the Artificial Ant problem is also discrete. The ant may carry out one of the following actions per time unit:

1. The ant can move for exactly one cell into the direction it faces. If this cell contains food, the ant consumes it in the very moment in which it enters the cell.
2. The ant may turn left or right by 90.
3. The ant may do nothing in a time unit.

Many researchers such as Collins and Jefferson [431, 1046, 433], Koza [1196], Lee and Wong [1268], Harries and Smith [900], Luke and Spector [1322], Kuscu [1226], Chellapilla [384], Ito et al. [1024], Langdon and Poli [1240], and Frey [749] have ever since used the Artificial Ant problem as benchmark in their research. Since the Artificial Ant problem neither imposes a special genome, phenome, nor otherwise restricts the parameters of the optimization process, it is the ideal environment for such tests. In order to make the benchmark results comparable, special instances of the problem like the *Santa Fe Trail* with well-defined features have been defined.

Santa Fe trail

One instance of the artificial ant problem is the "Santa Fe trail" sketched in Figure 21.16 designed by Langdon [1196]. It is a map of $32 * 32$ cells containing 89 food pellets distributed along a certain route. Initially, the ant will be placed in the upper left corner of the field facing east. In trail of food pellets, there are gaps of five forms:

1. one cells along a straight line
2. two cells along a straight line
3. one cell in a corner
4. two cells at a corner (requiring something like a "horse jump" in chess)
5. three cells at a corner

The goal is here to find some form of control for the ant that allows it to eat as many of the food pellets as possible (the maximum is 89) and to walk a distance as short as possible in order to do so (the optimal route is illustrated in Figure 21.16). Of course, there will be a time limit set for the ant to perform this task (normally 200 time units).

Solutions

Genetic Algorithm evolving Finite State Machines

Jefferson et al. [1046] applied a conventional genetic algorithm that evolved finite state machines encoded in a fixed-length binary string genome to the Artificial Ant problem. The sensor information together with the current state determines the next state, therefore a finite state machine with at most m states can be encoded in a chromosome using $2m$ genes. In order to understand the structure of such a chromosome, let us assume that $m = 2^n$. We then can specify the finite state machine as a table where $n + 1$ bits are used as row index. n of these indexes identify the current state and one bit is used for the sensor information (`1`=food ahead, `0`=no food ahead). In total, there are $2m$ rows. There is no need to store the row indices, just the cell contents: n bits encode the next state, and two bits encode the action to be performed at the state transition (`00` for nothing, `01` for turning left, `10` for turning right, `11` for moving). A chromosome encoding a finite state machine with m states can be encoded in $2m(n + 2) = 2^{n+1}(n + 2)$ bits. If the initial state in the chromosome is also to be stored, another n bits are needed to do so. Every chromosome represents a valid finite state machine.

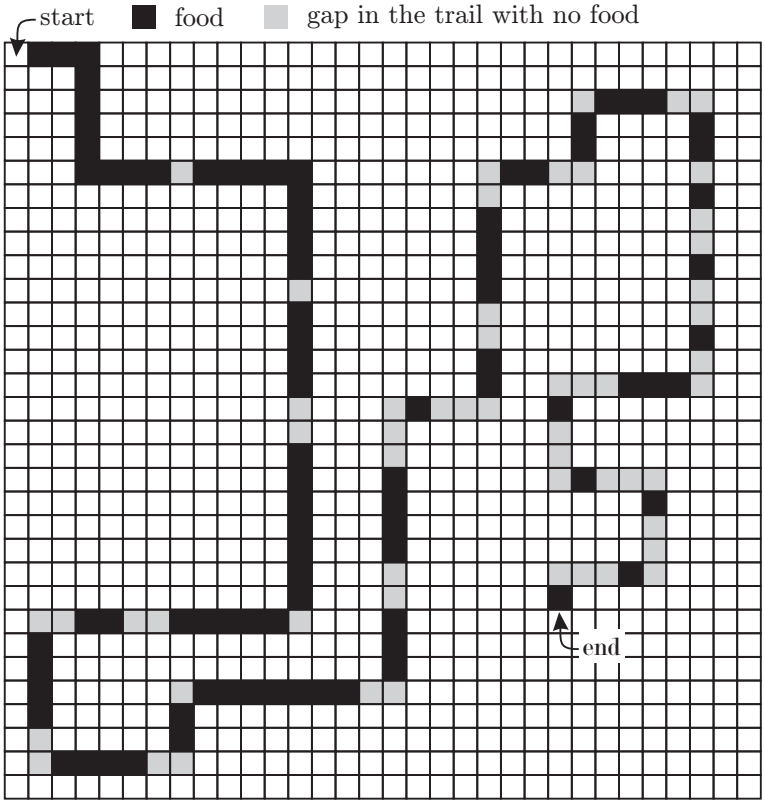


Figure 21.16: The Santa Fee Trail in the Artificial Ant Problem [1196].

Jefferson et al. [1046] allowed for 32 states (453 bit chromosomes) in their finite state machines. They used one objective function that returned the number of food pellets eaten by the ant in a simulation run (of maximal 200 steps) and made it subject to maximization. Using a population of 65 536 individuals, they found one optimal solution (with fitness 89).

Genetic Algorithm evolving Artificial Neural Networks

Collins and Jefferson [431] also evolved an artificial neural network (encoded in a 520 bit genome) with a genetic algorithm of the same population size to successfully solve the Artificial Ant problem. Later, they applied a similar approach [433] with 25 590 bit genomes which allowed even the structure of the artificial neural networks to evolve to a generalized problem exploring the overall behavior of ant colonies from a more biological perspective.

Genetic Programming evolving Control Programs

Koza [1189, 1188, 1187] solved the Artificial Ant problem by evolving LISP[13]-programs. Therefore, he introduced the parameterless instructions `MOVE`, `RIGHT`, and `LEFT` that moved the ant one unit, or turned it right or left respectively. Furthermore, the binary conditional expression `IF-FOOD-AHEAD` executed its first parameter expression if the ant could sense food and the second one otherwise. Two compound instructions, `PROGN2` and `PROGN3`, execute their two or three sub-expressions unconditionally. After 21 generations using a 500 individual population and fitness-proportional selection, Genetic Programming yielded an individual solving the Santa Fe trail optimally. Connolly [435] provides an easy step-by-step guide for solving the Artificial Ant problem with Genetic Programming (and for visualizing this process).

[13] http://en.wikipedia.org/wiki/Lisp_%28programming_language%29 [accessed 2007-07-03]

21.3.2 The Greatest Common Divisor

Another problem suitable to test Genetic Programming approaches is to evolve an algorithm that computes the greatest common divisor[14], the GCD.

Problem Definition

Definition 21.1 (GCD). For two integer numbers $a, b \in \mathbb{N}_0$, the greatest common divisor (GCD) is the largest number $c \in \mathbb{N}$ that divides both, a ($c \,|\, a \equiv a \bmod c = 0$) and b ($c \,|\, b \equiv b \bmod c = 0$).

$$c = \gcd(a, b) \Leftrightarrow c \,|\, a \wedge c \,|\, b \wedge (\nexists d \in \mathbb{N} : d \,|\, a \wedge d \,|\, b \wedge d > c) \tag{21.39}$$

$$\Leftrightarrow \max\{e \in \mathbb{N} : (a \bmod e = 0) \wedge (b \bmod e = 0)\} \tag{21.40}$$

The Euclidean Algorithm

The GCD can be computed with the Euclidean algorithm[15] which is specified in its original version in Algorithm 21.4 and in the improved, faster variant as Algorithm 21.5 [1559, 913].

Algorithm 21.4: $\gcd(a, b) \longleftarrow \text{euclidGcdOrig}(a, b)$

Input: $a, b \in \mathbb{N}_0$: two integers
Output: $\gcd(a, b)$: the greatest common divisor of a and b

```
1 begin
2     while b ≠ 0 do
3         if a > b then a ⟵ a − b
4         else b ⟵ b − a
5     return a
6 end
```

Algorithm 21.5: $\gcd(a, b) \longleftarrow \text{euclidGcd}(a, b)$

Input: $a, b \in \mathbb{N}_0$: two integers
Data: t: a temporary variable
Output: $\gcd(a, b)$: the greatest common divisor of a and b

```
1 begin
2     while b ≠ 0 do
3         t ⟵ b
4         b ⟵ a mod b
5         a ⟵ t
6     return a
7 end
```

The Objective Functions and the Prevalence Comparator

Although the GCD-problems seems to be more or less trivial since simple algorithms exist that solve it, it has characteristics that make it hard of Genetic Programming. Assume we

[14] http://en.wikipedia.org/wiki/Greatest_common_divisor [accessed 2007-10-05]
[15] http://en.wikipedia.org/wiki/Euclidean_algorithm [accessed 2007-10-05]

have evolved a program $x \in \mathbb{X}$ which takes the two values a and b as input parameters and returns a new value $c = x(a, b)$. Unlike in symbolic regression[16], it makes no sense to define the error between c and the real value $\gcd(a, b)$ as objective function, since there is no relation between the "degree of correctness" of the algorithm and $|c - \gcd(a, b)|$. Matter of fact, we cannot say that a program returning $c_1 = x_1(20, 15) = 6$ is better than $c_2 = x_2(20, 15) = 10$. 6 may be closer to the real result $\gcd(20, 15) = 5$ but shares no divisor with it whereas $5 \,|10 \equiv 10 \bmod 5 = 0$.

Based on the idea that the GCD is of the variables a and b is preserved in each step of the Euclidean algorithm, a suitable functional objective function $f_1 : \mathbb{X} \mapsto [0, 5]$ for this problem is Algorithm 21.6. It takes a training case (a, b) as argument and first checks whether the evolved program $x \in \mathbb{X}$ returns the correct result for it. If so, $f_1(x) = 0$ is returned. Otherwise, we check if the greatest common divisor of $x(a, b)$ and a is still the greatest common divisor of a and b. If this is not the case, 1 is added to the objective value. The same is repeated with $x(a, b)$ and b. Furthermore, negative values of $x(a, b)$ are penalized with 2 and results that are larger or equal to a or b are penalized with one additional point for each violation. Depending on the program representation, this objective function is very rugged because small changes in the program have a large potential impact on the fitness. It also exhibits a large amount of neutrality, since it can take on only integer values between 0 (the optimum) and 5 (the worst case).

Algorithm 21.6: $r \longleftarrow f_1(x, a, b)$

Input: $a, b \in \mathbb{N}_0$: the training case
Input: $x \in \mathbb{X}$: the evolved algorithm to be evaluated
Data: v: a variable holding the result of x for the training case
Output: r: the functional objective value of the functional objective function f_1 for the training case

```
1  begin
2      r ⟵ 0
3      v ⟵ x(a, b)
4      if v ≠ gcd(a, b) then
5          r ⟵ r + 1
6          if gcd(v, a) ≠ gcd(a, b) then r ⟵ r + 1
7          if gcd(v, b) ≠ gcd(a, b) then r ⟵ r + 1
8          if v ≤ 0 then
9              r ⟵ r + 2
10         else
11             if v ≥ a then r ⟵ r + 1
12             if v ≥ b then r ⟵ r + 1
13     return r
14 end
```

Additionally to f_1, two objective functions optimizing non-functional aspects should be present. $f_2(x)$ minimizes the number of expressions in x and $f_3(x)$ minimizes the number of steps x needs until it terminates and returns the result. This way, we further small and fast algorithms. These three objective functions, combined to a prevalence[17] comparator $\text{cmp}_{F,\text{gcd}}$, can serve as a benchmark on how good a Genetic Programming approach can cope with the rugged fitness landscape common to the evolution of real algorithms and how the parameter settings of the evolutionary algorithm influence this ability.

[16] See Section 23.1 for an extensive discussion of symbolic regression.
[17] See Definition 1.17 on page 39 for more information.

$$\text{cmp}_{F,\text{gcd}}(x_1, x_2) = \begin{cases} -1 \text{ if } f_1(x_1) < f_1(x_2) \\ 1 \text{ if } f_1(x_1) > f_1(x_2) \\ \text{cmp}_{F,Pareto}(x_1, x_2) \text{ otherwise} \end{cases} \tag{21.41}$$

In principle, Equation 21.41 gives the functional fitness precedence before any other objective. If (and only if) the functional objective values of both individuals are equal, the prevalence is decided upon a Pareto comparison of the remaining two (non-functional) objectives.

The Training Cases

The structure of the training cases is also very important. If we simply use two random numbers a and b, their greatest common divisor is likely to 1 or 2. Hence, we construct a single training case by first drawing a random number $r \in \mathbb{N}$ uniformly distributed in $[10, 100\,000]$ as lower limit for the GCD and then keep drawing uniformly distributed random numbers $a > r, b > r$ until $\gcd(a, b) \geq r$. Furthermore, if multiple training cases are involved in the individual evaluation, we ensure that they involve different magnitudes of the values of a, b, and r. If we change the training cases after each generation of the evolutionary algorithm, the same goes for two subsequent training case sets. Some typical training sets are noted in Listing 21.2.

```
Generation 0
====================================
a           b           gcd(a,b)
87546096    2012500485  21627
1656382     161406235   9101
7035        5628        1407
2008942236  579260484   972
556527320   1588840     144440
14328736    10746552    3582184
1390        268760      10
929436304   860551      5153
941094      1690414110  1386
14044248    1259211564  53604

Generation 1
====================================
a           b           gcd(a,b)
117140      1194828     23428
2367        42080       263
3236545     379925      65
1796284190  979395390   10
4760        152346030   10
12037362    708102      186
1785869184  2093777664  61581696
782331      42435530    23707
434150199   24453828    63
45509100    7316463     35007

Generation 2
====================================
a           b           gcd(a,b)
1749281113  82          41
25328611    99          11
279351072   2028016224  3627936
173078655   214140      53535
216         126         18
1607646156  583719700   2836
1059261     638524299   21
```

```
70903440    1035432    5256
26576383    19043      139
1349426     596258     31382
```

Listing 21.2: Some training cases for the GCD problem.

Rule-based Genetic Programming

We have conducted a rather large series of experiments on solving the GCD problem with Rule-based Genetic Programming (*RBGP*, see Section 4.8.4 on page 207). In this section, we will elaborate on the different parameters that we have tried out and what results could be observed for these settings.

Settings

As outlined in Table 21.5, we have tried to solve the GCD problem with Rule-based Genetic Programming with a lot of different settings (60 in total) in a factorial experiment. We will discuss these settings here in accordance to Section 20.1.

Parameter	Short	Description
Problem Space	$\mathbb{X}$	The space of Rule-based Genetic Programming-programs with between 2 and 100 rules. (see Section 4.8.4)
Objective Functions	$\boldsymbol{F}$	$F = \{f_1, f_2, f_3\}$ (see Section 21.3.2)
Search Space	$\mathbb{G}$	The variable-length bit strings with a gene size of 34 bits and a length between 64 and 3400 bits.
Search Operations	$\boldsymbol{Op}$	$mr = 30\%$ mutation (including single-bit flits, permutations, gene deletion and insertion), $cr = 70\%$ multi-point crossover
GPM	**gpm**	(see Figure 4.29)
Optimization Algorithm	$\boldsymbol{alg}$	The optimization algorithm applied. $alg = 0 \rightarrow$ evolutionary algorithm $alg = 1 \rightarrow$ Parallel Random Walks
Comparison Operator	$\boldsymbol{cm}$	(see Equation 21.41)
Population Size	$\boldsymbol{ps}$	$ps \in \{512, 1024, 2048\}$
Steady-State	$\boldsymbol{ss}$	The evolutionary algorithms were either steady state ($ss = 1$), meaning that the offspring had to compete with the already existing individuals in the selection phase, or generational/extinctive ($ss = 0$), meaning that only the offspring took part in the selection and the parents were discarded. (see Section 2.1.6) $ss = 0 \rightarrow$ generational $ss = 1 \rightarrow$ steady-state
Fitness Assignment Algorithm	$\boldsymbol{fa}$	For fitness assignment in the evolutionary algorithm, Pareto ranking was used. (see Section 2.3.3)
Selection Algorithm	$\boldsymbol{sel}$	A binary ($k = 2$) tournament selection was applied in the evolutionary algorithm. (see Section 2.4.4)
Convergence Prevention	$\boldsymbol{cp}$	(see Section 2.4.8) $cp \in \{0, 0.3\}$

Number of Training Cases	***tc***	The number of training cases used for evaluating the objective functions. $tc \in \{1, 10\}$
Training Case Change Policy	***ct***	The policy according to which the training cases are changed. $ct = 0 \rightarrow$ The training cases do not change. $ct = 1 \rightarrow$ The training cases change each generation.
Generation Limit	***mxt***	The maximum number of generations that each run is allowed to perform. (see Definition 1.43) $mxt = 501$
System Configuration	***Cfg***	normal off-the-shelf PCs with approximately 2 GHz processor power

Table 21.5: The settings of the RBGP-Genetic Programming experiments for the GCD problem.

Convergence Prevention In our past experiments, we have made the experience that Genetic Programming in rugged fitness landscapes and Genetic Programming of real algorithms (which usually leads to rugged fitness landscapes) is very inclined to converge prematurely. If it finds some half-baked solution, the population often tended to converge to this individual and the evolutions stopped.

There are many ways to prevent this, like modifying the fitness assignment process by using sharing functions (see Section 2.3.4 on page 114), for example. Such methods influence individuals close in objective space and decrease their chance to reproduce. Here, we decided to choose a very simple measure which only decreases probability of reproduction of individuals with exactly equal objective functions: the simple convergence prevention algorithm *SCP* introduced in Section 2.4.8. This filter has either been applied with strength $cp = 0.3$ or not been used ($cp = 0$).

Comparison with Random Walks We found it necessary to compare the Genetic Programming approach for solving this problem with random walks in order to find out whether or not Genetic Programming can provide any advantage in a rugged fitness landscape. Therefore, we either used an evolutionary algorithm with the parameters discussed above ($alg = 0$) or parallel random walks ($alg = 1$). Random walks, in this context, are principally evolutionary algorithms where neither fitness assignment nor selection are preformed. Hence, we can test parameters like *ps*, *ct*, and *tc*, but no convergence prevention ($cp = 0$) and also no steady state ($expSteadyState = 0$). The results of these random walks are the best individuals encountered during their course.

Results

We have determined the following parameters from the data obtained with our experiments.

Measure	Short	Description
Perfection Fraction	$p_{/r}$	The fraction of experimental runs that found perfect individuals. This fraction is also the estimate of the cumulative probability of finding a perfect individual until the 500th generation. (see Section 20.3.1) $p_{/r} = CPp(ps, 500)$ (see Equation 20.20)

Number of Perfect Runs	$\#\boldsymbol{p}$	The number of runs where perfect individuals were discovered. (see Section 20.3.1)
Number of Successful Runs	$\#\boldsymbol{s}$	The number of runs where successful individuals were discovered. (see Section 20.3.1)
Number of Comp. Runs	$\#\boldsymbol{r}$	The total number of completed runs with the specified configuration.
Mean Success Generation	$\overline{\boldsymbol{st}}$	The average number of generations t needed by the (successful) experimental runs to find a successful individual. (or $\emptyset$ if no run was successful) (see Section 20.3.1)
Runs Needed for Perfection	$\boldsymbol{pt_n}$	The estimated number $pt_n(0.99, ps, 500)$ of independent runs needed to find at least one perfect solution candidate with a probability of $z = 0.99$ until the 500th generation. (see Equation 20.21)
Evaluations Needed for Perfection	$\boldsymbol{p\tau_n}$	The estimated number $pt_n(0.99, ps, 500)$ of objective function evaluations runs needed to find at least one perfect solution candidate with a probability of $z = 0.99$ until the 500th generation. (see Section 20.3.2)

Table 21.6: Evaluation parameters used in Table 21.7.

In the context of this experiment, a *perfect* solution represents a correct GCD algorithm, i.e., is not overfitted. Solutions with optimal functional objective values ($f_1 = 0$, whether due to overfitting or not) are called *successful*. Overfitted programs, like the one illustrated in Listing 21.4, will not work with inputs a and b different from those used in their evaluation.

Not all configurations were tested with the same number of runs since we had multiple computers involved in these test series and needed to end it at some point of time. We then used the maximum amount of available data for our evaluation. With the given configurations, the evolutionary algorithm runs usually took about one to ten minutes (depending on the population size). The results of the application of the Rule-based Genetic Programming to the GCD problem are listed in Table 21.7 below.

rank	alg	cp	ss	ct	tc	ps	$p_{/r}$	$\#p$	$\#s$	$\#r$	$\overline{st}$	pt_n	$p\tau_n$
1.	0	0.3	1	0	1	1024	0.28	15	45	53	100.4	13.84	7 086 884
2.	0	0.3	1	0	1	512	0.12	6	35	51	98.5	36.79	9 419 095
3.	1	0	0	0	1	512	0.10	5	27	51	259.1	44.63	11 425 423
4.	0	0.3	1	0	10	2048	0.98	48	48	49	70.0	1.18	12 116 937
5.	1	0	0	0	1	1024	0.17	9	41	54	170.0	25.26	12 932 355
6.	0	0.3	0	0	1	2048	0.27	14	49	51	85.2	14.35	14 694 861
7.	0	0.3	1	0	10	1024	0.78	41	41	53	129.1	3.1	15 873 640
8.	0	0.3	1	0	1	2048	0.25	13	51	51	36.4	15.65	16 026 722
9.	0	0.3	1	0	10	512	0.49	25	25	51	153.0	6.84	17 498 481
10.	0	0.3	0	0	1	512	0.06	3	22	51	162.1	75.96	19 446 283
11.	0	0.3	0	1	10	1024	0.67	37	37	54	199.4	3.98	20 400 648
12.	0	0.3	0	1	1	1024	0.10	5	5	52	197.4	45.55	23 322 826
13.	0	0.3	1	1	10	2048	0.98	53	53	54	61.1	1.15	23 643 586
14.	0	0.3	0	0	1	1024	0.09	5	44	54	138.2	47.4	24 266 737
15.	0	0.3	0	0	10	2048	0.79	39	39	49	111.1	2.9	29 672 727
16.	0	0.3	0	1	10	2048	0.79	41	41	52	101.1	2.96	30 358 251
17.	0	0.3	1	1	10	1024	0.78	42	42	54	125.5	3.06	31 352 737

18.	0	0.3	1	1	1	2048	0.28	14	14	54	107.8	15.35	31 427 005
19.	0	0.3	0	0	10	1024	0.52	27	27	53	196.3	6.47	33 106 746
20.	0	0.3	0	1	10	512	0.25	13	13	52	231.5	16.01	40 980 085
21.	0	0.3	1	1	10	512	0.41	21	21	50	231.5	8.45	43 284 918
22.	0	0.3	0	1	1	2048	0.09	5	5	53	46.6	46.47	47 589 578
23.	0	0.3	0	0	10	512	0.19	10	10	52	250.8	21.56	55 199 744
24.	0	0.3	1	1	1	512	0.04	2	2	49	102.0	110.51	56 580 143
25.	0	0.3	1	1	1	1024	0.06	3	3	52	116.0	77.5	79 357 503
26.	0	0	0	0	10	1024	0.15	8	8	55	263.0	29.3	150 004 032
27.	0	0	1	0	10	1024	0.13	7	7	53	272.3	32.51	166 455 244
28.	0	0	0	0	10	2048	0.24	12	12	49	280.6	16.39	167 876 619
29.	0	0	1	0	1	2048	0.02	1	18	51	245.5	232.55	238 134 779
30.	1	0	0	0	1	2048	0.02	1	50	54	120.9	246.37	252 282 298
31.	0	0	1	0	10	2048	0.16	8	8	49	249.9	25.84	264 557 703
32.	0	0	1	1	10	512	0.08	4	4	50	320.3	55.23	282 777 841
33.	0	0	0	1	10	1024	0.06	3	3	53	264.3	79.03	404 649 274
34.	0	0	0	1	10	2048	0.10	5	5	50	237.4	43.71	447 576 992
35.	0	0	0	0	10	512	0.00	1	1	52	492.0	237.16	607 126 560
36.	0	0	1	1	10	2048	0.13	7	7	52	250.9	31.85	652 324 553
37.	0	0	1	1	10	1024	0.03	2	2	54	328.5	122.02	1 249 510 675
38.	1	0	0	1	1	2048	0.00	0	2	53	101.5	$+\infty$	$+\infty$
39.	0	0	0	0	1	2048	0.00	0	16	54	146.2	$+\infty$	$+\infty$
40.	0	0	1	0	1	512	0.00	0	6	51	202.0	$+\infty$	$+\infty$
41.	1	0	0	1	1	1024	0.00	0	2	53	209.0	$+\infty$	$+\infty$
42.	0	0	0	0	1	1024	0.00	0	9	54	257.1	$+\infty$	$+\infty$
43.	0	0	1	0	1	1024	0.00	0	16	54	277.3	$+\infty$	$+\infty$
44.	0	0	0	0	1	512	0.00	0	4	50	369.5	$+\infty$	$+\infty$
45.	0	0	0	1	1	1024	0.00	0	0	53	$\emptyset$	$+\infty$	$+\infty$
46.	0	0	0	1	1	2048	0.00	0	0	53	$\emptyset$	$+\infty$	$+\infty$
47.	0	0	0	1	1	512	0.00	0	0	51	$\emptyset$	$+\infty$	$+\infty$
48.	0	0	0	1	10	512	0.00	0	0	51	$\emptyset$	$+\infty$	$+\infty$
49.	0	0	1	0	10	512	0.00	0	0	52	$\emptyset$	$+\infty$	$+\infty$
50.	0	0	1	1	1	1024	0.00	0	0	52	$\emptyset$	$+\infty$	$+\infty$
51.	0	0	1	1	1	2048	0.00	0	0	54	$\emptyset$	$+\infty$	$+\infty$
52.	0	0	1	1	1	512	0.00	0	0	49	$\emptyset$	$+\infty$	$+\infty$
53.	0	0.3	0	1	1	512	0.00	0	0	52	$\emptyset$	$+\infty$	$+\infty$
54.	1	0	0	0	10	1024	0.00	0	0	55	$\emptyset$	$+\infty$	$+\infty$
55.	1	0	0	0	10	2048	0.00	0	0	49	$\emptyset$	$+\infty$	$+\infty$
56.	1	0	0	0	10	512	0.00	0	0	52	$\emptyset$	$+\infty$	$+\infty$
57.	1	0	0	1	1	512	0.00	0	0	51	$\emptyset$	$+\infty$	$+\infty$
58.	1	0	0	1	10	1024	0.00	0	0	53	$\emptyset$	$+\infty$	$+\infty$
59.	1	0	0	1	10	2048	0.00	0	0	51	$\emptyset$	$+\infty$	$+\infty$
60.	1	0	0	1	10	512	0.00	0	0	51	$\emptyset$	$+\infty$	$+\infty$

Table 21.7: Results of the RBGP test series on the GCD problem.

Each of the sixty rows of this table denotes one single test series. The first column contains the rank of the series according to $pt_n(., T, h)$e following seven columns specify the settings of the test series as discussed in defined Table 21.5 on page 361. The last seven columns contain the evaluation results, which are formatted as follows:

Figure 21.17 illustrates the relation between the functional objective value f_1 of the currently best individual of the runs to the generation for the twelve best test series (according to their $p_{/r}$-values). The curves are monotone for series with constant training sets ($ct = 0$) and jagged for those where the training data changed each generation ($ct = 1$).

```
1   false  ∨ true ⇒ b_{t+1} = b_t % a_t
2 (b_t ≤ a_t) ∨ false ⇒ a_{t+1} = a_t % b_t
3   false  ∨ true ⇒ c_{t+1} = b_t
```

Listing 21.3: The RBGP version of the Euclidean algorithm.

```
1     (a_t ≤ b_t)     ∧       true       ⇒ start_{t+1} =     1 − start_t
2       false       ∨ (start_t > a_t)    ⇒ start_{t+1} = start_t * 0
3     (a_t = 1)       ∧     (0 ≥ start)  ⇒ start_{t+1} = start_t / c_t
4       true        ∧     (c_t = start_t) ⇒     c_{t+1} =     c_t + 1
5     (c_t > 0)       ∨     (a_t ≤ b_t)    ⇒     a_{t+1} =     a_t * start_t
6       true        ∧       true         ⇒     c_{t+1} =     c_t − c_t
7       false       ∨     (a_t ≠ start_t) ⇒ start_{t+1} = start_t − start_t
8       true        ∨     (c_t = start_t) ⇒     c_{t+1} =     c_t + 1
9       false       ∨      (0 < start_t)  ⇒     b_{t+1} =     b_t * c_t
10 (start_t = c_t)  ∨      (1 > start_t)  ⇒     b_{t+1} =     b_t % 1
11     (0 ≤ 1)      ∧      (0 ≥ 0)        ⇒     a_{t+1} =     a_t / c_t
12      false       ∨     (b_t < 0)      ⇒     a_{t+1} =     1 − a_t
13 (start_t ≤ start_t) ∨      true        ⇒     c_{t+1} =     c_t / 0
14    (a_t = start_t) ∧       true        ⇒     c_{t+1} =     c_t + 0
15    (a_t ≤ b_t)     ∧       true        ⇒ start_{t+1} =     1 − start_t
```

Listing 21.4: An overfitted RBGP solution to the GCP problem.

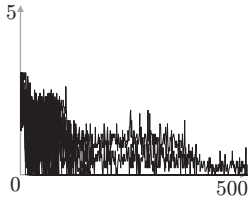


Fig. 21.17.a: *alg*=0, *cp*=1, *ss*=1, *ct*=1, *tc*=10, *ps*=2048

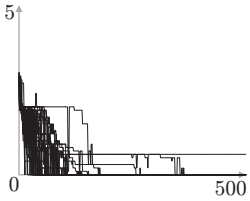


Fig. 21.17.b: *alg*=0, *cp*=1, *ss*=1, *ct*=0, *tc*=10, *ps*=2048

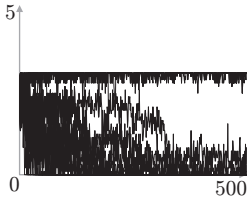


Fig. 21.17.c: *alg*=0, *cp*=1, *ss*=0, *ct*=1, *tc*=10, *ps*=2048

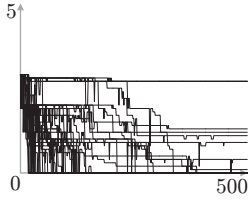


Fig. 21.17.d: *alg*=0, *cp*=1, *ss*=0, *ct*=0, *tc*=10, *ps*=2048

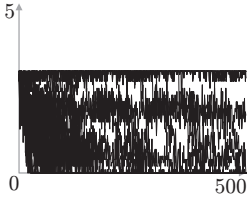


Fig. 21.17.e: *alg*=0, *cp*=1, *ss*=1, *ct*=1, *tc*=10, *ps*=1024

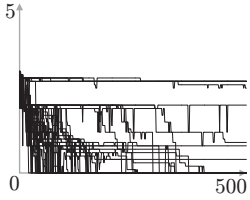


Fig. 21.17.f: *alg*=0, *cp*=1, *ss*=1, *ct*=0, *tc*=10, *ps*=1024

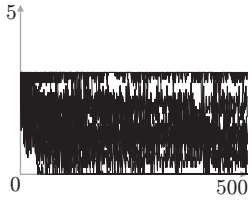


Fig. 21.17.g: *alg*=0, *cp*=1, *ss*=0, *ct*=1, *tc*=10, *ps*=1024

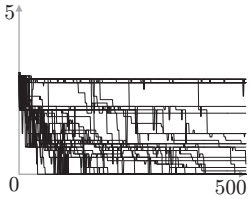


Fig. 21.17.h: *alg*=0, *cp*=1, *ss*=0, *ct*=0, *tc*=10, *ps*=1024

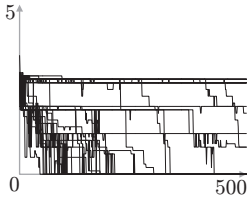


Fig. 21.17.i: *alg*=0, *cp*=1, *ss*=1, *ct*=0, *tc*=10, *ps*=512

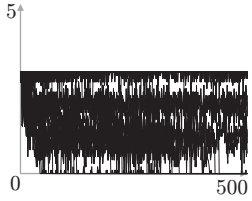


Fig. 21.17.j: *alg*=0, *cp*=1, *ss*=1, *ct*=1, *tc*=10, *ps*=512

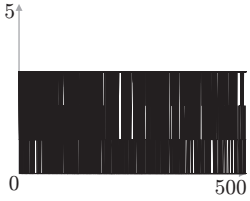


Fig. 21.17.k: *alg*=0, *cp*=1, *ss*=1, *ct*=1, *tc*=1, *ps*=2048

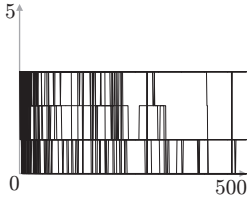


Fig. 21.17.l: *alg*=0, *cp*=1, *ss*=1, *ct*=0, *tc*=1, *ps*=1024

Figure 21.17: The f_1/generation-plots of the best configurations.

Discussion

We have sorted the runs in Table 21.7 according to their $p\tau_n$-values, i.e., the (estimate of the) number of individual evaluations that are consumed by the pt_n independent experimental runs needed to find a perfect individual with $z = 99\%$ probability. Interestingly, our first approach was to evaluate these experiments according to their $p_{/r}$-ratio, which led to a different order of the elements in the table.

Population Size The role of the population size ps is not obvious and there is no clear tendency which one to prefer when considering the $p\tau_n$-value only. Amongst the three best EAs according to this metric, we can find all three tested population sizes. When we focus on the $p_{/r}$-ratio instead, the four best runs all have a population size of 2048. At least in this experiment, the bigger the population, the bigger the chance of success of an experiment holds. The significance of this tendency is shown in Table 21.8, Table 21.9, and Table 21.10. Of course, this comes with the trade-off that more individuals need to be processed which decreases $p\tau_n$. If we perform multiple runs with smaller populations, we seemingly have a higher chance of finding at least one non-overfitted program with lesser objective function evaluations, but this trend could not be supported by the tests in the three tables.

$ps = 1024$ vs. $ps = 512$ (based on 19 samples)	
Test according to $p_{/r}$ (higher is better)	
Sign test: (see Section 28.8.1)	$\text{med}(p_{/r})\|_{ps=1024} = 0.19$, $\text{med}(p_{/r})\|_{ps=512} = 0.09$, $\alpha \approx 0.0063 \Rightarrow$ *significant* at level $\alpha = 0.05$
Randomization test: (see Section 28.8.1)	$\overline{p_{/r}}\|_{ps=1024} = 0.06$, $\overline{p_{/r}}\|_{ps=512} = 0.0$, $\alpha \approx 0.0024 \Rightarrow$ *significant* at level $\alpha = 0.05$
Signed rankt test: (see Section 28.8.1)	$R(p_{/r})\|_{ps:1024-512} = 114.0$, $\alpha \approx 0.019 \Rightarrow$ *significant* at level $\alpha = 0.05$
Test according to $p\tau_n$ (lower is better)	
Sign test: (see Section 28.8.1)	$\text{med}(p\tau_n)\|_{ps=1024} = 1.66 \cdot 10^8$, $\text{med}(p\tau_n)\|_{ps=512} = +\infty$, $\alpha \approx 0.1940 \Rightarrow$ *not significant* at level $\alpha = 0.05$
Randomization test: (see Section 28.8.1)	$\overline{p\tau_n}\|_{ps=1024} = +\infty$, $\overline{p\tau_n}\|_{ps=512} = +\infty$, *could not be applied*
Signed rankt test: (see Section 28.8.1)	$R(p\tau_n)\|_{ps:1024-512} = -94.0$, $\alpha \approx 0.0601 \Rightarrow$ *not significant* at level $\alpha = 0.05$

Table 21.8: $ps = 1024$ vs. $ps = 512$ (based on 19 samples)

$ps = 2048$ vs. $ps = 512$ (based on 20 samples)

Test according to $p_{/r}$ (higher is better)

Sign test: (see Section 28.8.1)	$\text{med}(p_{/r})\|_{ps=2048} = 0.115$, $\text{med}(p_{/r})\|_{ps=512} = 0.0$, $\alpha \approx 0.0017 \Rightarrow$ *significant* at level $\alpha = 0.05$
Randomization test: (see Section 28.8.1)	$\overline{p_{/r}}\|_{ps=2048} = 0.255$, $\overline{p_{/r}}\|_{ps=512} = 0.087$, $\alpha \approx 0.0006 \Rightarrow$ *significant* at level $\alpha = 0.05$
Signed rankt test: (see Section 28.8.1)	$R(p_{/r})\|_{ps:2048-512} = 150.0$, $\alpha \approx 0.0034 \Rightarrow$ *significant* at level $\alpha = 0.05$

Test according to $p\tau_n$ (lower is better)

Sign test: (see Section 28.8.1)	$\text{med}(p\tau_n)\|_{ps=2048} = 2.452 \cdot 10^8$, $\text{med}(p\tau_n)\|_{ps=512} = +\infty$, $\alpha \approx 0.293 \Rightarrow$ *not significant* at level $\alpha = 0.05$
Randomization test: (see Section 28.8.1)	$\overline{p\tau_n}\|_{ps=2048} = +\infty$, $\overline{p\tau_n}\|_{ps=512} = +\infty$, *could not be applied*
Signed rankt test: (see Section 28.8.1)	$R(p\tau_n)\|_{ps:2048-512} = -134.0$, $\alpha \approx 0.0104 \Rightarrow$ *significant* at level $\alpha = 0.05$

Table 21.9: $ps = 2048$ vs. $ps = 512$ (based on 20 samples)

$ps = 1024$ vs. $ps = 2048$ (based on 19 samples)

Test according to $p_{/r}$ (higher is better)

Sign test: (see Section 28.8.1)	$\text{med}(p_{/r})\|_{ps=1024} = 0.06$, $\text{med}(p_{/r})\|_{ps=2048} = 0.1$, $\alpha \approx 0.002 \Rightarrow$ *significant* at level $\alpha = 0.05$
Randomization test: (see Section 28.8.1)	$\overline{p_{/r}}\|_{ps=1024} = 0.186$, $\overline{p_{/r}}\|_{ps=2048} = 0.255$, $\alpha \approx 0.011 \Rightarrow$ *significant* at level $\alpha = 0.05$
Signed rankt test: (see Section 28.8.1)	$R(p_{/r})\|_{ps:1024-2048} = -148.0$, $\alpha \approx 0.002 \Rightarrow$ *significant* at level $\alpha = 0.05$

Test according to $p\tau_n$ (lower is better)

Sign test: (see Section 28.8.1)	$\text{med}(p\tau_n)\|_{ps=1024} = 1.66 \cdot 10^8$, $\text{med}(p\tau_n)\|_{ps=2048} = 2.523E8$, $\alpha \approx 0.1597 \Rightarrow$ *not significant* at level $\alpha = 0.05$
Randomization test: (see Section 28.8.1)	$\overline{p\tau_n}\|_{ps=1024} = +\infty$, $\overline{p\tau_n}\|_{ps=2048} = +\infty$, *could not be applied*
Signed rankt test: (see Section 28.8.1)	$R(p\tau_n)\|_{ps:1024-2048} = -22.0$, $\alpha \approx 0.6794 \Rightarrow$ *not significant* at level $\alpha = 0.05$

Table 21.10: $ps = 1024$ vs. $ps = 2048$ (based on 19 samples)

Steady State In many experimental runs, a configuration with $ss = 1$, i. e., steady-state was better than the exactly the same configuration with $ss = 0$ (compare, for instance, ranks 1 and 14 or rank 2 and 10 in Table 21.7). Also, if we look at the four best runs according to the $p_{/r}$-rate, we can see that the better two of them both have $ss = 1$ while the other two have $ss = 0$ – while all other parameters remained constant. In Table 21.11, these tendencies are reflected in the mean, median, and rank values but are not fully supported with sufficient evidence in the hypothesis tests.

$ss = 1$ vs. $ss = 0$ (based on 23 samples)	
Test according to $p_{/r}$ (higher is better)	
Sign test: (see Section 28.8.1)	$\text{med}(p_{/r})\|_{ss=1} = 0.12$, $\text{med}(p_{/r})\|_{ss=0} = 0.09$, $\alpha \approx 0.053 \Rightarrow$ *not significant* at level $\alpha = 0.05$
Randomization test: (see Section 28.8.1)	$\overline{p_{/r}}\|_{ss=1} = 0.249$, $\overline{p_{/r}}\|_{ss=0} = 0.186$, $\alpha \approx 0.0076 \Rightarrow$ *significant* at level $\alpha = 0.05$
Signed rankt test: (see Section 28.8.1)	$R(p_{/r})\|_{ss:1-0} = 125.0$, $\alpha \approx 0.057 \Rightarrow$ *not significant* at level $\alpha = 0.05$
Test according to $p\tau_n$ (lower is better)	
Sign test: (see Section 28.8.1)	$\text{med}(p\tau_n)\|_{ss=1} = 1.665 \cdot 10^8$, $\text{med}(p\tau_n)\|_{ss=0} = 1.679 \cdot 10^8$, $\alpha \approx 0.405 \Rightarrow$ *not significant* at level $\alpha = 0.05$
Randomization test: (see Section 28.8.1)	$\overline{p\tau_n}\|_{ss=1} = +\infty$, $\overline{p\tau_n}\|_{ss=0} = +\infty$, *could not be applied*
Signed rankt test: (see Section 28.8.1)	$R(p\tau_n)\|_{ss:1-0} = -27.0$, $\alpha \approx 0.692 \Rightarrow$ *not significant* at level $\alpha = 0.05$

Table 21.11: $ss = 1$ vs. $ss = 0$ (based on 23 samples)

Convergence Prevention The influence of our primitive convergence prevention mechanism is remarkable – the top 15 test series according to $p_{/r}$ all have $cp = 0.3$, and even generational tests with a population size of 512 beat steady-state runs with a population of 2048 individuals if using convergence prevention. Considering the estimated number $p\tau_n$ of individuals that need to be evaluated in pt_n independent runs to achieve 99% probability of finding a non-overfitted solution, this trend is even more obvious: all of the 23 best Genetic Programming approaches had the convergence prevention mechanism turned on. To be more precise: all but one single configuration with convergence prevention were better as all EA configurations with convergence prevention turned off. This trend is fully supported by the hypothesis tests from Table 21.12 for both, $p_{/r}$ and pt_n. It seems that keeping the evolutionary process going and not allowing a single program to spread unchanged all throughout the population increases the solution quality a lot.

$cp = 0.3$ vs. $cp = 0$ (based on 23 samples)	
Test according to $p_{/r}$ (higher is better)	
Sign test: (see Section 28.8.1)	$\text{med}(p_{/r})\|_{cp=0.3} = 0.27$, $\text{med}(p_{/r})\|_{cp=0} = 0.0$, $\alpha \approx 0 \Rightarrow$ *significant* at level $\alpha = 0.05$
Randomization test: (see Section 28.8.1)	$\overline{p_{/r}}\|_{cp=0.3} = 0.391$, $\overline{p_{/r}}\|_{cp=0} = 0.048$, $\alpha \approx 0 \Rightarrow$ *significant* at level $\alpha = 0.05$
Signed rankt test: (see Section 28.8.1)	$R(p_{/r})\|_{cp:0.3-0} = 274.0$, $\alpha \approx 0 \Rightarrow$ *significant* at level $\alpha = 0.05$
Test according to $p\tau_n$ (lower is better)	
Sign test: (see Section 28.8.1)	$\text{med}(p\tau_n)\|_{cp=0.3} = 2.96 \cdot 10^7$, $\text{med}(p\tau_n)\|_{cp=0} = +\infty$, $\alpha \approx 0 \Rightarrow$ *significant* at level $\alpha = 0.05$
Randomization test: (see Section 28.8.1)	$\overline{p\tau_n}\|_{cp=0.3} = +\infty$, $\overline{p\tau_n}\|_{cp=0} = +\infty$, *could not be applied*
Signed rankt test: (see Section 28.8.1)	$R(p\tau_n)\|_{cp:0.3-0} = -276.0$, $\alpha \approx 0 \Rightarrow$ *significant* at level $\alpha = 0.05$

Table 21.12: $cp = 0.3$ vs. $cp = 0$ (based on 23 samples)

Number of Training Cases According to the $p\tau_n$ measure, using one training case $tc = 1$ is sometimes better than using $tc = 10$. Then, we fewer individual evaluations are needed for finding a non-overfitted individual if fewer training cases are used. Obviously, using ten training cases corresponds to ten times as many individual evaluations per generation. When comparing row 1 and 7 in Table 21.7, the difference in estimated evaluations needed is only approximately two, and the configuration of row 23 needs approximately three times as many evaluations as row 10. The median in table Table 21.13 points into the other direction: because of many zero values for $p_{/r}$ with one training case, these test series perform worse. The only applicable test, the sign test, supports that ten training cases are better than one.

If we consider the fraction $p_{/r}$ of experiments that led to a perfect individual compared to the total number of experiments run for a configuration, this effect becomes even more obvious. The number of training cases has a very drastic effect: Then, the top ten test series all are based on ten training cases ($tc = 10$). Table 21.13 clearly emphasizes the significance of this tendency.

We can think of a very simple reason for that which can be observed very well when comparing for example Fig. 21.17.l with Fig. 21.17.i. In the best series based on only a single training case ($tc = 1$) and illustrated in Fig. 21.17.l, only six values (0..5) for the objective function f_1 could occur. The ninth best series depicted in Fig. 21.17.i on the other hand, had a much broader set of values of f_1 available. Since $tc = 10$ training cases were used and the final objective value assigned to an individual is the average of the scores reached in all these tests, it had much lower variations f_1 with $51 = |\{0.0, 0.1, 0.2, \ldots, 4.8, 4.9, 5.0\}|$ levels. By using multiple training sets for these runs, we have effectively reduced the ruggedness of the fitness landscape and made it easier for the evolutionary algorithm to descend a gradient. The effect of increasing the resolution of the objective functions by increasing the number of training cases has also been reported in other researchers such as Lasarczyk and Banzhaf [1258] in the area of Algorithmic Chemistries[18].

What we see is that a higher number of training cases decreases overfitting and increases the chance of a run to find good solutions. It does, however, not decrease the expected number of individuals to be processed until a good solution is found.

[18] You can find Algorithmic Chemistries discussed in Section 4.8.2 on page 205.

$tc = 1$ vs. $tc = 10$ (based on 29 samples)	
Test according to $p_{/r}$ (higher is better)	
Sign test: (see Section 28.8.1)	$\text{med}(p_{/r})\|_{tc=1} = 0.0$, $\text{med}(p_{/r})\|_{tc=10} = 0.13$, $\alpha \approx 0.0004 \Rightarrow$ *significant* at level $\alpha = 0.05$
Randomization test: (see Section 28.8.1)	$\overline{p_{/r}}\|_{tc=1} = 0.058$, $\overline{p_{/r}}\|_{tc=10} = 0.273$, *could not be applied*
Signed rankt test: (see Section 28.8.1)	$R(p_{/r})\|_{tc:1-10} = -362.0$, $\alpha \approx 0.000\,02 \Rightarrow$ *significant* at level $\alpha = 0.05$
Test according to $p\tau_n$ (lower is better)	
Sign test: (see Section 28.8.1)	$\text{med}(p\tau_n)\|_{tc=1} = +\infty$, $\text{med}(p\tau_n)\|_{tc=10} = 2.646 \cdot 10^8$, $\alpha \approx 0.0241 \Rightarrow$ *significant* at level $\alpha = 0.05$
Randomization test: (see Section 28.8.1)	$\overline{p\tau_n}\|_{tc=1} = +\infty$, $\overline{p\tau_n}\|_{tc=10} = +\infty$, *could not be applied*
Signed rankt test: (see Section 28.8.1)	$R(p\tau_n)\|_{tc:1-10} = 111.0$, *could not be applied*

Table 21.13: $tc = 1$ vs. $tc = 10$ (based on 29 samples)

Changing Training Cases In this experiment, the EAs with constant training ($ct = 0$) cases seemingly outperform those with training cases that change each generation ($ct = 1$) according to the $p\tau_n$ metric. This is strange, since one would expect that this approach would reduce overfitting and thus, since it does not a priori require more evaluations, improve the $p\tau_n$. Still, only one of tests from Table 21.14 supports the significance of this result. The average first success generation $\overline{st}$ remains roughly constant, regardless if the training data changes or not.

The best ten series according to $\overline{st}$ all use ten training cases ($tc = 10$), which seems to prevent overfitting sufficiently on its own. There is a difference in $tc = 1$, though, when we compare the perfect runs with those which were just successful. In all runs that find a solution $x \in \mathbb{X}$ with $f_1(x) = 0$, this solution is also correct if $ct = 1$, i. e., $\#p = \#s$. In the test series where $ct = 0$, usually only a fraction of the runs that found an individual with optimal functional fitness had indeed found a solution. Here, overfitting takes place and $\#p < \#s$ can be usually observed.

In the context of this experiment, the parameter ct has no substantial influence on the chance of finding a solution to the GCD problem in a run. Using training cases that change each generation even has a negatively influence on the $p\tau_n$ values. Maybe the proportion of possible programs that are truly correct compared to those that just perform good when applied to the training cases due to overfitting is relatively high in this problem. Then, the influence of this parameter could be different in other scenarios.

$ct = 0$ vs. $ct = 1$ (based on 29 samples)	
Test according to $p_{/r}$ (higher is better)	
Sign test: (see Section 28.8.1)	$\text{med}(p_{/r})\|_{ct=0} = 0.1,\ \text{med}(p_{/r})\|_{ct=1} = 0.03,$ $\alpha \approx 0.053 \Rightarrow$ *not significant* at level $\alpha = 0.05$
Randomization test: (see Section 28.8.1)	$\overline{p_{/r}}\|_{ct=0} = 0.191,\ \overline{p_{/r}}\|_{ct=1} = 0.165,$ *could not be applied*
Signed rankt test: (see Section 28.8.1)	$R(p_{/r})\|_{ct:0-1} = 86.0,$ *not significant*
Test according to $p\tau_n$ (lower is better)	
Sign test: (see Section 28.8.1)	$\text{med}(p\tau_n)\|_{ct=0} = 1.665 \cdot 10^8,\ \text{med}(p\tau_n)\|_{ct=1} = 1.25 \cdot 10^9,$ $\alpha \approx 0.458 \Rightarrow$ *not significant* at level $\alpha = 0.05$
Randomization test: (see Section 28.8.1)	$\overline{p\tau_n}\|_{ct=0} = +\infty,\ \overline{p\tau_n}\|_{ct=1} = +\infty,$ *could not be applied*
Signed rankt test: (see Section 28.8.1)	$R(p\tau_n)\|_{ct:0-1} = -313.0,$ $\alpha \approx 0.0003 \Rightarrow$ *significant* at level $\alpha = 0.05$

Table 21.14: $ct = 0$ vs. $ct = 1$ (based on 29 samples)

Comparison to Random Walks According to the chance $p_{/r}$ that a test series finds a non-overfitted solution, the best 17 configurations all were evolutionary algorithms, and apart from the 18th and 26th best series, no random walk made it into the top 30. Strangely, two random walks obtain very good placements (third and fifth rank) when considering the $p\tau_n$ metric but then, the next best random walk resides on rank 38. The two good random walks are configured in a way that leads to few evaluations, which leads to good values of $p\tau_n$ when accidentally a good solution was found. They are also the cause why only one of the tests in Table 21.15 is really significant. Nevertheless, having two random walks in such high placements could either mean that the GCD problem is very hard (so searching with an EA is not really better than with a random walk) or very simple (since randomly created programs can solve it in many cases).

Be it how it be, the dominance of Genetic Programming in most of the measurements and evaluation results of this problem indicates that there is a benefit of using EAs. One of the reasons for many of the bad performances of the random walks was that the individuals tended to become unreasonable large. This also increased the amount of time needed for evaluation. However, at least sometimes it seems to be a good idea to also try some runs which utilize the brute force of random walks when trying to solve a GP problem.

$\boldsymbol{alg = 0}$ **vs.** $\boldsymbol{alg = 1}$ (based on 12 samples)	
Test according to $\boldsymbol{p_{/r}}$ (higher is better)	
Sign test: (see Section 28.8.1)	$\text{med}(p_{/r})\|_{alg=0} = 0.0,\ \text{med}(p_{/r})\|_{alg=1} = 0.0,$ *could not be applied*
Randomization test: (see Section 28.8.1)	$\overline{p_{/r}}\|_{alg=0} = 0.046,\ \overline{p_{/r}}\|_{alg=1} = 0.024,$ $\alpha \approx 0.5 \Rightarrow$ *not significant* at level $\alpha = 0.05$
Signed rankt test: (see Section 28.8.1)	$R(p_{/r})\|_{alg:0-1} = -3.0,$ $\alpha \approx 0.925 \Rightarrow$ *not significant* at level $\alpha = 0.05$
Test according to $\boldsymbol{p\tau_n}$ (lower is better)	
Sign test: (see Section 28.8.1)	$\text{med}(p\tau_n)\|_{alg=0} = +\infty,\ \text{med}(p\tau_n)\|_{alg=1} = +\infty,$ $\alpha \approx 0.774 \Rightarrow$ *not significant* at level $\alpha = 0.05$
Randomization test: (see Section 28.8.1)	$\overline{p\tau_n}\|_{alg=0} = +\infty,\ \overline{p\tau_n}\|_{alg=1} = +\infty,$ *could not be applied*
Signed rankt test: (see Section 28.8.1)	$R(p\tau_n)\|_{alg:0-1} = -27.0,$ $\alpha \approx 0.289 \Rightarrow$ *not significant* at level $\alpha = 0.05$

Table 21.15: $alg = 0$ vs. $alg = 1$ (based on 12 samples)

22

Contests

For most of the problems that can be solved with the aid of computers, a multitude of different approaches exist. They are often comparably good and their utility in single application cases strongly depends on parameter settings and thus, the experience of the user. Contests provide a stage where students, scientists, and practitioners from the industry can demonstrate their solutions to specific problems. They not only provide indications for which techniques are suitable for which tasks, but also give incitements and trickle scientific interest to improve and extend them. The RoboCup[1], for example is known to be the origin of many new, advanced techniques in robotics, image processing, cooperative behavior, multivariate data fusion, and motion controls [114, 780, 115, 1102][2]. In this chapter, we discuss Genetic Programming approaches to competitions like the Data-Mining-Cup or the Web Service Challenge.

22.1 DATA-MINING-CUP

22.1.1 Introduction

Data Mining

Definition 22.1 (Data Mining). Data mining[3] can be defined as *the nontrivial extraction of implicit, previously unknown, and potentially useful information from data* [743] and *the science of extracting useful information from large data sets or databases* [885].

Today, gigantic amounts of data are collected in the web, in medical databases, by enterprise resource planning (ERP) and customer relationship management (CRM) systems in corporations, in web shops, by administrative and governmental bodies, and in science projects. These data sets are way too large to be incorporated directly into a decision making process or to be understood as-is by a human being. Instead, automated approaches have to be applied that extract the relevant information, to find underlying rules and patterns, or to detect time-dependent changes. Data mining subsumes the methods and techniques capable to perform this task. It is very closely related to estimation theory in stochastic (discussed in Section 28.7 on page 499) – the simplest summary of a data set is still the arithmetic mean of its elements. Data mining is also strongly related to artificial intelligence [1780, 569], which includes learning algorithms that can generalize the given information. Some of the most wide spread and most common data mining techniques are:

[1] http://www.robocup.org/ [accessed 2007-07-03] and http://en.wikipedia.org/wiki/Robocup [accessed 2007-07-03]

[2] Big up to the Carpe Noctem Robotic Soccer Team founded by my ingenious colleagues Baer and Reichle [114] (http://carpenoctem.das-lab.net/ [accessed 2008-04-23])!

[3] http://en.wikipedia.org/wiki/Data_mining [accessed 2007-07-03]

1. artificial neural networks (ANN) [207, 210],
2. support vector machines (SVM) [2107, 2150, 306, 2092],
3. logistic regression [16],
4. decision trees [186, 2243],
5. Learning Classifier Systems as introduced in Chapter 7 on page 233, and
6. naïve Bayes Classifiers [578, 1741].

The DATA-MINING-CUP

The DATA-MINING-CUP[4] (DMC) has been established in the year 2000 by the *prudsys AG*[5] and the *Technical University of Chemnitz*[6]. It aims to provide an independent platform for data mining users and data analysis tool vendors and builds a bridge between academic science and economy. Today, it is one of Europe's biggest and most influential conferences in the area of data mining.

The DATA-MINING-CUP Contest is the biggest international student data mining competition. In the spring of each year, students of national and international universities challenge to find the best solution of a data analysis problem. Figure 22.1 shows the logos of the DMC from 2005 till 2007 obtained from http://www.data-mining-cup.com/ [accessed 2007-07-03].

Fig. 22.1.a: 2005

Fig. 22.1.b: 2006

Fig. 22.1.c: 2007

Figure 22.1: Some logos of the DATA-MINING-CUP.

22.1.2 The 2007 Contest – Using Classifier Systems

In Mai 2007, the students Stefan Achler, Martin Göb, and Christian Voigtmann came into my office and told me about the DMC. They knew that evolutionary algorithms are methods for global optimization that can be applied to a wide variety of tasks and wondered if they can be utilized for the DMC too. After some discussion about the problem to be solved, we together came up with the following approach which was then realized by them. While we are going to talk about our basic ideas and the results of the experiments, a detailed view on the implementation issues using the Java Sigoa framework are discussed in Section 26.1 on page 445. We have also summarized our work for this contest in a technical report [2178].

[4] The DATA-MINING-CUP is a registered trademark of prudsys AG. Der DATA-MINING-CUP ist eine eingetragene Marke der prudsys AG. http://www.data-mining-cup.com/ [accessed 2007-07-03], http://www.data-mining-cup.de/ [accessed 2007-07-03]

[5] http://www.prudsys.de/ [accessed 2007-07-03]

[6] http://www.tu-chemnitz.de [accessed 2007-07-03] (Germany) – By the way, that's the university I've studied at, a great place with an excellent computer science department.

A Structured Approach to Data Mining

Whenever any sort of problem should be solved, a structured approach is always advisable. This goes for the application of optimization methods like evolutionary algorithms as well as for deriving classifiers in a data mining problem. In this section we discuss a few simple steps which should be valid for both kinds of tasks and which have been followed in our approach to the 2007 DMC.

The first step is always to clearly specify the problem that should be solved. Parts of this specification are possible target values and optimization criteria as well as the semantics of the problem domain. The optimization criteria tell us how different possible solutions can be compared with each other. If we were to sell tomatoes, for example, the target value (subject to maximization) would be the profit. Then again, the semantics of the problem domain allow us to draw conclusions on what features are important in the optimization or data mining process. When selling tomatoes, for instance, the average weight of the vegetables, their color, and maybe the time of the day when we open the store are important. The names of our customers on the other hand are probably not. The task of the DMC 2007 Contest, outlined in Section 22.1.2, is a good example for such a problem definition.

Before choosing or applying any data mining or optimization technique, an initial analysis of the given data should be performed. With this review and the problem specification, we can filter the data and maybe remove unnecessary features. Additionally, we will gain insight in the data structure and hopefully can already eliminate some possible solution approaches. It is, of course, better to exclude some mining techniques that cannot lead to good results in the initial phase instead of wasting working hours in trying them out to avail. Finding solutions with offline evolutionary computation usually takes a while, so we have now to decide on one or two solution approaches that are especially promising for the problem defined. We have performed this step for the DMC 2007 Contest data in Section 22.1.2 on page 377.

After this, we can apply the selected approaches. Of course, running an optimizer on all known sample data at once is not wise. Although we will obtain a result with which we can solve the specified problem for all the known data samples, it is possible not a good solution. Instead, it may be overfitted and can *only* process the data we were given. Normally however, we will only be provided with fraction of the "real data" and want to find a system that is able to perform well also on samples that are not yet known to us. Hence, we need to find out whether or not our approach generalizes. Therefore, solutions are derived for a subset of the available data samples only, the training data. These solutions are then tested on the test set, the remaining samples not used in its creations.[7] The system we have created generalizes well if it is rated approximately equally good by the optimization criterion for both, the training and the test data. Now we can repeat the process by using all available data. We have evolved classifier systems that solve the DMC 2007 Contest according to this method in Section 22.1.2 on page 379.

The students Achler, Göb, and Voigtmann have participated in the 2007 DMC Contest and proceeded according to this pattern. In order to solve the challenge, they chose for a genetic algorithm evolving a fuzzy classifier system. The results of their participation are discussed in Section 22.1.2 on page 382. The following sub-sections are based on their experiences and impressions, and reproduce how they proceeded.

The Problem Definition

Rebate systems are an important means to animate customers to return to a store in classical retail. In the 2007 contest, we consider a check-out couponing system. Whenever a customer leaves a store, at the end of her bill a coupon can be attached. She then can use the coupon to receive some rebate on her next purchase. When printing the bill at the checkout, there are three options for couponing:

[7] See also Section 1.4.8 for this approach.

Case N: attach no coupon to the bill,
Case A: attach coupon type **A**, a general rebate coupon, to the bill, or
Case B: attach coupon type **B**, a special voucher, to the bill.

The profit of the couponing system is defined as follows:

1. Each coupon which is not redeemed costs 1 money unit.
2. For each redeemed coupon of type **A**, the retailer gains 3 money units.
3. For each coupon of type **B** which is redeemed, the retailer gains 6 money units.

It is thus clear that simply printing both coupons at the end of each bill makes no sense. In order to find a good strategy for coupon printing, the retailer has initiated a survey. She wants to find out which type of customer has an affinity to cash in coupons and, if so, which type of coupon most likely. Therefore the behavior of 50000 customers has been anonymously recorded. For all these customers, we know the customer ID, the number of redemptions of 20 different coupons and the historic information whether coupon type **A**, coupon type **B**, or none of them has been redeemed. Cases where both have been cashed in are omitted.

ID	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	Coupon
97006	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	**N**
97025	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**N**
97032	1	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	**A**
97051	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	**N**
97054	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	**N**
97061	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	1	0	0	0	**A**
97068	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	**N**
97082	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	**N**
97093	0	0	0	1	0	0	1	0	0	0	0	1	0	0	1	0	1	0	0	0	**B**
97113	0	0	1	1	0	0	1	0	0	1	0	0	0	0	1	0	0	0	0	0	**A**
97128	1	1	0	0	0	0	1	0	0	1	0	0	0	0	1	0	0	0	0	0	**N**
97143	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	**N**
97178	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	**N**
97191	0	0	1	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	**N**
97204	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**N**
97207	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	**N**
94101	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	**N**
94116	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	**N**
94118	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	**N**
94126	1	0	0	1	0	0	1	0	1	1	0	1	0	0	1	0	0	0	0	0	**A**
94129	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	**N**
94140	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	**N**
94143	0	0	0	1	0	0	1	0	0	1	0	0	0	0	1	0	0	0	0	0	**N**
94148	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	**N**
94151	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	**N**
83159	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	**N**
83162	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	**N**
83172	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	**N**
83185	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	**N**
83197	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	**N**
83203	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	**N**
83224	0	0	1	0	0	0	0	1	0	0	1	0	0	0	1	0	1	0	0	0	**N**
83229	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1	0	1	0	0	0	**N**
83233	0	0	1	1	0	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	**N**
83235	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	**N**
83245	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	**N**
83259	0	0	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	**N**
83264	0	0	1	0	0	0	0	1	0	0	1	0	0	0	1	0	0	0	0	0	**N**
83268	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	**N**
83276	0	1	1	1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	**N**
83281	0	0	1	1	0	0	1	0	0	1	1	1	0	0	1	0	0	0	0	0	**N**
83285	0	0	1	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	**N**
83298	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	**N**
83315	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0	**N**
83337	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**A**
83347	0	0	0	1	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	**N**

Figure 22.2: A few samples from the DMC 2007 training data.

Figure 22.2 shows some samples from this data set. The task is to use it as training data in order to derive a classifier C that is able to decide from a record of the 20 features whether

a coupon **A**, **B**, or none should be provided to a customer. This means to maximize the profit $P(C)$ of retailer gained by using the classifier C which can be computed according to

$$P(C) = 3 * AA + 6 * BB - 1 * (NA + NB + BA + AB) \tag{22.1}$$

where

1. AA is the number of correct assignments for coupon **A**.
2. BB is the number of correct assignments for coupon **B**.
3. NA is the number of wrong assignments to class **A** from the real class **N**.
4. NB is the number of wrong assignments to class **B** from the real class **N**.
5. BA is the number of wrong assignments to class **A** from the real class **B**.
6. AB is the number of wrong assignments to class **B** from the real class **A**.

Wrong assignments from the classes **A** and **B** to **N** play no role.

The classifier built with the 50000 data samples is then to be applied to another 50000 data samples. There however, the column *Coupon* is missing and should be the result of the classification process. Based on the computed assignments, the profit score P is calculated for each contestant by the jury and the team with the highest profit will win.

Initial Data Analysis

The test dataset has some properties which make it especially hard for learning algorithms to find good solutions. Figure 22.3 for example shows three data samples with exactly the same features but different classes. In general, there is some degree of fuzzyness and noise, and clusters belonging to different classes overlap and contain each other. Since the classes

ID	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C16	C17	C18	C19	C20	Coupon
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
97054	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	**N**
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
94698	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	**A**
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
96366	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	**B**

Figure 22.3: DMC 2007 sample data – same features but different classes.

cannot be separated by hyper-planes in a straightforward manner, the application of neural networks and support vector machines becomes difficult. Furthermore, the values of the features take on only four different values and are zero to 83.7%, as illustrated in Table 22.1. In general, such a small number of possible feature values makes it hard to apply methods

value	number of occurrences
0	837 119
1	161 936
2	924
3	21

Table 22.1: Feature-values in the 2007 DMC training sets.

that are based on distances or averages. Stefan, Martin, and Christian had already come to this conclusion when we met. At least, one positive fact can easily be found by eyesight when

inspecting the training data: the columns *C6*, *C14*, and *C20*, marked gray in Figure 22.2, are most probably insignificant since they are almost always zero and, hence, can be excluded from further analysis. The same goes for the first column, the customer *ID*, by common sense.

The Solution Approach: Classifier Systems

From the initial data analysis, we can reduce the space of values a feature may take on to `0`, `1`, and `>1`. This limited, discrete range is especially suited for Learning Classifier Systems (LCS) discussed in Chapter 7 on page 233.

Since we already know the target function, $P(C)$, we do not need the learning part of the LCS. Instead, our idea was to use the profit $P(C)$ defined in Equation 22.1 directly as objective function for a genetic algorithm.

Very much like in the Pitt-approach [1926, 516, 1912] in LCS, the genetic algorithm would be based on a population of classifier systems. Such a classifier system is a list of rules (the single classifiers). A rule contains a classification part and one condition for each feature in the input data. We used a two bit alphabet for the conditions, allowing us to encode the four different conditions per feature listed in Table 22.2. The three different classes can be

condition (in genotype)	condition (in phenotype)	corresponding feature value
`00`	0	must be 0
`01`	1	must be ≥ 1
`10`	2	must be > 1
`11`	3	do not care (i. e., any value is ok)

Table 22.2: Feature conditions in the rules.

represented using two additional bits, where `00` and `11` stands for **A**, `01` means **B**, and `10` corresponds to **N**. We leave three insignificant features away, so a rule is in total $17*2+2 = 36$ bits small. This means that we need less memory for a classifier system with 17 rules than for 10 double precision floating point numbers, as used by a neural network, for example.

When a feature is to be classified, the rules of a classifier system are applied step by step. A rule fits to a given data sample if none of its conditions are violated by a corresponding sample feature. As soon as such a rule is found, the input is assigned to the class identified by the classification part of the rule. This stepwise interpretation creates a default hierarchy that allows classifications to include each other: a more specific rule (which is checked before the more general one) can represent a subset of features which is subsumed by a rule which is evaluated later. If no rule in the classifier systems fits to a data sample, **N** is returned per default since misclassifying an **A** or **B** as an **N** at least does not introduce a penalty in $P(C)$ according to Equation 22.1.

Since the input data is noisy, it turned out to be a good idea to introduce some fuzzyness in our classifiers, too, by modifying this default rule. During the classification process, we remember the rule which was violated by the least features. In the case that no rule fits perfectly, we check if the number of these misfits is less than one fifth of the features, in this case $\frac{17}{5} \approx 3$. If so, we consider it as a match and classify the input according to the rules classification part. Otherwise, the original default rule is applied and **N** is returned. Figure 22.4 outlines the relation of the genotype and phenotype of such a fuzzy classifier system. It shows a classifier system consisting of four rules that has been a real result of the genetic algorithm. In this graphic, we also apply it to the second sample of the dataset that is to be classified. As one can easily see, none of the four rules matches fully – which strangely is almost always the case for classifier systems that sprung of the artificial evolution

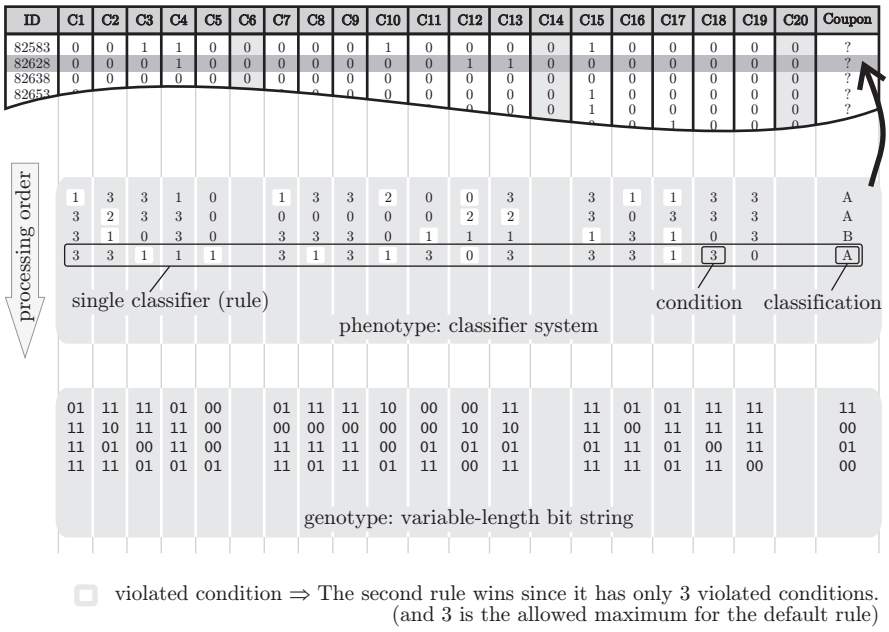


Figure 22.4: An example classifier for the 2007 DMC.

performed by us. The data sample, however, violates only three conditions of the second rule and, hence, stays exactly at the $\frac{1}{5}$-threshold. Since no other rule in the classifier system has less misfit conditions, the result of this classification process will be **A**.

Analysis of the Evolutionary Process

Table 22.3 lists the settings of the evolutionary algorithm that we have applied evolve classifiers for the Data-Mining-Cup 2007 problem.

Parameter	Short	Description
Problem Space	$\mathbb{X}$	The space of classifiers consisting of between 2 and 55 rules. (see Section 22.1.2)
Objective Functions	$\boldsymbol{F}$	$F = \{f_1, f_1\}$, where $f_1(C) = -P(C)$ rates the profic and $f_2(C) = \max\{\text{len}(C), 3\}$ is the non-functional length criterion.
Search Space	$\mathbb{G}$	The variable-length bit strings with a length between 74 and 2035 bits and a gene size of 37 bits. (see Section 3.5)
Search Operations	$\boldsymbol{Op}$	$cr = 70\%$ multi-point crossover, $mr = 30\%$ mutation (including single-bit flips, insertion, and deletion of genes)
GPM	**gpm**	(see Figure 22.4)
Optimization Algorithm	$\boldsymbol{alg}$	elitist evolutionary algorithm (see Algorithm 2.2)
Comparison Operator	$\boldsymbol{cm}$	Pareto comparison (see Section 1.2.2)
Population Size	$\boldsymbol{ps}$	$ps = 10\,243$
Maximum Archive Size	$\boldsymbol{as}$	The size of the archive with the best known individuals was limited to $as = 101$. (see Definition 2.4)

Steady-State	***ss***	The algorithm was generational (not steady-state) ($ss = 0$). (see Section 2.1.6)
Fitness Assignment Algorithm	***fa***	For fitness assignment in the evolutionary algorithm, Pareto ranking was used. (see Section 2.3.3)
Selection Algorithm	***sel***	A binary ($k = 2$) tournament selection was applied. (see Section 2.4.4)
Convergence Prevention	***cp***	No additional means for convergence prevention were used, i. e., $cp = 0$. (see Section 2.4.8)
Generation Limit	***mxt***	The maximum number of generations that each run is allowed to perform. (see Definition 1.43) $mxt = 1001$

Table 22.3: The settings of the experiments for the Data-Mining-Cup.

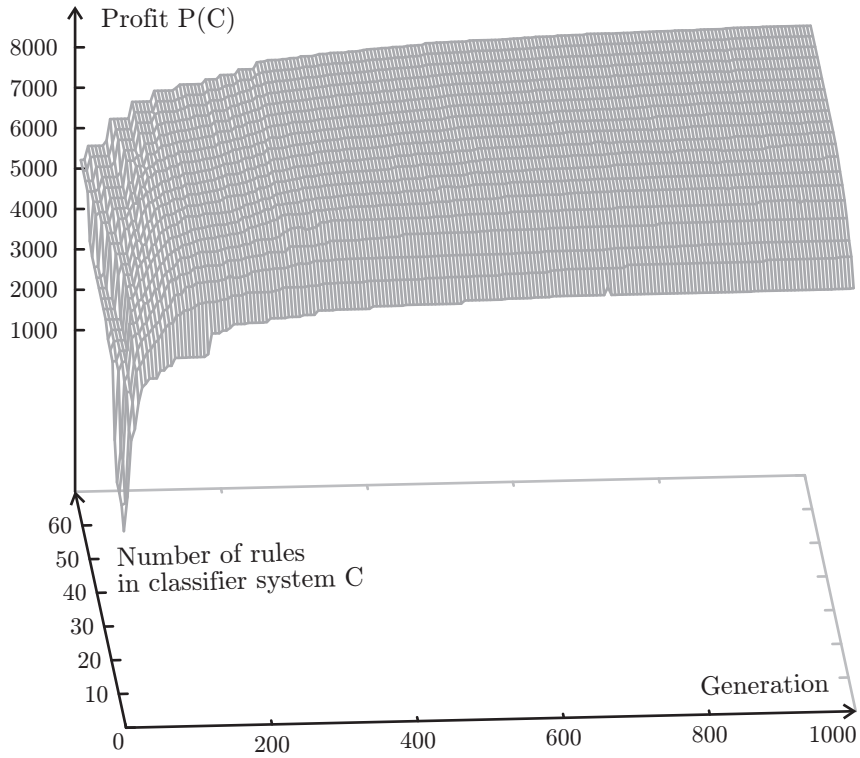


Figure 22.5: The course of the classifier system evolution.

Figure 22.5 illustrates the course of the classifier system evolution. We can see a logarithmic growth of the profit with the generations as well as with the number of rules in the classifier systems. A profit of 8800 for the 50 000 data samples has been reached. Experiments with 10 000 datasets held back and an evolution on the remaining 40 000 samples indicated that the evolved rule sets generalize sufficiently well. The cause for the generalization of the results is the second, non-functional objective function which puts pressure into the direction of smaller classifier systems and the modified default rule which allows noisy input data. The result of the multi-objective optimization process is the Pareto-optimal set. It comprises all solution candidates for which no other individual exists that is better in at least one objective value and not worse in any other. Figure 22.6 displays some classifier systems which are members of this set after generation 1000. $C1$ is the smallest non-dominated classifier

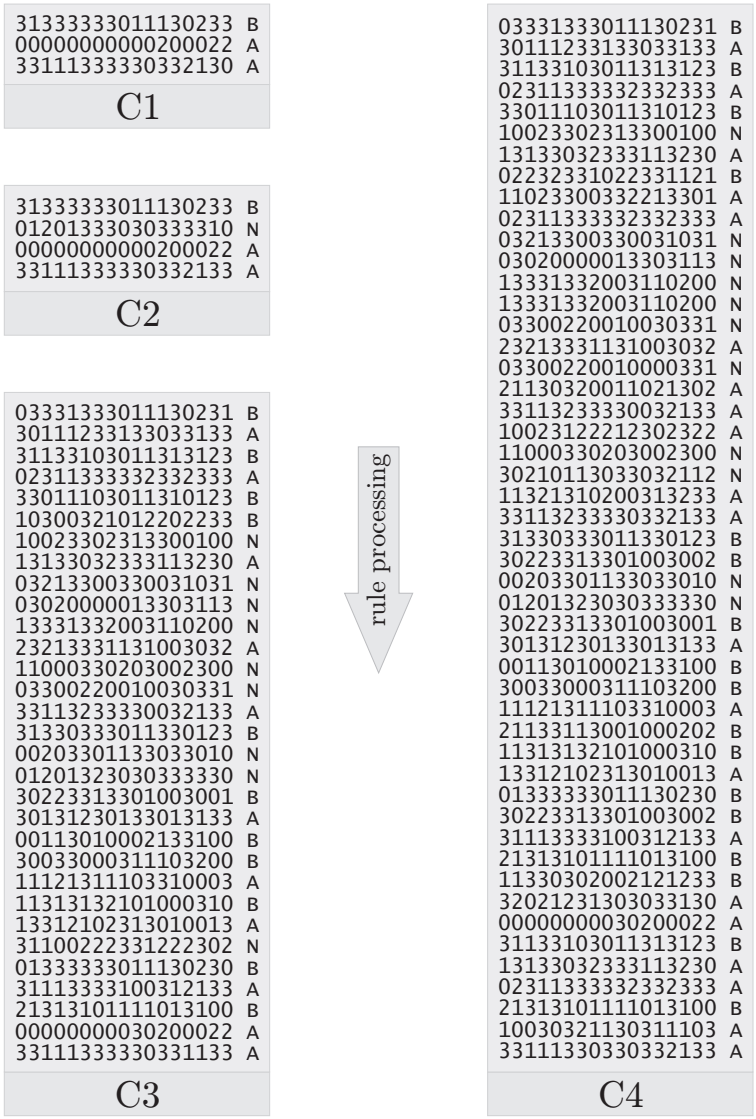


Figure 22.6: Some Pareto-optimal individuals among the evolved classifier systems.

system. It consists of three rules which lead to a profit of 7222. $C2$, with one additional rule, reaches 7403. The 31-rule classifier system $C3$ provides a gain of 8748 to which the system with the highest profit evolved, $C4$, adds only 45 to a total of 8793 with a trade-off of 18 additional rules (49 in total).

As shown in Table 22.1 on page 377, most feature values are `0` or `1` and there are only very few `2` and `3`-valued features. In order to find out how different treatment of those will influence the performance of the classifiers and of the evolutionary process, we slightly modified the condition semantics in Table 22.4 by changing the meaning of rule `2` from > 1 to ≤ 1 (compare with Table 22.2 on page 378).

The progress of the evolution depicted in Figure 22.7 exhibits no significant difference to the first one illustrated in Figure 22.5. With the modified rule semantics, the best classifier system evolved delivered a profit of 8666 by utilizing 37 rules. This result is also not very much different from the original version. Hence, the treatment of the features with the values 2 and 3 does not seem to have much influence on the overall result. In the first approach,

condition (in genotype)	condition (in phenotype)	corresponding feature value
00	0	must be 0
01	1	must be ≥ 1
10	2	must be ≤ 1
11	3	do not care (i. e., any value is ok)

Table 22.4: Different feature conditions in the rules.

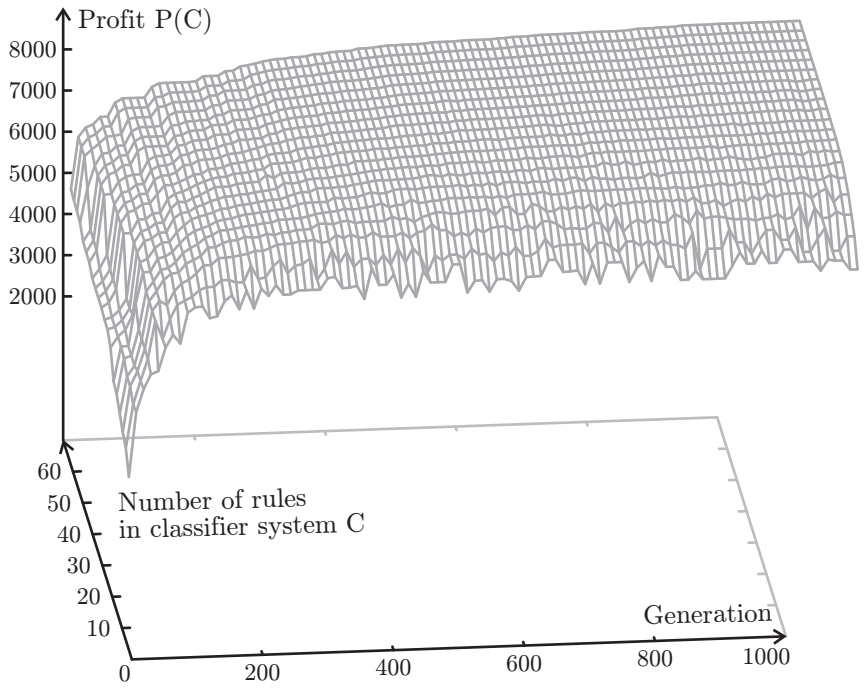


Figure 22.7: The course of the modified classifier system evolution.

rule-condition 2 used them as distinctive criterion. The new method treats them the same as feature value 1, with slightly worse results.

Contest Results and Placement

A record number of 688 teams from 159 universities in 40 countries registered for the 2007 DMC Contest, from which only 248 were finally able to hand in results. The team of the RWTH Aachen won place one and two by scoring 7890 and 7832 points on the contest data set. Together with the team from the Darmstadt University of Technology, ranked third, they occupy the first eight placements. Our team reached place 29 which is quite a good result considering that none of its members had any prior experience in data mining.

Retrospectively, one can recognize that the winning gains are much lower than those we have discussed in the previous experiments. They are, however, results of the classification of a different data set – the profits in our experiment are obtained from the training sets and not from the contest data. Although our classifiers did generalize well in the initial tests, they seem to suffer from some degree of overfitting. Furthermore, the systems discussed here are the result of reproduced experiments and not the original contribution from the students. The system with the highest profit that the students handed in also had gains around 8600 on the training sets. With a hill climbing optimizer, we squeezed out another

200, increasing, of course, the risk of additional overfitting. In the challenge, the best scored score of our team, a total profit of 7453 (5.5% less than the winning team). This classifier system was however grown with a much smaller population (4096) than in the experiments here, due to time restrictions.

It should also be noted that we did not achieve the best result with the best single classifier system evolved, but with a primitive combination of this system with another one: If both classifier systems delivered the same result for a record, this result was used. Otherwise, `N` was returned, which at least would not lead to additional costs (as follows from Equation 22.1 on page 377).

Conclusion

In order to solve the 2007 Data-Mining-Cup contest we exercised a structured approach. After reviewing the data samples provided for the challenge, we have adapted the idea of classifier systems to the special needs of the competition. As a straightforward way of obtaining such systems, we have chosen a genetic algorithm with two objective functions. The first one maximized the utility of the classifiers by maximizing the profit function provided by the contest rules. The section objective function minimized a non-functional criterion, the number of rules in the classifiers. It was intended to restrict the amount of overfitting. The bred classifier systems showed reasonable good generalization properties on the test data sets separated from the original data samples, but seem to be overfitted when comparing these results with the profits gained in the contest. A conclusion is that it is hard to prevent overfitting in an evolution based on limited sample data – the best classifier system obtained will possibly be overfitted. In the challenge, the combination of two classifiers yielded the best results. Such combinations of multiple, independent systems will probably perform better than each of them alone.

In further projects, especially the last two conclusions drawn should be considered. Although we used a very simple way to combine our classifier systems for the contest, it still provided an advantage.

A classifier system in principle is nothing more but an estimator[8]. There exist many sophisticated methods of combining different estimators in order to achieve better results [88]. The original version of such "boosting algorithms", developed by Schapire [1825], theoretically allows to achieve an arbitrarily low error rate, requiring basic estimators with a performance only slightly better than random guessing on any input distribution. The AdaBoost algorithm by Freund and Schapire [746, 747] additionally takes into consideration the error rates of the estimators. With this approach, even classifiers of different architectures like a neural network and a Learning Classifier System can be combined. Since the classification task in the challenge required non-fuzzy answers in form of definite set memberships, the usage of weighted majority voting [745, 1826], as already applied in a very primitive manner, would probably have been the best approach.

22.2 The Web Service Challenge

22.2.1 Introduction

Web Service Composition

The necessity for fast service composition systems and the overall idea of the WS-Challenge is directly connected with the emergence of Service-Oriented Architectures (SOA).

Today, companies rely on IT-architectures which are as flexible as their business strategy. The software of an enterprise must be able to adapt to changes in the business processes

[8] See our discussion on estimation theory in Section 28.7 on page 499.

like accounting, billing, the workflows, and even in the office software. If external vendors, suppliers, or customers change, the interfaces to their IT systems must be newly created or modified too. Hence, the architecture of corporate software has to be built with the anticipation of changes and updates [796, 1026, 2199].

A SOA is the ideal architecture for such systems [1362, 635]. Service oriented architectures allow us to modularize the business logic and to implement it in the form of services accessible in a network. Services are building blocks for service processes which represent the workflows of an enterprise. They can be added, removed, and updated at runtime without interfering with the ongoing business. A SOA can be seen as a complex system with manifold services as well as *n:m* dependencies between services and applications:

1. An application may need various service functionalities.
2. Different applications may need the same service functionality.
3. A certain functionality may be provided by multiple services.

Business now depends on the availability of service functionality, which is ensured by service management. Manual service management however becomes more and more cumbersome and ineffective with a rising number of relations between services and applications. Here, self-organization promises a solution for finding services that offer a specific functionality automatically.

Self-organizing approaches need a combination of syntactic and semantic service descriptions in order to decide whether a service provides a wanted functionality or not. Common syntactic definitions like WSDL [249] specify the order and types of service parameters and return values. Semantic interface description languages like OWL-S [71] or WSMO [1748, 1749] annotate these parameters with a meaning. While WSDL can be used to define a parameter `myisbn` of the type `String`, with OWL-S we can define that `myisbn` expects a `String` which actually contains an `ISBN`. Via a taxonomy we can now deduce that values which are annotated as either `ISBN-10` or `ISBN-13`[9] can be passed to this service.

A wanted functionality is defined by a set of required output and available input parameters. A service offers this functionality if it can be executed with these available input parameters and its return values contain the needed output values. In order to find such services, the semantic concepts of their parameters are matched rather than their syntactic data types.

Many service management approaches employ semantic service discovery [223, 224, 222, 1314, 1748, 1749, 830, 831]. Still, there is a substantial lack of research on algorithms and system design for fast response service discovery. This is especially the case in service composition where service functionality is not necessarily provided by a single service. Instead, combinations of services (*compositions*) are discovered. The sequential execution of these services provides the requested functionality.

The Web Service Challenge

Since 2005, the annual Web Service Challenge[10] (WS-Challenge, WSC) provides a platform for researchers in the area of web service composition to compare their systems and exchange experiences [212, 213, 214]. It is co-located with the IEEE Conference on Electronic Commerce (CEC) and the IEEE International Conference on e-Technology, e-Commerce, and e-Service (EEE).

Each team participating in this challenge provides one software system. A jury then uses these systems to solve different, complicated web service discovery and composition tasks. The major evaluation criterion for the composers is the speed with which the problems are solved. Another criterion is the completeness of the solution. Additionally, there is also a prize for the best overall system architecture.

[9] There are two formats for International Standard Book Numbers (ISBNs), ISBN-10 and ISBN-13, see also `http://en.wikipedia.org/wiki/Isbn` [accessed 2007-09-02].

[10] see `http://www.ws-challenge.org/` [accessed 2007-09-02]

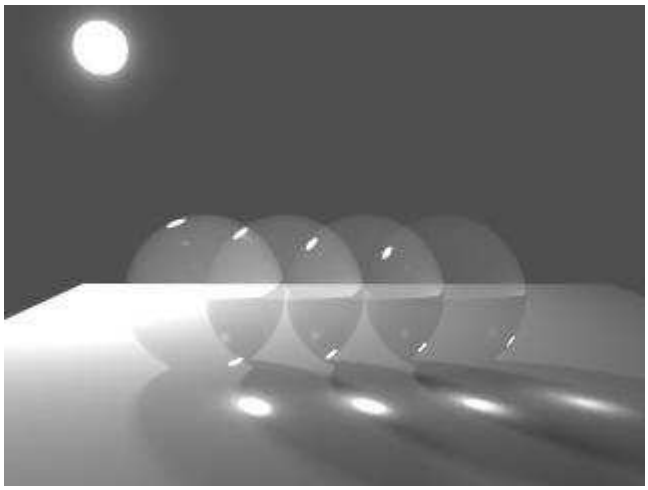


Figure 22.8: The logo of the Web Service Challenge.

22.2.2 The 2006/2007 Semantic Challenge

We have participated in the 2006 and 2007 Web Service Challenges [225, 226]. Here we present the system, algorithms and data structures for semantic web service composition that we applied in both challenges. A slightly more thorough discussion of this topic can be found in Weise et al. [2179, 2184]. The tasks of the 2006 Web Service Challenge in San Francisco, USA and the 2007 WSC in Tokyo, Japan are quite similar and only deviate in the way in which the solutions have to be provided by the software systems. Hence, we will discuss the two challenges together in this single section. Furthermore, we only consider the semantic challenges, since they are more demanding than mere syntactic matching.

Semantic Service Composition

In order to discuss the idea of semantic service composition properly, we need some prerequisites. Therefore, let us initially define the set of all semantic concepts $\mathbb{M}$. All concepts that exist in the knowledge base are members of $\mathbb{M}$ and can be represented as nodes in a wood of taxonomy trees.

Definition 22.2 (subsumes**).** Two concepts $A, B \in \mathbb{M}$ can be related in one of four possible ways. We define the predicate subsumes : $\mathbb{M} \times \mathbb{M} \mapsto \mathbb{B}$ to express this relation as follows:

1. subsumes(A, B) holds if and only if A is a generalization of B (B is then a specialization of A).
2. subsumes(B, A) holds if and only if A is a specialization of B (B is then a generalization of A).
3. If neither subsumes(A, B) nor subsumes(B, A) holds, A and B are not related to each other.
4. subsumes(A, B) and subsumes(B, A) is `true` if and only if $A = B$ (antisymmetrie, as defined in Equation 27.59 on page 463).

The subsumes relation is transitive (see Equation 27.55 on page 462), and so are generalization and specialization: If A is a generalization of B (subsumes(A, B)) and B is a generalization of C (subsumes(B, C)), then A is also a generalization of C (subsumes(A, C)). The same goes vice versa for specialization, here we can define that if A is a specialization of B (subsumes(B, A)) and A is also a specialization of C (subsumes(C, A)), then either

subsumes(B, C) or subsumes(C, B) (or both) must hold,i. e., either C is a specialization of B, or B is a specialization of C, or $B = C$.

If a parameter x of a service is annotated with A and a value y annotated with B is available, we can set $x = y$ and call the service only if subsumes(A, B) holds (*contravariance*). This means that x expects less or equal information than given in y. The hierarchy defined here is pretty much the same as in object-oriented programming languages. If we imagine `A` and `B` to be classes in Java, subsumes(A, B) can be considered to be equivalent to the expression `A.class.isAssignableFrom(B.class)`. If it evaluates to `true`, a value `y` of type `B` can be assigned to a variable `x` of type `A` since `y instanceof A` will also be `true`.

From the viewpoint of a composition algorithm, there is no need for a distinction between parameters and the annotated concepts. The set $\mathbb{S}$ contains all the services s known to the service registry. Each service $s \in \mathbb{S}$ has a set of required input concepts $s.in \subseteq \mathbb{M}$ and a set of output concepts $s.out \subseteq \mathbb{M}$ which it will deliver on return. We can trigger a service if we can provide all of its input parameters.

Similarly, a composition request R always consists of a set of available input concepts $R.in \subseteq \mathbb{M}$ and a set of requested output concepts $R.out \subseteq \mathbb{M}$. A composition algorithm in the sense of the Web Service Challenges 2006 and 2007 discovers a (topologically sorted[11]) set of n services $S = \{s_1, s_2, \ldots, s_n\} : s_1, \ldots, s_n \in \mathbb{S}$. As shown in Equation 22.2, the first service (s_0) of a valid composition can be executed with instances of the input concepts $R.in$. Together with $R.in$, its outputs $(s1.out)$ are available for executing the next service (s_2) in S, and so on. The composition provides outputs that are either annotated with exactly the requested concepts $R.out$ or with more specific ones (*covariance*). Assuming that $R.in \cap R.out = \emptyset$, for each composition solving the request R, the predicate isGoal(S) will hold. With Equation 22.2, we have defined the goal predicate which we can use in any form of informed or uninformed state space search (see Chapter 17 on page 289).

$$\begin{aligned}\text{isGoal}(S) \Leftrightarrow\ & \forall A \in s_1.in\ \exists B \in R.in : \text{subsumes}(A, B) \wedge \\ & \forall A \in s_i.in, i \in \{2..n\}\ \exists B \in R.in \cup s_{i-1}.out \cup .. \cup s_1.out : \text{subsumes}(A, B) \wedge \\ & \forall A \in R.out\ \exists B \in s_1.out \cup .. \cup s_n.out : \text{subsumes}(A, B)\end{aligned} \tag{22.2}$$

The Problem Definition

In the 2006 and 2007 Web Service Challenge, the composition software is provided with three parameters:

1. A concept taxonomy to be loaded into the knowledge base of the system. This taxonomy was stored in a file of the XML Schema format [641].
2. A directory containing the specifications of the service to be loaded into the service registry. For each service, there was a single file given in WSDL format [249].
3. A query file containing multiple service composition requests $R_1, R_2, \ldots$ in a made-up XML [284] format.

These formats are very common and allow the contestants to apply the solutions in real world applications later as well as using customized versions of their already existing applications. The expected result to be returned by the software was also a stream of data in a proprietary XML dialect containing all possible service compositions that solved the queries according to Equation 22.2. It was possible that a request R_i was resolved by multiple service compositions. In the 2006 challenge, the communication between the jury and the programs was via command line or other interfaces provided by the software, in 2007 a web service interface was obligatory.

[11] The set S is only partially ordered since, in principle, some services may be executed in parallel if they do not depend on each other.

We will not discuss the data formats used in this challenge any further since they are replaceable and do not contribute to the way the composition queries are solved. Remarkably, however, were the following restrictions in the challenge tasks:

1. There exists at least one solution for each query.
2. The services in the solutions are represented as a sequence of sets. Each set contains equivalent services. Executing one service from each set forms a valid composition S. This representation does not allow for any notation of parallelization.

Before we elaborate on the solution itself, let us define the operation "promising" which obtains the set of all services $s \in \mathbb{S}$ that produce an output parameter annotated with the concept A (regardless of their inputs).

$$\forall A \in \mathbb{S}, \forall s \in \text{promising}(A) \Rightarrow \exists B \in s.out : \text{subsumes}(A, B) \tag{22.3}$$

The composition system that we have applied in the 2007 WSC consists of three types of composition algorithms. The problem space $\mathbb{X}$ that they investigate is basically the set of all possible permutations of all possible sets of services. The power set $\mathcal{P}(\mathbb{S})$ includes all possible subsets of $\mathbb{S}$. $\mathbb{X}$is the set of all possible permutations of the elements in such subsets, in other words $\mathbb{X} \subseteq \{\forall \text{ permutation}(\xi) : \xi \in \mathcal{P}(\mathbb{S})\}$.

An (Uninformed) Algorithm Based on IDDFS

The most general and straightforward approach to web service composition is the uninformed search, an iterative deepening depth-first search (IDDFS) algorithm as discussed in Section 17.3.4 on page 294. Uninformed search algorithms do not make use of any information different from goal predicates as defined in Equation 22.2. We can build such a composition algorithm based on iterative deepening depth-first search. It is only fast in finding solutions for small service repositories but optimal if the problem requires an exhaustive search. Thus, it may be used by the strategic planner in conjunction with another algorithm that runs in parallel if the size of the repository is reasonable small or if it is unclear whether the problem can actually be solved.

Algorithm 22.1 (webServiceCompositionIDDFS) builds a valid web service composition starting from the back. In each recursion, its internal helper method dl_dfs_wsc tests all elements A of the set *wanted* of yet unknown parameters. It then iterates over the set of all services s that can provide A. For every single s, *wanted* is recomputed. If it becomes the empty set $\emptyset$, we have found a valid composition and can return it. If dl_dfs_wsc is not able to find a solution within the maximum depth limit (which denotes the maximum number of services in the composition), it returns $\emptyset$. The loop in Algorithm 22.1 iteratively invokes dl_dfs_wsc by increasing the depth limit step by step, until a valid solution is found.

An (Informed) Heuristic Approach

The IDDFS-algorithm just discussed performs an uninformed search in the space of possible service compositions. As we know from Section 17.4 on page 295, we can increase the search speed by defining good heuristics and using domain information. Such information can easily be derived in this research area. Therefore, we will again need some further definitions. Notice that the set functions specified in the following does not need to be evaluated every time they are queried, since we can maintain their information as meta-data along with the composition and thus save runtime.

Let us first define the set of unsatisfied parameters wanted$(S) \subseteq \mathbb{M}$ in a candidate composition S as

$$\begin{aligned}\text{wanted}(S) \Leftrightarrow\ & \{A : s_i \in S \wedge A \in s_i.in \wedge A \notin R.in \wedge (\nexists s_j \in S : 0 \leq j < i \wedge A \in s_j.out)\} \cup \\ & R.out \setminus \left(R.in \cup \bigcup_{\forall s \in S} s.out\right)\end{aligned} \tag{22.4}$$

Algorithm 22.1: $S \longleftarrow$ webServiceCompositionIDDFS(R)

Input: R: the composition request
Data: $maxDepth, depth$: the maximum and the current search depth
Data: in, out: current parameter sets
Output: S: a valid service composition solving R

```
1  begin
2      maxDepth ⟵ 2
3      repeat
4          S ⟵ dl_dfs_wsc(R.in, R.out, ∅, 1)
5          maxDepth ⟵ maxDepth + 1
6      until S ≠ ∅
7      Subalgorithm S ⟵ dl_dfs_wsc(in, out, composition, depth)
8      begin
9          foreach A ∈ out do
10             foreach s ∈ promising(A) do
11                 wanted ⟵ out
12                 foreach B ∈ wanted do
13                     if ∃C ∈ s.out : subsumes(B, C) then wanted ⟵ wanted \ {B}
14                 foreach D ∈ s.in do
15                     if ∄E ∈ in : subsumes(D, E) then wanted ⟵ wanted ∪ {D}
16                 comp ⟵ s ⊕ composition
17                 if wanted = ∅ then
18                     return comp
19                 else
20                     if depth < maxDepth then
                       comp ⟵ dl_dfs_wsc(in, wanted, comp, depth + 1)
21                     if comp ≠ ∅ then return comp
22         return ∅
23     end
24 end
```

In other words, a wanted parameter is either an output concept of the composition query or an input concept of any of the services in the composition candidate that has not been satisfied by neither an input parameter of the query nor by an output parameter of any service. Here we assume that the concept A wanted by service s is not also an output parameter of s. This is done for simplification purposes – the implementation has to keep track of this possibility.

The set of eliminated parameters of a service composition contains all input parameters of the services of the composition and queried output parameters of the composition request that already have been satisfied.

$$\text{eliminated}(S) = \left(R.out \cup \bigcup_{\forall s \in S} s.in \right) \setminus \text{wanted}(S) \tag{22.5}$$

Finally, the set of known concepts is the union of the input parameters defined in the composition request and the output parameters of all services in the composition candidate.

$$\text{known}(S) = R.in \cup \bigcup_{\forall s \in S} s.out \tag{22.6}$$

Instead of using these sets to build a heuristic function h, we can derive a comparator function cmp_{wsc} directly. This comparator function has the advantage that we also can apply randomized optimization methods like evolutionary algorithms based on it.

Algorithm 22.2: $r \longleftarrow \text{cmp}_{wsc}(S_1, S_2)$

Input: S_1, S_2: two composition candidates
Output: $r \in \mathbb{Z}$: indicating whether S_1 ($r < 0$) or S_2 ($r > 0$) should be expanded next

```
1  begin
2      i1 ⟵ len(wanted(S1))
3      i2 ⟵ len(wanted(S2))
4      if i1 = 0 then
5          if i2 = 0 then return len(S1) − len(S2)
6          else return −1
7      if i2 = 0 then return 1
8      e1 ⟵ len(eliminated(S1))
9      e2 ⟵ len(eliminated(S2))
10     if e1 > e2 then return 1
11     else if e1 < e2 then return −1
12     if i1 < i2 then return −1
13     else if i1 < i2 then return 1
14     if len(S1) ≠ len(S2) then return len(S1) − len(S2)
15     return len(known(S1)) − len(known(S2))
16 end
```

Algorithm 22.2 defines cmp_{wsc} which compares two composition candidates S_1 and S_2. This function can be used by a greedy search algorithm in order to decide which of the two possible solutions is more prospective. cmp_{wsc} will return a negative value if S_1 seems to be closer to a solution than S_2, a positive value if S_2 looks as if it should be examined before S_1, and zero if both seem to be equally good.

First, it compares the number of wanted parameters. If a composition has no such unsatisfied concepts, it is a valid solution. If both, S_1 and S_2 are valid, the solution involving fewer services wins. If only one of them is complete, it also wins. Otherwise, both candidates still have unsatisfied concepts. Only if both of them have the same number of satisfied parameters, we again compare the wanted concepts. For us, it was surprising that using the number of already satisfied concepts as comparison criterion with a higher priority than the number of remaining unsatisfied concepts. However, if we do so, the search algorithms perform significantly faster. If their numbers are also equal, we prefer the shorter composition candidate. If even the compositions are of the same length, we finally base the decision of the total number of known concepts. The interesting form of this comparator function is maybe caused by the special requirements of the WSC data. Nevertheless, it shows which sorts of information about a composition can be incorporated into the search.

Using such the comparator function cmp_{wsc}, we can customize the greedy search approach defined it Algorithm 17.6 on page 296 for web service composition. The function greedyComposition defined in Algorithm 22.3 performs such a greedy compositing by maintaining an internal list which is descendingly sorted according to cmp_{wsc}. In each iteration, the last element is popped from the list and either returned (if it is a valid composition) or expanded by appending services providing wanted concepts.

An Evolutionary Approach

In order to use a evolutionary algorithm to breed web service compositions, we first need to define a proper genome $\mathbb{G}$ able to represent service sequences. A straightforward yet efficient way is to use (variable-length) strings of service identifiers which can be processed by standard genetic algorithms (see Section 3.5 on page 149). A service can be identified by a number from $\mathbb{N}_0$ denoting its index in the list of all services in the registry. The genotype-phenotype mapping transforming the genotypes $g \in \mathbb{G}$ which are sequences of such identifiers to sequences of services, i. e., the phenotypes $S \in \mathbb{X}$, is thus trivial.

Algorithm 22.3: $S \longleftarrow$ greedyComposition(R)

Input: R: the composition request
Data: X: the descendingly sorted list of compositions to explore
Output: S: the solution composition found, or $\emptyset$

```
1  begin
2      X ⟵ ⋃_{∀A∈R.out} promising(A)
3      while X ≠ ∅ do
4          X ⟵ sortList_d(X, wanted)
5          S ⟵ X_[len(X)−1]
6          X ⟵ deleteListItem(X, len(X) − 1)
7          if isGoal(S) then
8              return S
9          foreach A ∈ wanted(S) do
10             foreach s ∈ promising(A) do
11                 X ⟵ addListItem(X, s ⊕ S)
12     return ∅
13 end
```

Because of the well-known string form, we could apply the standard creation, mutation, and crossover operators. However, by specifying a specialized mutation operation, we can make the search more efficient. This new operation either deletes the first service in S (via mutate_{wsc1}) or adds a promising service to S (as done in mutate_{wsc2}). Using the adjustable variable σ as a threshold we can tell the search whether it should prefer growing or shrinking the solution candidates.

$$\text{mutate}_{wsc1}(S) \equiv \begin{cases} \{s_2, s_3, .., s_{\text{len}(S)}\} & \text{if } \text{len}(S) > 1 \\ S & \text{otherwise} \end{cases} \tag{22.7}$$

$$\text{mutate}_{wsc2}(S) \equiv s \oplus S : A \in \text{wanted}(S) \wedge s \in \text{promising}(A) \tag{22.8}$$

$$\text{mutate}_{wsc}(S) \equiv \begin{cases} \text{mutate}_{wsc1}(S) & \text{if } \text{random}_u() > \sigma \\ \text{mutate}_{wsc2}(S) & \text{otherwise} \end{cases} \tag{22.9}$$

A new create operation for building the initial random configurations can be defined as a sequence of mutate_{wsc2} invocations of random length. Initially, $\text{mutate}_{wsc2}(\emptyset)$ will return a composition consisting of a single service that satisfies at least one parameter in $R.out$. We iteratively apply mutate_{wsc2} to its previous result a random number of times in order to create a new individual.

The Comparator Function and Pareto Optimization

As driving force for the evolutionary process we can reuse the comparator function cmp_{wsc} as specified as for the greedy search in Algorithm 22.2 on the preceding page. It combines multiple objectives, putting pressure towards the direction of

1. compositions which are complete,
2. small compositions,
3. compositions that resolve many unknown parameters, and
4. compositions that provide many parameters.

On the other hand, we could as well separate these single aspects into different objective functions and apply direct Pareto optimization. This has the drawback that it spreads the pressure of the optimization process over the complete Pareto frontier[12].

[12] See Section 1.2.2 on page 33 for a detailed discussion on the drawbacks of pure Pareto optimization.

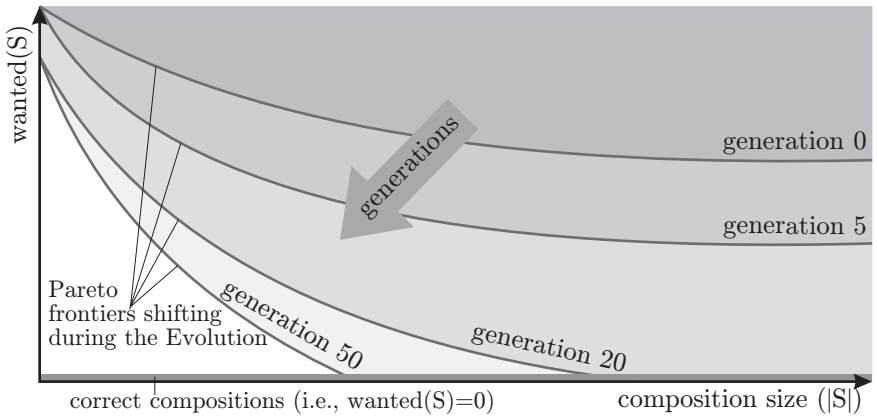


Figure 22.9: A sketch of the Pareto front in the genetic composition algorithm.

Figure 22.9 visualizes the multi-objective version of the optimization problem "web service composition" by sketching a characteristic example for Pareto frontiers of several generations of an evolutionary algorithm. We concentrate on the two dimensions *composition size* and *number of wanted (unsatisfied) parameters*. Obviously, we need to find compositions which are correct, i. e., where the latter objective is zero. On the other hand, an evolution guided only by this objective can (and will) produce compositions containing additional, useless invocations of services not related to the problem at all. The size objective is thus also required.

In the manufactured example depicted in Figure 22.9, the first five or so generations are not able to produce good compositions yet. We just can observe that longer compositions tend to provide more parameters (and have thus a lower number of wanted parameters). In generation 20, the Pareto frontier is pushed farther forward and touches the abscissa – the first correct solution is found. In the generations to come, this solution is improved and useless service calls are successively removed, so the composition size decreases. There will be a limit, illustrated as generation 50, where the shortest compositions for all possible values of wanted are found. From now on, the Pareto front cannot progress any further and the optimization process has come to a rest.

As you can see, pure Pareto optimization does not only seek for the best correct solution but also looks for the best possible composition consisting of only one service, for the best one with two service, with three services, and so on. This spreading of the population of course slows down the progress into the specific direction where wanted(S) decreases.

The comparator function cmp_{wsc} proven to be more efficient in focusing the evolution on this part of the problem space. The genetic algorithm based on it is superior in performance and hence, is used in our experiments.

Experimental Results

In Table 22.5, we illustrate the times that the algorithms introduced in this section needed to perform composition tasks of different complexity[13]. We have repeated the experiments multiple times on an off-the-shelf PC[14] and noted the mean values. The times themselves are not so important, rather are the proportions and relations between them.

[13] The test sets used here are available at `http://www.it-weise.de/documents/files/BWG2007WSC_software.zip` [accessed June 26, 2009]. Well, at least partly, I've accidentally deleted set 12 and 13. Sorry.

[14] 2GHz, Pentium IV single core with Hyper-Threading, 1GiB RAM, Windows XP, Java 1.6.0._03-b05

Test	Depth of Solution	No. of Concepts	No. of Services	IDDFS (ms)	Greedy (ms)	GA (ms)
1	5	56 210	1000	241	34	376
2	12	56 210	1000	-	51	1011
3	10	58 254	10 000	-	46	1069
4	15	58 254	2000	-	36	974
5	30	58 254	4000	-	70	6870
6	40	58 254	8000	-	63	24 117
7	1	1590	118	≤16	≤16	290
8.1	2	15 540	4480	≤16	≤16	164
8.2	2	15 540	4480	≤16	≤16	164
8.3	2	15 540	4480	≤16	≤16	164
8.4	2	15 540	4480	≤16	≤16	234
8.5	3	15 540	4480	≤16	≤16	224
8.6	3	15 540	4480	≤16	≤16	297
8.7	4	15 540	4480	18	24	283
8.8	3	15 540	4480	≤16	≤16	229
8.9	2	15 540	4480	≤16	≤16	167
11.1	8	10 890	4000	-	31	625
11.3	2	10 890	4000	-	21	167
11.5	4	10 890	4000	22 021	≤16	281
12.1	5	43 680	2000	200 320	≤16	500
12.3	7	43 680	2000	99	31	375
13	6	43 680	2000	250	32	422

Table 22.5: Experimental results for the web service composers.

The IDDFS approach can only solve smaller problems and becomes infeasible very fast. When building simpler compositions though, it is about as fast as the heuristic approach, which was clearly dominating in all categories. A heuristic may be misleading and (although this didn't happen in our experiments) could lead to a very long computation time in the worst case. Furthermore, if a problem cannot be solved, the heuristic will cannot be faster than an uninformed search. Thus, we decided to keep both, the IDDFS and the heuristic approach in our system and run them in parallel on each task if sufficient CPUs are available.

The genetic algorithm (population site 1024, tournament selection) was able to resolve all composition requests correctly for all knowledge bases and all registry sizes. It was able to build good solutions regardless how many services had to be involved in a valid solution (solution depth). In spite of this correctness, it always was a magnitude slower than the greedy search which provided the same level of correctness.

If the compositions would become more complicated or involve quality of service (QoS) aspects, it is not clear if these can be resolved with a simple heuristic. Then, the genetic algorithm could outperform greedy search approaches.

Architectural Considerations

In 2007, we introduced a more refined version [226] of our 2006 semantic composition system [225]. The architecture of this composer, as illustrated in Figure 22.10, is designed in a very general way, making it not only a challenge contribution but also part of the ADDO web service brokering system [222, 223, 224]: In order to provide the functionality of the composition algorithms to other software components, it was made accessible as a Web Service shortly after WSC'06. The web service composer is available for any system where semantic service discovery with the Ontology Web Language for Services (OWL-S) [71] or similar languages is used. Hence, this contest application is indeed also a real-world application.

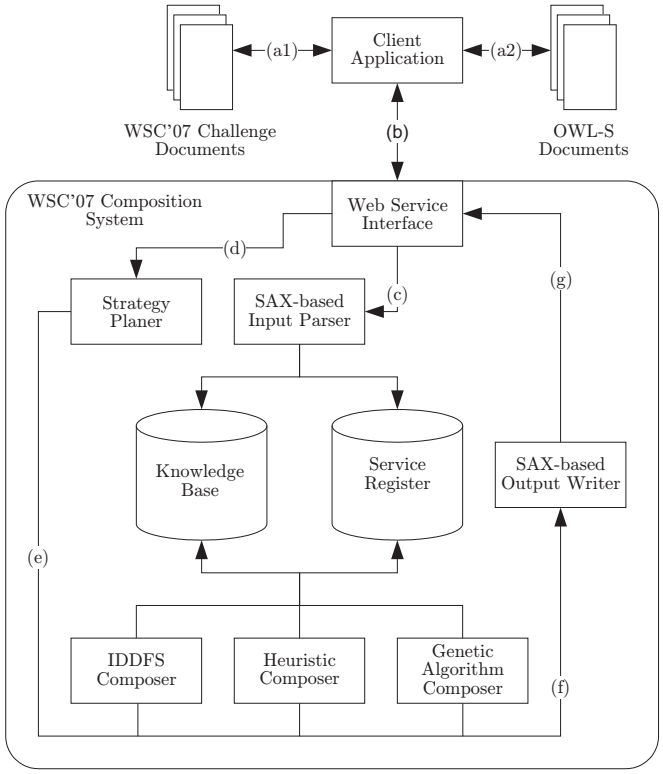


Figure 22.10: The WSC 2007 Composition System of Bleul et al. [225, 226].

An application accesses the composition system by submitting a service request (illustrated by (b)) through its *Web Service Interface*. It furthermore provides the services descriptions and their semantic annotations. Therefore, WSDL and XSD formatted files as used in the WSC challenge or OWL-S descriptions have to be passed in (($a1$) and ($a2$)). These documents are parsed by a fast *SAX-based Input Parser* (c). The composition process itself is started by the *Strategy Planer* (d). The Strategy Planer chooses an appropriate composition algorithm and instructs it with the composition challenge document (e).

The software modules containing the basic algorithms all have direct access to the *Knowledge Base* and to the *Service Register*. Although every algorithm and composition strategy is unique, they all work on the same data structures. One or more composition algorithm modules solve the composition requests and pass the solution to a *SAX-based Output Writer*, an XML document generating module (f) faster than DOM serialization. Here it is also possible to transform it to, for example, BPEL4WS [989] descriptions. The result is afterwards returned through the Web Service Interface (g).

One of the most important implementation details is the realization of the operation "promising" since it is used by all composition algorithms in each iteration step. Therefore, we transparently internally merge the knowledge base and the service registry. This step is described here because it is very crucial for the overall system performance.

A semantic concept is represented by an instance of the class `Concept`. Each instance of `Concept` holds a list of services that directly produce a parameter annotated with it as output. The method `getPromisingServices(`A`)` of `Concept`, illustrated in Figure 22.11, additionally returns all the `Service`s that provide a specialization of the concept A as output. In order to determine this set, all the specializations of the concept have to be traversed and their promising services have to be accumulated. The crux of the routine is that this costly traversal is only performed once per concept. Our experiments substantiated

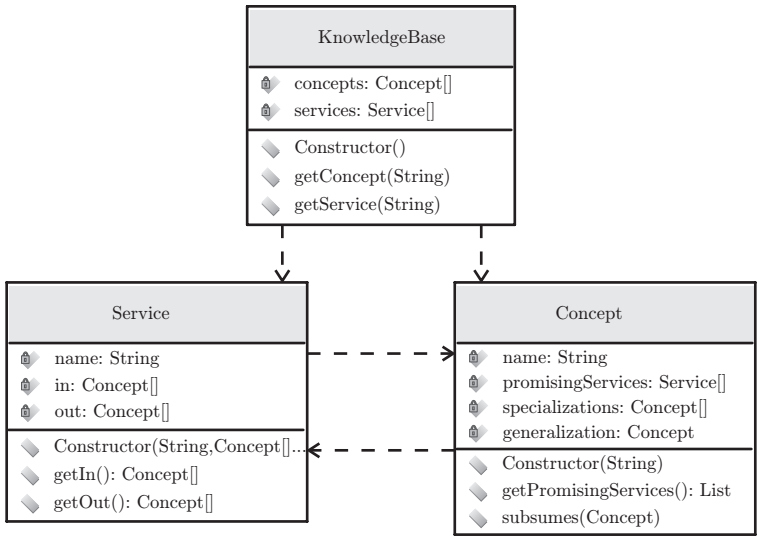


Figure 22.11: The Knowledge Base and Service Registry of our Composition System.

that the resource *memory*, even for largest service repositories, is not a bottleneck. Hence, `getPromisingServices` caches its results.

This caching is done in a way that is thread-safe on one hand and does not need any synchronization on the other. Each instance `X` of `Concept` holds an internal variable `promisingServices` which is initially `null`. If `X.getPromisingServices()` is invoked, it first looks up if `X.promisingServices` is `null`. If so, the list of promising services is computed, stored in `X.promisingServices`, and returned. Otherwise, `X.promisingServices` is returned directly. Since we do not synchronize this method, it may be possible that the list is computed concurrently multiple times. Each of these computations will produce the same result. Although all parallel invocations of `x.getPromisingServices()` will return other lists, their content is the same. The result of the computation finishing last will remain in `x.promisingServices` whereas the other lists will get lost and eventually be freed by the garbage collector. Further calls to `x.getPromisingServices()` always will yield the same, lastly stored, result. This way, we can perform caching which is very important for the performance and spare costly synchronization while still granting a maximum degree of parallelization.

Conclusions

In order to solve the 2006 and 2007 Web Service Challenges, we utilized three different approaches, an uninformed search, an informed search, and a genetic algorithm. The uninformed search proofed generally unfeasible for large service repositories. It can only provide a good performance if the resulting compositions are very short.

However, in the domain of web service composition, the maximum number of services in a composition is only limited by the number of services in the repositories and cannot be approximated by any heuristic. Therefore, any heuristic or metaheuristic search cannot be better than the uninformed search in the case that a request is sent to the composer which cannot be satisfied. This is one reason why the uninformed approach was kept in our system, along with its reliability for short compositions.

Superior performance for all test sets could be obtained by utilizing problem-specific information encapsulated in a fine-tuned heuristic function to guide a greedy search. This approach is more efficient than the other two tested variants by a magnitude.

Genetic algorithms are much slower, but were also always able to provide correct results to all requests. To put it simple, the problem of semantic composition as defined in the context of the WSC is not complicated enough to fully unleash the potential of evolutionary algorithms. They cannot cope with the highly efficient heuristic used in the greedy search. We anticipate however, that, especially in practical applications, additional requirements will be imposed onto a service composition engine. Such requirements could include quality of service (QoS), the question for optimal parallelization, or the generation of complete BPEL [1071] processes. In this case, heuristic search will most probably become insufficient but genetic algorithms and Genetic Programming [1196] will still be able to deliver good results.

23

Real-World Applications

In this chapter, we will explore real-world applications of global optimization techniques. Such applications are well-researched and established to a point where people are willing to bet money on them. They can safely be utilized in a productive system. Some of these areas where global optimization algorithms are applied in a practical fashion, aiding scientists and engineers with their work, are summarized in this chapter.

23.1 Symbolic Regression

In statistics, regression analysis examines the unknown relation $\varphi :\in \mathbb{R}^n \mapsto \in \mathbb{R}$ of a dependent variable $y \in \mathbb{R}$ to specified independent variables $\mathbf{x} \in \mathbb{R}^m$. Since φ is not known, the goal is to find a reasonable good approximation $\psi^\star$.

Definition 23.1 (Regression). Regression[1] [1150, 739, 631, 595] is a statistic technique used to predict the value of a variable which is dependent one or more independent variables.

The result of the regression process is a function $\psi^\star : \mathbb{R}^m \mapsto \mathbb{R}$ that relates the m independent variables (subsumed in the vector $\mathbf{x}$ to one dependent variable $y \approx \psi^\star(\mathbf{x})$. The function $\psi^\star$ is the best estimator chosen from a set Ψ of candidate functions $\psi : \mathbb{R}^m \mapsto \psi$. Regression is strongly related to the estimation theory outlined in Section 28.7 on page 499. In most cases, like linear[2] or nonlinear[3] regression, the mathematical model of the candidate functions is not completely free. Instead, we pick a specific one from an array of parametric functions by finding the best values for the parameters.

Definition 23.2 (Symbolic Regression). Symbolic regression [1196, 87, 2270, 1791, 1792, 606, 607, 1112, 1699] is one of the most general approaches to regression. It is not limited to determining the optimal values for the set of parameters of a certain array of functions. Instead, regression functions can be constructed by combining elements of a set of mathematical expressions, variables and constants.

23.1.1 Genetic Programming: Genome for Symbolic Regression

One of the most widespread methods to perform symbolic regression is to apply Genetic Programming. Here, the candidate functions are constructed and refined by an evolutionary process. In the following we will discuss the genotypes (which are also the phenotypes) of the evolution as well as the objective functions that drive it. As illustrated in Figure 23.1, the solution candidates, i. e., the candidate functions, are represented by a tree of mathematical expressions where the leaf nodes are either constants or the fields of the independent variable vector $\mathbf{x}$.

[1] http://en.wikipedia.org/wiki/Regression_analysis [accessed 2007-07-03]
[2] http://en.wikipedia.org/wiki/Linear_regression [accessed 2007-07-03]
[3] http://en.wikipedia.org/wiki/Nonlinear_regression [accessed 2007-07-03]

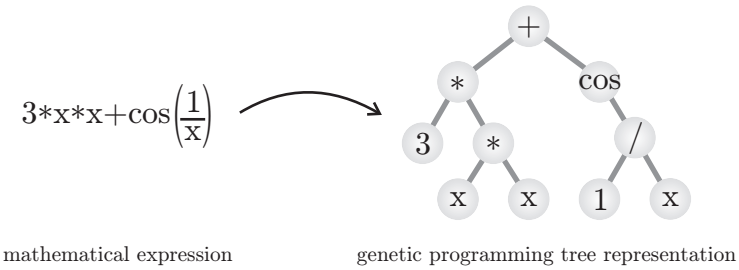


Figure 23.1: An example genotype of symbolic regression of with $x = \mathbf{x} \in \mathbb{R}^1$.

The set Ψ of functions ψ that can possible be evolved is limited by the set of expressions Σ available to the evolutionary process.

$$\Sigma = \{+, -, *, /, \exp, \ln, \sin, \cos, \max, \min, \dots\} \tag{23.1}$$

Another aspect that influences the possible results of the symbolic regression is the concept of constants. In general, constants are not really needed since they can be constructed indirectly via expressions. The constant 2.5, for example, equals the expression $\frac{x}{x+x} + \frac{\ln x * x}{\ln x}$. The evolution of such artificial constants, however, takes rather long. Koza [1196] has therefore introduced the concept of ephemeral random constants.

Definition 23.3 (Ephemeral Random Constants). If a new individual is created and a leaf in its expression-tree is chosen to be an ephemeral random constant, a random number is drawn uniformly distributed from a reasonable interval. For each new constant leaf, a new constant is created independently. The values of the constant leafs remain unchanged and can be moved around and copied by crossover operations.

According to Koza's idea ephemeral random constants remain unchanged during the evolutionary process. In our work, it has proven to be practicable to extend his approach by providing a mutation operation that changes the value c of a constant leaf of an individual. A good policy for doing so is by replacing the old constant value c_{old} by a new one c_{new} which is a normally distributed random number with the expected value c_{old} (see Definition 28.70 on page 528):

$$c_{new} = \text{random}_n\left(c_{old}, \sigma^2\right) \tag{23.2}$$

$$\sigma^2 = e^{-\text{random}_u(0,10)} * |c_{old}| \tag{23.3}$$

Notice that the other reproduction operators for tree genomes have been discussed in detail in Section 4.3 on page 162.

23.1.2 Sample Data, Quality, and Estimation Theory

In the following elaborations, we will reuse some terms that we have applied in our discussion on likelihood in Section 28.7.2 on page 500 in order to find out what measures will make good objective functions for symbolic regression problems.

Again, we are given a finite set of sample data A containing $n = |A|$ pairs of $(\mathbf{x}_i, y_i)$ where the vectors $\mathbf{x}_i \in \mathbb{R}^m$ are known inputs to an unknown function $\varphi : \mathbb{R}^m \mapsto \mathbb{R}$ and the scalars y_i are its observed outputs (possible contaminated with noise and measurement errors subsumed in the term η_i, see Equation 28.237 on page 500). Furthermore, we can access a (possible infinite large) set Ψ of functions $\psi : \mathbb{R}^m \mapsto \mathbb{R} \in \Psi$ which are possible estimators of φ. For the inputs $\mathbf{x}_i$, the results of these functions ψ deviate by the estimation error ε (see Definition 28.53 on page 499) from the y_i.

$$y_i = \varphi(\mathbf{x}_i) + \eta_i \; \forall i \in [0..n-1] \tag{23.4}$$

$$y_i = \psi(\mathbf{x}_i) + \varepsilon_i(\psi) \; \forall \psi \in \Psi, \; i \in [0..n-1] \tag{23.5}$$

In order to guide the evolution of estimators (in other words, for driving the regression process), we need an objective function that furthers solution candidates that represent the sample data A and thus, resemble the function φ, closely. Let us call this "driving force" quality function.

Definition 23.4 (Quality Function). The quality function $f(\psi, A)$ defines the quality of the approximation of a function φ by a function ψ. The smaller the value of the quality function is, the more precisely is the approximation of φ by ψ in the context of the sample data A.

Under the conditions that the measurement errors η_i are uncorrelated and are all normally distributed with an expected value of zero and the same variance (see Equation 28.238, Equation 28.239, and Equation 28.240 on page 500), we have shown in Section 28.7.2 that the best estimators minimize the mean square error MSE (see Equation 28.253 on page 502, Definition 28.60 on page 503 and Definition 28.56 on page 499). Thus, if the source of the values y_i complies at least in a simplified, theoretical manner with these conditions or even is a real measurement process, the square error is the quality function to choose.

$$f_{\sigma \neq 0}(\psi, A) = \sum_{i=0}^{\text{len}(A)-1} \left(y_i - \psi(\mathbf{x}_i)\right)^2 \tag{23.6}$$

While this is normally true, there is one exception to the rule: The case where the values y_i are no measurements but direct results from φ and $\eta = 0$. A common example for this situation is if we apply symbolic regression in order to discover functional identities [1785, 1527, 1196] (see also Section 23.1.3). Different from normal regression analysis or estimation, we then know φ exactly and want to find another function $\psi^\star$ that is another, equivalent form of φ. Therefore, we will use φ to create sample data set A beforehand, carefully selecting characteristic points $\mathbf{x}_i$. Thus, the noise and the measurement errors η_i all become zero. If we would still regard them as normally distributed, their variance s^2 would be zero, too.

The proof for the statement that minimizing the square errors maximizes the likelihood is based on the transition from Equation 28.248 to Equation 28.249 on page 502 where we cut divisions by s^2. This is not possible if σ becomes zero. Hence, we may or may not select metrics different from the square error as quality function. Its feature of punishing larger deviation stronger than small ones, however, is attractive even if the measurement errors become zero. Another metric which can be used as quality function in these circumstances are the sums of the absolute values of the estimation errors:

$$f_{\sigma=0}(\psi, A) = \sum_{i=0}^{\text{len}(A)-1} |y_i - \psi(\mathbf{x}_i)| \tag{23.7}$$

23.1.3 An Example and the Phenomenon of Overfitting

If multi-objective optimization can be applied, the quality function should be complemented by an objective function that puts pressure in the direction of smaller estimations ψ. In symbolic regression by Genetic Programming, the problem of code bloat (discussed in Section 4.10.3 on page 224) is eminent. Here, functions do not only grow large because they include useless expressions (like $\frac{x*x+x}{x} - x - 1$). A large function may consist of functional

expressions only, but instead of really representing or approximating φ, it is degenerated to just some sort of misfit decision table. This phenomenon is called overfitting and has initially been discussed in Section 1.4.8 on page 72.

Let us, for example, assume we want to find a function similar to Equation 23.8. Of course, we would hope to find something like Equation 23.9.

$$y = \varphi(x) = x^2 + 2x + 1 \tag{23.8}$$

$$y = \psi^\star(x) = (x+1)^2 = (x+1)(x+1) \tag{23.9}$$

i	x_i	$y_i = \varphi(x_i)$	$f_2^\star(x_i)$
01	−5	16	15.59
1	−4.9	15.21	15.40
2	0.1	1.21	1.11
3	2.9	15.21	15.61
4	3	16	16
5	3.1	16.81	16.48
6	4.9	34.81	34.54
7	5	36	36.02
8	5.1	37.21	37.56

Table 23.1: Sample Data $A = \{(x_i, y_i) : i \in [0..8]\}$ for Equation 23.8

For testing purposes, we choose randomly the nine sample data points listed in Table 23.1. As result of Genetic Programming based symbolic regression we may obtain something like Equation 23.10, outlined in Figure 23.2, which represents the data points quite precisely but has nothing to do with the original form of our equation.

$\psi_2^\star(x) =$ (((((0.934911896352446 * 0.258746335682841) - (x * ((x / ((x - 0.763517999368926) + (0.0452368900127981 - 0.947318140392111))) / ((x - (x + x)) + (0.331546588012695 * (x + x)))))) + 0.763517999368926) + ((x - (((0.934911896352446 * ((0.934911896352446 / x) / (x + 0.947390132934724))) + ((x * 0.235903629190878) * (x - 0.331546588012695)) + ((x * x) + x))) / x)) * (((((x - (x * (0.258746335682841 / 0.455160839551232))) / (0.0452368900127981 - 0.763517999368926)) * x) * (0.763517999368926 * 0.947318140392111)))) - ((((((x - (x * (0.258746335682841 / 0.455160839551232))) / (0.0452368900127981 - 0.763517999368926)) * 0.763517999368926) * x) + (x - (x * (0.258746335682841 * 0.934911896352446))))) (23.10)

We obtained both functions $\psi_1^\star$ (in its second form) and $\psi_2^\star$ using the symbolic regression applet of Hannes Planatscher which can be found at `http://www.potschi.de/sr/` [accessed 2007-07-03][4]. It needs to be said that the first (wanted) result occurred way more often than absurd variations like $\psi_2^\star$. But indeed, there are some factors which further the evolution of such eyesores:

1. If only few sample data points are provided, the set of prospective functions that have a low estimation error becomes larger. Therefore, chances are that symbolic regression provides results that only match those points but differ in all other points significantly from φ.
2. If the sample data points are not chosen wisely, their expressiveness is low. We for instance chose 4.9,5, and 5.1 as well as 2.9, 3 and 3.1 which form two groups with members very close to each other. Therefore, a curve that approximately hits these two clouds is rated automatically with a high quality value.

[4] Another good applet for symbolic regression can be found at `http://alphard.ethz.ch/gerber/approx/default.html` [accessed 2007-07-03]

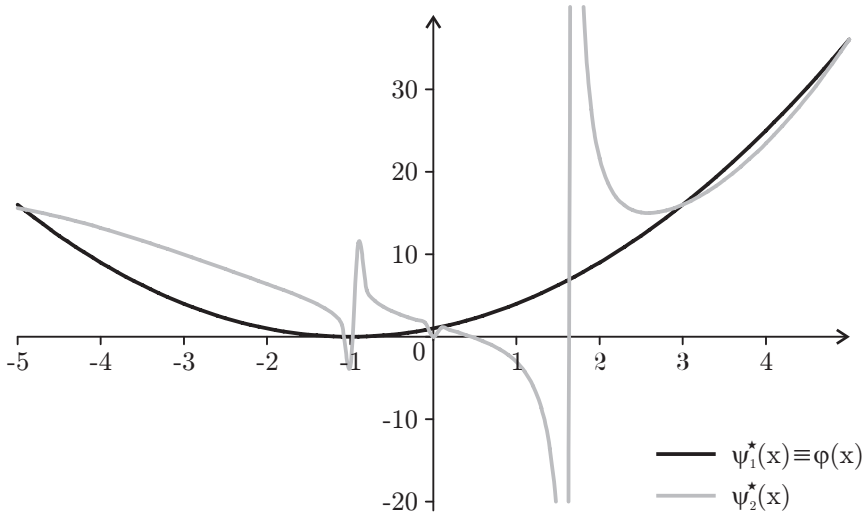


Figure 23.2: $\varphi(x)$, the evolved $\psi_1^\star(x) \equiv \varphi(x)$, and $\psi_2^\star(x)$.

3. A small population size decreases the diversity and furthers "incest" between similar solution candidates. Due to a lower rate of exploration, only a local minimum of the quality value will often be yielded.
4. Allowing functions of large depth and putting low pressure against bloat (see Section 4.10.3 on page 224) leads to uncontrolled function growth. The real laws φ that we want to approximate with symbolic regression do usually not consist of more than 40 expressions. This is valid for most physical, mathematical, or financial equations. Therefore, the evolution of large functions is counterproductive in those cases.

Although we made some of these mistakes intentionally, there are many situations where it is hard to determine good parameter sets and restrictions for the evolution and they occur accidentally.

23.1.4 Limits of Symbolic Regression

Often, we cannot obtain an optimal approximation of φ, especially if φ cannot be represented by the basic expressions available to the regression process. One of these situations has already been discussed before: the case where φ has no closed arithmetical expression. Another possibility is that the regression method tries to generate a polynomial that approximates the φ, but φ does contain different expressions like sin or e^x or polynomials of an order higher than available. Yet another problem is that the values y_i are often not results computed by φ directly but could, for example, be measurements taken from some physical entity and we want to use regression to determine the interrelations between this entity and some known parameters. Then, the measurements will be biased by noise and systematic measurement errors. In this situation, $f(\psi^\star, A)$ will be greater than zero even after a successful regression.

23.2 Global Optimization of Distributed Systems

23.2.1 Introduction

Optimization algorithms are methods for finding optimal configurations of different features of their solution candidates. Many aspects of distributed systems are configurable or depend

on parameter settings, such as the topology, security, and routing. Hence, there is a huge potential for using global optimization algorithms in order to improve them.

And indeed, this potential is widely utilized. The study by Sinclair [1886] from 1999 reported that more than 120 papers had been published on work which employed Evolutionary Computation for optimizing network topologies and dimension, node placement, routing, and wavelength or frequency allocation. The comprehensive master's thesis by Kampstra from 2005 [1087, 1088] builds on this aforementioned study and classifies over 400 papers. According to Kampstra, communication networks was the field with the most researchers listed in EvoWeb, the European Network of Excellence in Evolutionary Computing, in 2005. The first workshop on this topic, *Evolutionary Telecommunications* [1889], took place in 1999. In the year 2000 alone, two books ([450] and [1630]) have been published on the application of Evolutionary Computation to networking. Further summary papers appeared around the same time [1851, 1629, 2033, 2109]. The recent studies from Alba and Chicano [31] and Cortés Achedad et al. [453] as well as the high number of papers published every year show that the interest in applying global optimization techniques in this problem domain has by no means decreased.

Most of the mentioned summaries concentrate on giving an overview in form of a more prosaic version of paper listings. We [2186] provide such a listing in a condensed form in Section 23.2.3, but focus on giving clear and detailed in-depth discussions of multiple example applications and also introduce the optimization algorithms utilized in them. This way, the subject becomes more tangible for audience which is rooted in only one the two involved subject areas.

We studied more than 130 papers from two decades of research in evolutionary telecommunication. Figure 23.3 illustrates how these papers distribute over the time from 1987 to 2008. The papers are classified according to the area of application, their optimization

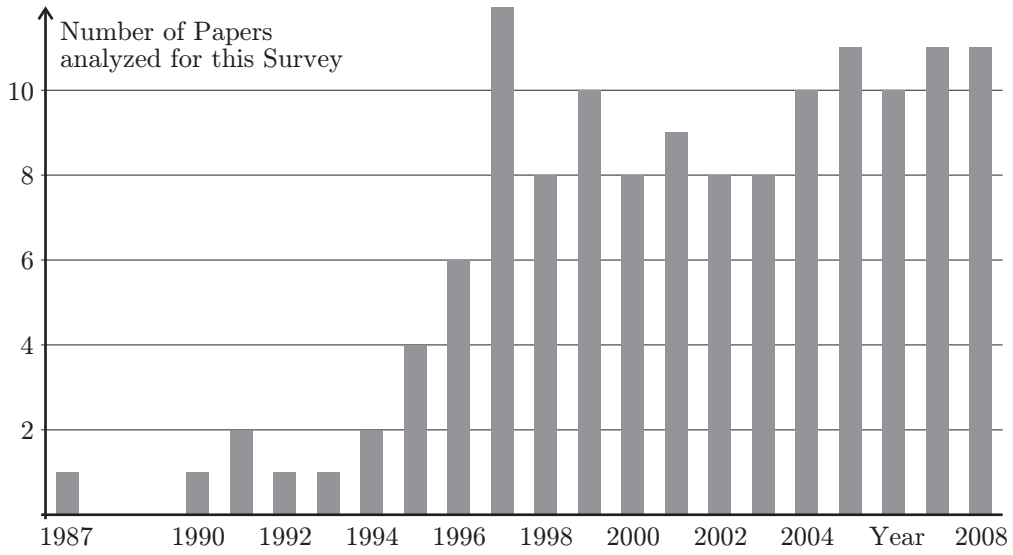


Figure 23.3: The number of papers studied for this survey per year.

goals, problem representations, and the optimization algorithms utilized. Figure 23.4 gives an overview of which areas were tackled by the researchers and which optimization algorithms were applied in the papers we studied. Here, it is important to notice that a paper may deal with multiple applications at once (like routing algorithms which also perform load balancing) and thus may appear in multiple columns. The complete subject catalog resulting from our survey can be found in Section 23.2.3. Such a list, however, gives only a limited

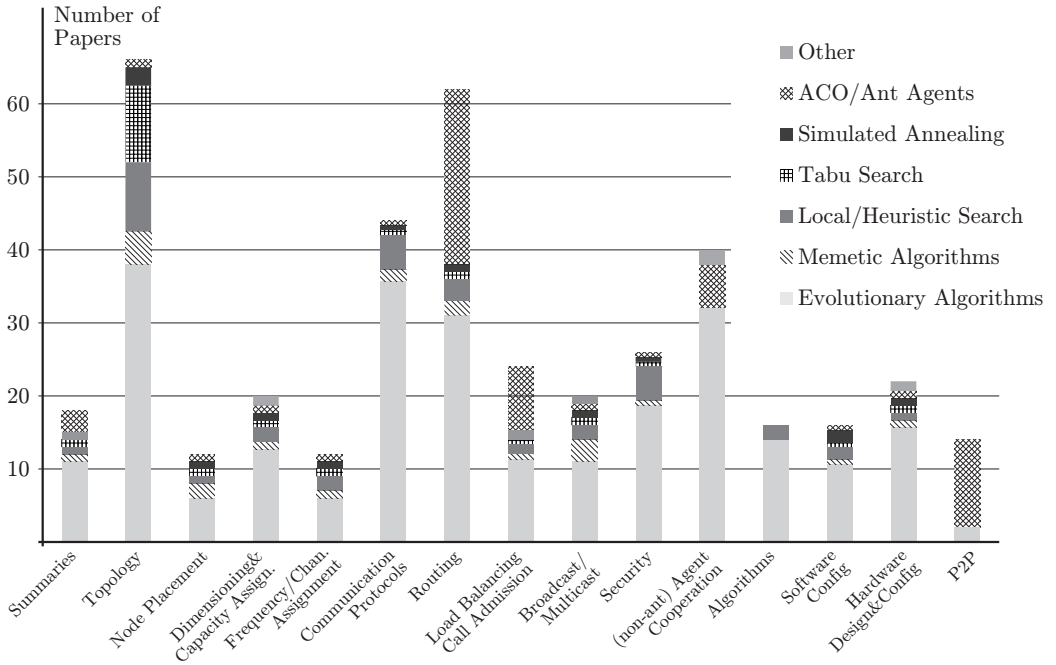


Figure 23.4: The number of papers analyzed, broken down to application area and optimization method.

idea about the actual approaches that have been developed. Therefore, we will use the following sections to first take a deeper look into some interesting optimization approaches from various areas of distributed systems which stand exemplary for the variety and the potential of this field of research. Different methods to synthesize or to improve network topologies are outlined in **??**, adaptive or evolved routing protocols will be discussed in **??**, and different approaches to the generation of protocols with global optimization algorithms are summarized in Section 23.2.2. In **??**, we illustrate some security aspects and how they were optimized by different research groups before ending our overview on applications with software configuration and parameter adaption approaches in **??**. After a representative list of publications in from all these research areas (Section 23.2.3), we conclude this summary in Section 23.2.4.

TODO some content temporarily disabled

23.2.2 Synthesizing Protocols

Protocols like IP [1013] and TCP [362] are the rules for message and information exchange in a network. Depending on the application, protocols can become arbitrarily complex and strongly influence the efficiency and robustness of a distributed system.

TODO some content temporarily disabled

Evolving Fraglet-based Protocols

In Section 4.9.2, we have outlined the Fraglet language. This form of protocol representation is predestined for automated synthesis via evolutionary algorithms: Fraglets have almost no syntactical constraints and can represent complicated protocols in a compact manner. Similar to us, Tschudin investigated the offline evolution of protocols using a genetic algorithm.

In his work, a complete communication system was simulated for a given number of time steps during the evaluation of each Fraglet program. The objective values denote the correlation of the behavior observed during the simulation and the target behavior. Tschudin concluded that evolutionary methods are suitable to optimize existing Fraglet protocols, but also indicated that the evolution of new distributed algorithms is difficult because of a strong tendency to overfitting (see Section 1.4.8) and the all-or-nothing problem known from Genetic Programming (see Section 4.10.2 on page 223).

Online Protocol Adaptation and Evolution with Fraglets

Autonomic networks are networks where manual management is not desired or hard to realize, such as systems of hundreds of gadgets in an e-home, sensor networks, or arbitrary mesh networks with wireless and wired links. Yamamoto and Tschudin [2275] pointed out that software in such networks should be self-modifying so as to be able to react to unforeseen network situations. They distinguish two forms of such reactions – adaption and evolution. Adaption is the short-term reconfiguration of existing modules whereas evolution stands for the modification of old and the discovery of new functionality and happens at a larger timescale. Software with these abilities probably cannot predict whether the effects of a modification are positive or negative in advance and therefore, needs to be resilient in order to mitigate faulty code that could evolve. In [2059], Tschudin and Yamamoto showed that such resilience can be achieved to a certain degree by introducing redundancy into Fraglet protocols.

Complementing Tschudin's work on offline protocol synthesis and optimization [2058], Yamamoto and Tschudin [2275] describe online protocol evolution as a continuously ongoing, decentralized, and asynchronous process of constant competition and selection of the most feasible modules. Genetic Programming with mutation and homologous crossover is chosen for accomplishing these features. The fitness measure (subject to maximization) is the performance of the protocols as perceived by the applications running in the network. The score of a solution candidate (i. e., a protocol) is incremented if it behaves correctly and decremented whenever an error is detected. The resource consumption in terms of the memory allocated by the protocols is penalized proportionally.

Yamamoto and Tschudin [2273, 2274] create populations containing a mix of different confirmed delivery and reliable delivery protocols for messages. These populations were then confronted with either reliable or unreliable transmission challenges and were able to adapt to these conditions quickly. If the environment changes afterwards, when a formerly reliable channel becomes unreliable, for example, the degree of re-adaptation was, however, unsatisfying. The loss of diversity due to the selection of only highly fit protocols during the adaptation phase could not yet be compensated by mutation in these first experiments.

Further information on approaches for evolutionary online optimization of communication protocols can be found in the report *Framework for Distributed On-line Evolution of Protocols and Services, 2nd Edition* from the EU-sponsored project BIONETS [1429].

23.2.3 Paper List

In this section, we list the papers concerning the optimization of distributed systems. This concise list groups the papers according to the area of application, the optimization goals, the problem representations, and the optimization algorithms utilized. This collection lists a wide variety of approaches developed by a large number of authors (nearly 200 authors

are involved in the papers listed). In our opinion, this heterogeneity and distribution should be interpreted as a clear indicator that the optimization of distributed systems lends itself to heuristic and metaheuristic approaches. Many papers provide engineering-level solutions which often deliver excellent results.

Topology Optimization and Terminal Assignment

General Networks or Theory

1. Abuali et al. [10, 11] (1994) *aims*: synthesis, costs; *representation*: integer strings; *optimization methods*: evolutionary algorithms and local search, see also **??**
2. Khuri and Chiu [1133] (1997) *aims*: synthesis, costs; *representations*: bit strings and integer strings; *optimization methods*: evolutionary algorithms and local search, see also **??**
3. Salcedo-sanz and Yao [1788] (2004) *aims*: synthesis, costs; *representations*: bit strings and integer strings; *optimization method*: evolutionary algorithms
4. Lehmann and Kaufmann [1270, 2335] (2005–2007) *aims*: synthesis, self-organization, QoS features, dynamic or adaptive behavior; *representation*: information distributed over the network; *optimization method*: evolutionary algorithms, see also **??**

Computer Networks in General

5. Michalewicz [1405] (1991) *aims*: synthesis, robustness; *optimization method*: evolutionary algorithms, see also **??**
6. Kumar et al. [1220] (1993) *aims*: synthesis, robustness, QoS features; *representation*: bit strings; *optimization method*: evolutionary algorithms, see also **??**
7. Pierre and Legault [1644, 1645] (1996–1998) *aims*: synthesis, QoS features, costs; *representation*: bit strings; *optimization method*: evolutionary algorithms
8. Ko et al. [1164] (1997) *aims*: synthesis, QoS features, costs; *representation*: bit strings; *optimization methods*: evolutionary algorithms and local search
9. Dengiz et al. [555] (1997) *aims*: synthesis, robustness, costs; *representation*: integer strings; *optimization methods*: evolutionary algorithms and Memetic Algorithms
10. Montana et al. [1445] (2002–2004) *aims*: QoS features, dynamic or adaptive behavior; *representation*: integer strings; *optimization method*: evolutionary algorithms
11. Yao et al. [2286] (2005) *aims*: synthesis, costs; *representation*: trees; *optimization methods*: evolutionary algorithms and local search, see also **??**

Wireless or Mobile Computer Networks

12. Lai et al. [1231, 1232] (2007) *aims*: synthesis, robustness; *representation*: integer strings; *optimization method*: evolutionary algorithms

Telecommunication Networks in General

13. Dengiz et al., see entry 9.
14. Pierre and Elgibaoui [1641] (1997) *aims*: synthesis, robustness, QoS features, costs; *optimization method*: Tabu Search

Wireless or Mobile Telecommunication Networks

15. Pierre and Houéto [1643, 1642] (2002) *aims*: synthesis, costs; *representation*: bit strings; *optimization method*: Tabu Search
16. Quintero and Pierre [1685, 1683, 1684] (2002–2003) *aim*: costs; *representation*: integer strings; *optimization methods*: evolutionary algorithms, local search, Memetic Algorithms, Tabu Search, and Simulated Annealing

17. St-Hilaire et al. [1953, 1952] (2006) *aims*: synthesis, costs; *optimization methods*: local search and Tabu Search
18. Salcedo-Sanz et al. [1789, 1790] (2008) *aims*: synthesis, QoS features, costs; *representation*: bit strings; *optimization methods*: evolutionary algorithms and Memetic Algorithms

Optical Networks in General

19. Sinclair [1885, 23, 1887, 1888] (1995–2000) *aims*: synthesis, robustness, costs; *representations*: bit strings and trees plus genotype-phenotype mappings; *optimization method*: evolutionary algorithms, see also **??**
20. Brittain et al. [290] (1997) *aims*: synthesis, costs; *representations*: bit strings and integer strings; *optimization methods*: evolutionary algorithms and local search

Node Placement

21. Alba et al. [35] (2002) *aims*: synthesis, robustness, costs; *representation*: bit strings; *optimization method*: evolutionary algorithms
22. Salcedo-Sanz et al., see entry 18.

Dimensioning and Capacity Assignment

Computer Networks in General

23. Coombs and Davis [441] (1987) *aim*: QoS features; *optimization method*: evolutionary algorithms, see also **??**
24. Ko et al., see entry 8.
25. Martin et al. [1363] (2008) *aim*: synthesis; *representation*: integer strings; *optimization methods*: evolutionary algorithms, Extremal Optimization, and Particle Swarm Optimization

Telecommunication Networks in General

26. Martin et al., see entry 25.

Frequency and Channel Assignment

27. Tan and Sinclair [2004] (1995) *aims*: synthesis, costs; *representation*: bit strings; *optimization methods*: evolutionary algorithms and local search

Protocol Generation and Optimization

General Networks or Theory

28. Mackin and Tazaki [1340, 1341, 1342] (1999–2002) *aim*: synthesis; *representation*: trees; *optimization method*: evolutionary algorithms, see also **??**

Computer Networks in General

29. El-Fakihy et al. [2272, 628] (1995–1999) *aims*: synthesis, QoS features; *representation*: bit strings; *optimization methods*: evolutionary algorithms and Memetic Algorithms, see also **??**
30. Sharples and Wakeman [1860, 1862, 1861] (1999–2001) *aims*: synthesis, robustness, QoS features; *representation*: bit strings plus genotype-phenotype mappings; *optimization method*: evolutionary algorithms, see also **??**

31. Song et al. [1637, 1918] (2000–2001) *aims*: synthesis, QoS features; *representation*: trees; *optimization method*: local search, see also **??**
32. Grace [847] (2000) *aims*: synthesis, robustness, QoS features; *representation*: trees; *optimization method*: evolutionary algorithms, see also **??**
33. Ye and Kalyanaraman [2290, 2289] (2001–2003) *aims*: QoS features, dynamic or adaptive behavior; *representation*: real vectors
34. Van Belle et al. [2089, 2090] (2001–2003) *aims*: synthesis, robustness, QoS features; *representation*: bit strings; *optimization method*: evolutionary algorithms, see also **??**
35. de Araújo et al. [504, 505] (2003) *aim*: synthesis; *representation*: integer strings; *optimization method*: evolutionary algorithms, see also **??**
36. Tschudin [2058] (2003) *aims*: synthesis, robustness; *representation*: linear programs; *optimization method*: evolutionary algorithms, see also Section 23.2.2
37. Yamamoto and Tschudin [2274, 2275, 2273] (2005) *aims*: synthesis, robustness, dynamic or adaptive behavior; *representations*: information distributed over the network and linear programs; *optimization method*: evolutionary algorithms, see also Section 23.2.2

Wireless or Mobile Computer Networks

38. Montana and Redi [1444] (2005) *aim*: QoS features; *representation*: real vectors; *optimization method*: evolutionary algorithms
39. Weise et al. [2180, 2187] (2007–2008) *aims*: synthesis, dynamic or adaptive behavior; *optimization method*: evolutionary algorithms

Routing

General Networks or Theory

40. Christensen et al. [401] (1997) *aims*: QoS features, costs; *representation*: integer strings; *optimization method*: evolutionary algorithms
41. Kirkwood et al. [1143] (1997) *aims*: synthesis, robustness; *representation*: trees; *optimization method*: evolutionary algorithms, see also **??**
42. Zhu et al. [2324] (1998) *aim*: synthesis; *representation*: integer strings; *optimization methods*: evolutionary algorithms and local search

Computer Networks in General

43. Kirkwood et al., see entry 41.
44. Munetomo et al. [1487, 1488, 1489, 1484] (1997–1999) *aims*: self-organization, robustness, dynamic or adaptive behavior; *representations*: integer strings and information distributed over the network; *optimization method*: evolutionary algorithms, see also **??**
45. Ko et al., see entry 8.
46. Di Caro and Dorigo [561, 560, 559] (1998–2004) *aims*: self-organization, robustness, dynamic or adaptive behavior; *representation*: information distributed over the network; *optimization method*: ACO/ant agents, see also **??**
47. Bonabeau et al. [245] (1999) *aim*: dynamic or adaptive behavior; *representation*: information distributed over the network; *optimization method*: ACO/ant agents
48. Fei et al. [647] (1999) *aims*: robustness, QoS features; *representation*: bit strings
49. Liang et al. [1281, 1282] (2002–2006) *aims*: robustness, dynamic or adaptive behavior; *representation*: information distributed over the network; *optimization methods*: evolutionary algorithms and ACO/ant agents, see also **??**
50. Sim and Sun [1880] (2002) *representation*: information distributed over the network; *optimization method*: ACO/ant agents

Telecommunication Networks in General

51. Cox, Jr. et al. [461] (1991) *aims*: QoS features, costs, dynamic or adaptive behavior; *representation*: integer strings; *optimization method*: evolutionary algorithms
52. Schoonderwoerd et al. [1832, 1833, 1834] (1996–1997) *aims*: synthesis, self-organization, robustness, QoS features, dynamic or adaptive behavior; *representation*: information distributed over the network; *optimization method*: ACO/ant agents, see also **??**
53. Christensen et al., see entry 40.
54. Zhu et al., see entry 42.
55. Lukschandl et al. [1329, 1330, 1331] (1999–2000) *aims*: robustness, costs; *representation*: linear programs; *optimization method*: evolutionary algorithms, see also Section 4.6.5
56. Galiasso and Wainwright [759] (2001) *aims*: synthesis, costs; *representation*: integer strings; *optimization methods*: evolutionary algorithms and Memetic Algorithms
57. Sandalidis et al. [1798] (2001) *aim*: dynamic or adaptive behavior; *representation*: information distributed over the network; *optimization method*: ACO/ant agents

Optical Networks in General

58. Wang et al. [2155] (2004) *aim*: QoS features; *optimization method*: evolutionary algorithms

Load Balancing and Call Admission

Computer Networks in General

59. Munetomo et al., see entry 44.
60. Oates and Corne [1553] (2001) *aim*: QoS features; *representation*: integer strings; *optimization methods*: evolutionary algorithms, local search, and Simulated Annealing
61. Zapf and Weise [2311, 2310] (2007) *aims*: synthesis, self-organization; *representation*: bit strings; *optimization method*: evolutionary algorithms

Telecommunication Networks in General

62. Schoonderwoerd et al., see entry 52.

Peer-To-Peer Systems

63. Iles and Deugo [1011] (2002) *aims*: robustness, dynamic or adaptive behavior; *representation*: trees; *optimization method*: evolutionary algorithms, see also **??**
64. Forestiero et al. [724, 725, 728, 727, 726, 729] (2005–2008) *aims*: self-organization, QoS features, dynamic or adaptive behavior; *representation*: information distributed over the network; *optimization method*: ACO/ant agents

Broadcast and Multicast

General Networks or Theory

65. Christensen et al., see entry 40.
66. Zhu et al., see entry 42.
67. Comellas and Giménez [434] (1998) *aims*: synthesis, QoS features; *representation*: trees; *optimization method*: evolutionary algorithms, see also **??**

Computer Networks in General

68. Fei et al., see entry 48.
69. Grace, see entry 32.

Telecommunication Networks in General

70. Christensen et al., see entry 40.
71. Zhu et al., see entry 42.
72. Galiasso and Wainwright, see entry 56.

Other

73. Jaroš and Dvořák [1040] (2008) *aims*: synthesis, QoS features; *representation*: integer strings; *optimization methods*: Memetic Algorithms and estimation of distribution algorithms, see also **??**

Security and Intrusion Detection

Computer Networks in General

74. Heady et al. [911] (1990) *aim*: synthesis; *representation*: bit strings; *optimization method*: evolutionary algorithms, see also **??**
75. Song et al., see entry 31.
76. Song et al. [1919, 1920] (2003) *aim*: synthesis; *representation*: linear programs; *optimization method*: evolutionary algorithms, see also **??**
77. Liu et al. [1296] (2004) *aim*: synthesis; *optimization method*: evolutionary algorithms
78. Mukkamala et al. [1482] (2004) *aims*: synthesis, robustness; *representation*: linear programs; *optimization method*: evolutionary algorithms, see also **??**
79. Lu and Traore [1316] (2004) *aim*: synthesis; *representation*: integer strings plus genotype-phenotype mappings; *optimization method*: evolutionary algorithms, see also **??**
80. Folino et al. [710] (2005) *aim*: synthesis; *representations*: trees and linear programs; *optimization method*: evolutionary algorithms, see also **??**
81. Hansen et al. [887] (2007) *aim*: synthesis; *representation*: linear programs; *optimization method*: evolutionary algorithms, see also **??**

Wireless or Mobile Computer Networks

82. LaRoche and Zincir-Heywood [1257] (2005) *aim*: synthesis; *representation*: linear programs; *optimization method*: evolutionary algorithms, see also **??**

Agent Cooperation (non-ant)

General Networks or Theory

83. Werner and Dyer [2194] (1992) *aim*: synthesis; *representation*: integer strings plus genotype-phenotype mappings; *optimization method*: artificial life, see also **??**
84. Andre [54] (1995) *aim*: synthesis; *representation*: trees; *optimization method*: evolutionary algorithms
85. Qureshi [1687, 1686, 1688] (1996–2001) *aim*: synthesis; *representation*: trees; *optimization method*: evolutionary algorithms
86. Iba et al. [984, 987, 985, 986] (1996–1999) *aims*: synthesis, robustness; *representation*: trees; *optimization method*: evolutionary algorithms, see also **??**
87. Mackin and Tazaki, see entry 28.

Computer Networks in General

88. Nakano and Suda [1497, 1498, 1499] (2004–2007) *aims*: self-organization, QoS features, dynamic or adaptive behavior; *representations*: real vectors and information distributed over the network; *optimization method*: evolutionary algorithms, see also **??**
89. Zapf and Weise, see entry 61.

Telecommunication Networks in General

90. Schoonderwoerd et al., see entry 52.

Software Configuration

91. Grace, see entry 32.
92. Iles and Deugo, see entry 63.
93. Xi et al. [2268] (2004) *aim*: QoS features; *representation*: integer strings; *optimization methods*: local search and Simulated Annealing, see also **??**
94. Nakano and Suda, see entry 88.

Hardware Design and Configuration

Networks in General

95. Martin et al., see entry 25.

Wireless or Mobile Networks in General

96. Choo et al. [400] (2000) *aim*: synthesis; *representation*: bit strings; *optimization method*: evolutionary algorithms
97. Lohn et al. [1307] (2004) *aim*: synthesis; *representation*: real vectors; *optimization method*: evolutionary algorithms
98. Villegas et al. [2116] (2004) *aim*: synthesis; *optimization method*: evolutionary algorithms
99. Koza et al. [1212] (2005) *aim*: synthesis; *representation*: trees; *optimization method*: evolutionary algorithms
100. John and Ammann [1057, 1058] (2006) *aim*: synthesis; *representation*: bit strings; *optimization method*: evolutionary algorithms
101. Chattoraj and Roy [378] (2006) *aim*: synthesis; *representation*: bit strings; *optimization method*: evolutionary algorithms

Algorithm Synthesis

Computer Networks in General

102. Weise et al. [2179, 2184] (2007–2008) *aim*: synthesis; *representation*: integer strings; *optimization methods*: evolutionary algorithms and local search
103. Weise et al. [2181] (2007) *aim*: synthesis; *representation*: bit strings; *optimization method*: evolutionary algorithms

Wireless or Mobile Computer Networks

104. Weise and Geihs [2176, 2175, 2177] (2001–2006) *aims*: synthesis, robustness; *representation*: linear programs; *optimization method*: evolutionary algorithms
105. Weise et al. [2182, 2183] (2007) *aim*: synthesis; *representation*: linear programs; *optimization method*: evolutionary algorithms

23.2.4 Conclusions

In this study, we gave a short overview on the wide variety of applications of global optimization to distributed systems. For the last ten years, this has been one of the most active research areas in Evolutionary Computation, with many researchers steadily contributing new and enhanced approaches.

We not only provided a representative list and classification of publications, but also introduced many interesting approaches in a detailed way. Yet, we can only offer a small glimpse on the real amount of work available. The master's thesis of Kampstra [1087] is now already four years old and referenced over 400 papers. From the related work section of the papers that we have summarized we know that there should exist at least another 200 contributions not mentioned in his list or not yet published when it was compiled.

Practitioners in the area of networking or telecommunication tend to feel skeptical when it comes to the utilization of such eerie things like randomized or bio-inspired approaches for optimizing, managing, or controlling their systems. One argument against the use of metaheuristics is that the worse case results may be unpredictably bad although they may provide good solutions in average.

Nevertheless, certain problems (like the Terminal Assignment Problem, see **??**) are $\mathcal{NP}$-hard and therefore can only be solved efficiently with such approaches. This, of course, goes hand in hand with a certain trade-off in terms of optimality, for instance. In static design scenarios, the worst case situations in which an EA would create inferior solutions can be ruled out by checking its results before the actual deployment or realization.

In practice, additional application-specific constraints are often imposed on standard problems. The influence of these constraints on the problem hardness and the applicability of the well-known solutions is not always easy to comprehend. Thus, incorporating the constraints into a global optimization procedure tends to be much easier than customizing a problem-specific heuristic algorithm. Assume that we want to find fast routes in a network which are also robust against a certain fraction of failed links. If we have an EA with an objective function that measures the time a message travels in a fully functional network, it is intuitively clear that we can extend this approach by simply applying this function to a couple of scenarios with randomly created link failures, too. Creating a corresponding extension of Dijkstra's algorithm, however, is less straightforward.

Nature-inspired approaches have not only shown their efficiency in static optimization problems, but were proven to be especially robust in dynamic applications, too. This is particularly interesting in the looming age of networks of larger scale. Wireless networks [1707, 829, 1440], sensor networks [1012], wireless sensor networks [326]), Smart Home networks [899, 897], ubiquitous computing [850, 1218], and more require self-organization, efficient routing, optimal parameter settings, and power management. We are sure that nature and physics-inspired global optimization methods will provide viable answers to many of these questions which will become more and more eminent in the near future.

When condensing the essence of this summary down to a single sentence, "Evolutionary Computing in Telecommunications – A likely EC success story", the title of Kampstra's thesis, maybe fits best. However, we believe that the *likely* is no longer required, since many of the methods developed already reached engineering-level applicability.

24

Research Applications

Research applications differ from real-world application by the fact that they have not yet reached the maturity to be applied in the mainstream of their respective area. They initiate a process of improvement and refinement, until we obtain solutions that are on par or at least comparable with those obtained by the traditional methodologies. Such a process can, for instance, be observed when following the progress in the area of Genetic Programming via the book series of Koza [1196, 1195, 1210, 1212]. On the other hand, research applications differ from toy problems because they are not intended to be used as sole demonstration examples or benchmarks but are first steps into a new field of application.

The future of a research application is either to succeed and become a real-world application or to fail. In case of a failure, it may turn into a toy application where some certain features of global optimization methods like evolutionary algorithms can be tested.

24.1 Genetic Programming of Distributed Algorithms

24.1.1 Introduction

Distributed systems are one of the most vital components of our economy. While many internet technologies, protocols, and applications grew into maturity and have widely been researched, new forms of networks and distributed computing have emerged. Amongst them, we can find wireless networks [1707, 829, 1440], sensor networks [1012] (and wireless sensor networks [326]), Smart Home networks [899, 897], ubiquitous computing [850, 1218], and ideas like amorphous computing [5, 313]. These distributed systems introduce new requirements like self-adaptation to change in the environment (nodes may enter and leave the networks frequently) or change the priority of others (such as energy consumption).

It may be a bold statement to say that such new requirements ask for new programming paradigms and future will shows whether it holds or not. Nobody will, however, argue that developing applications for these new distributed systems is surely to become more complicated than in traditional networks. Hence, exploring the utility of new programming methodologies (and new representations for algorithms especially tailored to them) is a demand of the current situation.

The design of a distributed algorithm is basically the transformation of a specification of the behavior of a network on the global scale to a program that must be executed locally on each of the nodes of the network in order to achieve this behavior. Up to now, no general method for automating this process illustrated in Fig. 24.1.a has been developed and it is unlikely that this will change in near future.

The transformation of global system behavior to local rules is no process specific only to distributed algorithm design. Matter of fact, a widely studied example for are swarming behaviors in nature [1723, 2033]. These behaviors have evolved for millions of years. By allowing many individuals of a species to travel together in a configuration which has a

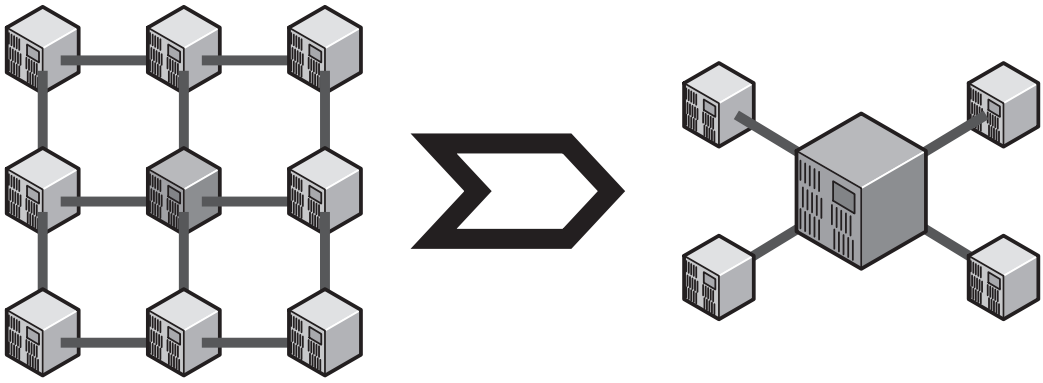


Fig. 24.1.a: Design of distributed algorithms.

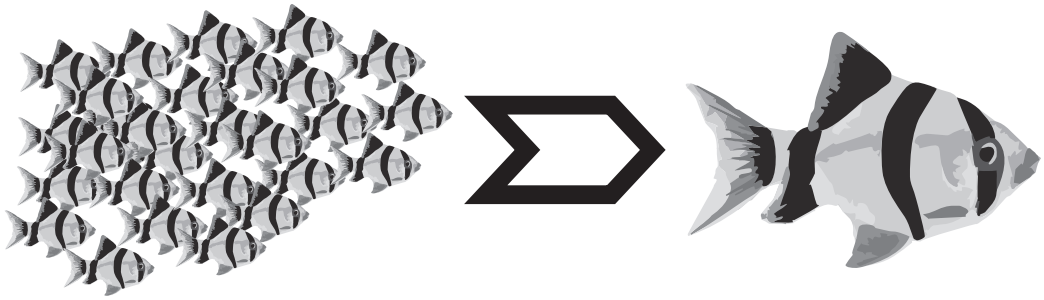


Fig. 24.1.b: Evolutionary behavior design.

Figure 24.1: Global → Local behavior transformations.

good volume/surface-ratio, they improve defense against predators, increase the chance of finding mating partners, enhance foraging success, improve hydro or aerodynamics and so on. Nature has evolved many efficient swarming behaviors, such as the shoaling of fish (depicted in Fig. 24.1.b), flocking of birds, herding of cows, and swarming of locusts.

Evolutionary algorithms copy the evolutionary process itself for solving complex optimization problems [99, 821] and Genetic Programmingis the family of EAs which can be used for deriving programs [1196]. Here we will utilize it for breeding distributed algorithms – in other words – for transforming descriptions of global behaviors to local algorithms.

These global descriptions are therefore encoded in objective functions, which rate "how close" the behavior of an evolved program x comes to the wanted one. In order to approximate its quality, we execute x on nodes represented by virtual machines in a whole simulated network. As in reality, many of these VMs run asynchronously at approximately the same speed, which may differ from VM to VM and cannot be assumed to be constant either. For different problems, different topologies are simulated.

We apply multi-objective Genetic Programming since it allows us to optimize the algorithms for different aspects during the evolution. While the functional objective functions perform the actual comparison of the observed behavior of the simulated network (running the evolved algorithms) with the desired global behavior, non-functional objective functions foster the economical use of resources, minimizing communication and program size, for instance.

24.1.2 Evolving Proactive Aggregation Protocols

In this section we discuss what proactive aggregation protocols are and how we can evolve them using a modified symbolic regression approach with Genetic Programming.

Aggregation Protocols

Definition 24.1 (Aggregate). In computer science, an aggregate function[1] $\alpha : \mathbb{R}^m \mapsto \mathbb{R}$ computes a single result $\alpha(\mathbf{x})$ from a set of input data $\mathbf{x}$. This result represents some features of the input like its arithmetic mean.

Other examples for aggregate functions are the variance and the number of points in the input data. In general, an aggregate[2] is a fusion of a (large) set of low-level data to one piece of high-level information. Aggregation operations in databases and knowledge bases [1284, 1595, 1152, 1309, 385, 551, 1904], be they local or distributed, for instance, have been an active research area in the past decades. Here, large datasets from different tables are combined to an aggregate by structured queries which need to be optimized for maximal performance.

With the arising interest in peer-to-peer applications (see Section 30.2.2) and sensor networks (discussed in Section 30.2.2), a whole new type of aggregation came into existence in the form of aggregation protocols. These protocols are a key functional building block by providing the processes in such distributed systems with access to global information including network size, average load, mean uptime, location and description of hotspots, and so on [2099, 1048]. Robust and adaptive applications often require this local knowledge of such properties of the whole. If, for example, the average concentration of some toxin (which is aggregated from the measurements of multiple sensors in a chemical laboratory) exceeds a certain limit, an alarm should be triggered.

In aggregation protocols, the data vector $\mathbf{x}$ is no longer locally available but its elements are spread all over the network. When computing the aggregate under these circumstances, we cannot just evaluate α. Instead, some form of data exchange must be performed by the nodes. This exchange can happen in two ways: either *reactive* or *proactive*. In a reactive aggregation protocol, one of the nodes in the network issues a query to all other nodes. Only this node receives the answer in form of the result (the aggregate) or the data needed to compute the result as illustrated in Fig. 24.2.a. A proactive aggregation protocol, as sketched in Fig. 24.2.b, on the other hand allows all nodes in the network to receive knowledge of the aggregate. This is achieved by repetitive data exchange amongst the nodes and iterative local refinement of the estimates of the wanted value. Notice that the trivial solution would be that all nodes send their information to all other nodes. Generally, this is avoided since it is not a viable approach and instead, the data is disseminated step by step as part of the estimates.

Gossip-Based Aggregation

Jelasity et al. [1048] propose a simple yet efficient type of proactive aggregation protocols [1123]. In their model, a network consists of many nodes in a dynamic topology where every node can potentially communicate with every other node. Errors in communication may occur, Byzantine faults not. The basic assumption of the protocol is that each node in the network holds one numerical value x. This value represents some information about the node or its environment, like, for example, the current work load. The task of the protocol is to provide all nodes in the network with an up-to-date estimate of the aggregate function $\alpha(\mathbf{x})$ of the vector of all values $\mathbf{x} = (x_p, x_q, \dots)^T$.

The nodes hold local states s (possibly containing x) which they can exchange via communication. Therefore, each nodes knows picks its communication partners with the getNeighbor() method.

The skeleton of the gossip-based aggregation protocol is specified in Algorithm 24.1 and consists of an active and a passive part. Once in each $\delta > 0$ time units, at a randomly picked time, the active thread of a node p selects a neighbor q. Both partners exchange their

[1] http://en.wikipedia.org/wiki/Aggregate_function [accessed 2007-07-03]
[2] http://en.wikipedia.org/wiki/Aggregate_data [accessed 2007-07-03]

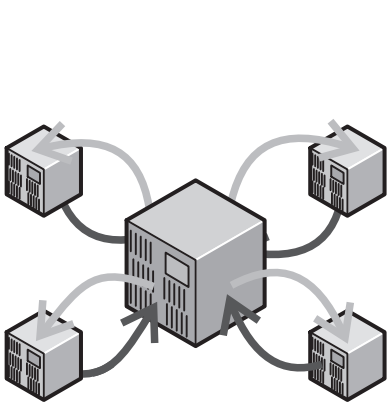


Fig. 24.2.a: reactive aggregation

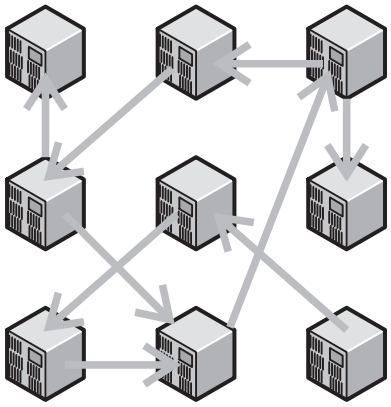


Fig. 24.2.b: proactive aggregation

Figure 24.2: The two basic forms of aggregation protocols.

information and update their states with the update method: p calls update(s_p, s_q) in its active thread and q calls update(s_q, s_p) in the passive thread. update is defined according to the aggregate that we want to be computed

Algorithm 24.1: gossipBasedAggregation()

Data: p: the node running the algorithm
Data: s_p: the local state of the node p
Data: s_q, s_r: states received as messages from the nodes q and r
Data: q, p, r: neighboring nodes in the network

```
1  begin
2      Subalgorithm activeThread
3      begin
4          while true do
5              do exactly once in every δ units at a randomly picked time:
6              q ⟵ getNeighbor()
7              sendTo(q, s_p)
8              s_q ⟵ receiveFrom(q)
9              s_p ⟵ update(s_p, s_q)
10     end
11     Subalgorithm passiveThread
12     begin
13         while true do
14             s_r ⟵ receiveAny()
15             sendTo(getSender(s_r), s_p)
16             s_p ⟵ update(s_p, s_r)
17     end
18 end
```

Example – Distributed Average

Assume that we have built a sensor network measuring the temperature as illustrated in Figure 24.3. Each of our sensor nodes is equipped with a little display visible to the public.

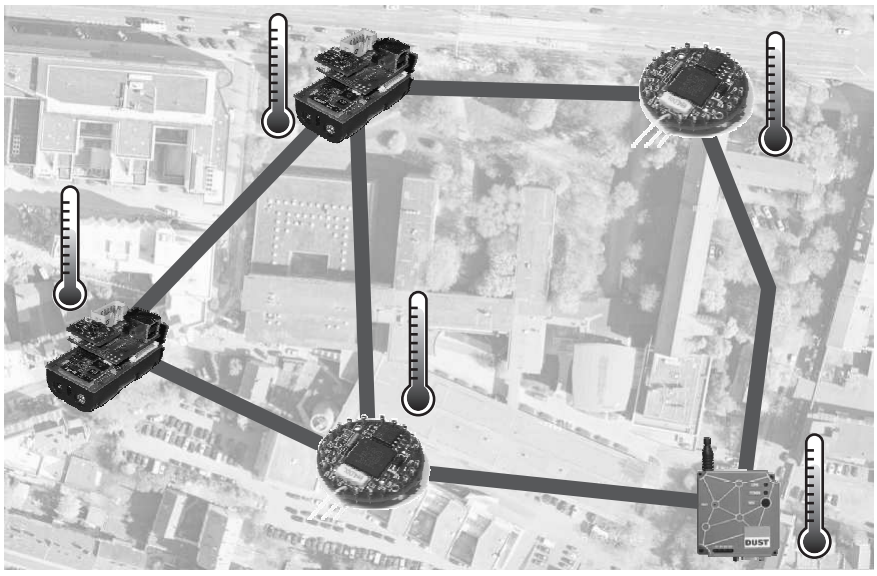


Figure 24.3: An example sensor network measuring the temperature.

The temperatures measured locally will fluctuate because of wind or light changes. Thus, the displays should not only show the temperature measured by the sensor node they are directly attached to, but also the average of all temperatures measured by all nodes. Then, the network needs to execute a distributed aggregation protocol in order to estimate that average.

If we therefore choose a gossip-based average protocol, each node will hold a state variable which contains its local estimation of the mean. The update function, henceforth receiving the local approximation and the estimate of another node, returns the mean of its inputs.

$$\text{update}_{avg}(s_p, s_q) = \frac{s_p + s_q}{2} \tag{24.1}$$

If two nodes p and q communicate with each other, the new value of s_p and s_q will be $s_p(t+1) = s_q(t+1) = 0.5 * (s_p(t) + s_q(t))$. The sum – and thus also the mean – of both states remains constant. Their variance, however, becomes 0 and so the overall variance in the network gradually decreases.

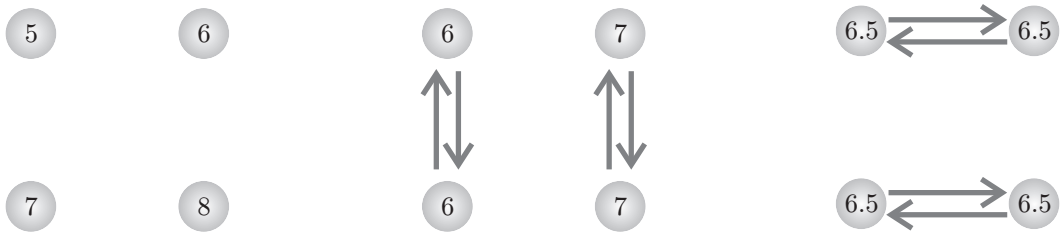


Fig. 24.4.a: initial state

Fig. 24.4.b: after step 1

Fig. 24.4.c: after step 2

Figure 24.4: An gossip-based aggregation of the average example.

In order to visualize how that type of protocol works, let us assume that we have a network of four nodes with the initial values $\mathbf{x} = (5, 6, 7, 8)^T$ as illustrated in Fig. 24.4.a.

The arithmetic mean of its elements is

$$\frac{5+6+7+8}{4} = \frac{13}{2} = 6.5 \tag{24.2}$$

The initial variance is

$$\frac{(5-6.5)^2+(6-6.5)^2+(7-6.5)^2+(8-6.5)^2}{4} = \frac{5}{4} \tag{24.3}$$

In the first step of the protocol, the nodes with the initial values 5 and 7 as well as the other two exchange data with each other and update their values to 6 and 7 respectively (see Fig. 24.4.b). Now the average of all estimates is still

$$\frac{6+6+7+7}{4} = 6.5 \tag{24.4}$$

but the variance has been reduced to

$$\frac{(6-6.5)^2+(6-6.5)^2+(7-6.5)^2+(7-6.5)^2}{4} = 1 \tag{24.5}$$

After the second protocol step, outlined in Fig. 24.4.c, all nodes estimate the mean with the correct value 6.5 (and thus, the variance is 0). The distributed average protocol is only one example of gossip-based aggregation. Others are:

1. **Minimum** and **Maximum**. The minimum and maximum of a value in the network can be computed by setting $\text{update}_{min}(s_p, s_q) = \min\{s_p, s_q\}$ and $\text{update}_{max}(s_p, s_q) = \max\{s_p, s_q\}$ respectively.
2. **Count**. The number of nodes in a network $\mathbf{N}$ can be computed using the average protocol: the initiator sets its state to 1 and all other nodes begin with 0. Then the average is computed is then $\frac{1+0+0+\ldots}{\text{numNodes}(\mathbf{N})} = \frac{1}{\text{numNodes}(\mathbf{N})}$ where $\text{numNodes}(\mathbf{N})$ is the number of nodes in $\mathbf{N}$. The nodes now just need to invert the computed value locally and obtain $\frac{1}{\frac{1}{\text{numNodes}(\mathbf{N})}} = \text{numNodes}(\mathbf{N})$.
3. **Sum**. The sum of all values in the network can be computed by estimating both, the mean value $\overline{x}$ and the number of nodes $\text{numNodes}(\mathbf{N})$ in the network $\mathbf{N}$ simultaneously and multiplying both with each other: $\text{numNodes}(\mathbf{N})\,\overline{x} = \sum x$.
4. **Variance**. As declared in Equation 28.61 on page 474, the variance of a data set is the difference of the mean of the squares of the values and the square of their means. Therefore, if we compute $\overline{x^2}$ and $\overline{x}$ by using the average protocol, we can subtract them $\text{var}(x) \approx \overline{x^2} - \overline{x}^2$ and, hence, obtain an estimation of the variance.

Further considerations are required if $\mathbf{x}$ is not constant but changes by and by. Both, peer-to-peer networks as well as sensor networks, have properties (discussed in Section 30.2.2) which are very challenging for distributed applications and lead to an inherent volatility of $\mathbf{x}$. According to Jelasity et al. [1048], a default approach to handle unstable data is to periodically restart the aggregation protocols. In our research, we were able to provide alternative aggregation protocols capable of dealing with dynamically changing data. This approach is discussed in Section 24.1.2 on page 414.

The Solution Approach: Genetic Programming

In order to derive certain aggregate functions automatically, we could modify the Genetic Programming approach for symbolic regression introduced in Section 23.1 on page 397 [2180, 2187]. Let $\alpha : \mathbb{R}^m \mapsto \mathbb{R}$ be the exact aggregate function. It works on a vector of the dimension m containing the data elements. $m \in \mathbb{N}$ is not a predetermined constant but depends on the network size, i. e., $m = \text{numNodes}(\mathbf{N})$ and α will return exact results for $m = 1, 2, 3, \ldots$. In Section 28.7.2 on page 503, we will show that the dimension m of the domain $\mathbb{R}^m$ of α

plays no role when approximating it with a maximum likelihood estimator. The theorems used there are again applied in symbolic regression (see Equation 23.6 on page 399), so the value of m does not affect the correctness of the approach. Deriving aggregation functions for distributed systems, however, exceeds the capabilities of normal symbolic regression. Each of the $m = \text{numNodes}(\mathbf{N})$ nodes in the network $\mathbf{N}$ holds exactly one element of the data vector. α cannot be computed directly anymore since it requires access to all data elements at once. Instead, each node has to execute local rules that define how data is exchanged and how an approximation of the aggregate value is calculated. How to find these rules automatically is the subject of our research here. There are three use cases for such an automated aggregation protocol generation:

1. We may already know a valid aggregation protocol but want to find an equivalent protocol which has advantages like faster convergence or robustness in terms of input volatility. This case is analogous to finding arithmetic identities in symbolic regression.
2. We do not know the aggregate function α nor the protocol but have a set of sample data vectors $\mathbf{x}_i$ (maybe differing in dimensionality) and corresponding aggregates y_i. Using Genetic Programming, we attempt to find an aggregation protocol that fits to this sample information.
3. The most probable use case is that we know how to compute the aggregate locally with a given function α but want to find a distributed protocol that does the same. We, for example, are well aware of how to compute the arithmetic mean of a data set $(x_1, x_2, \ldots, x_m)$ – we just divide the sum of the single data items by their number m. If these items, however, are distributed and not locally available, we cannot simply sum them up. The correct solution described in Section 24.1.2 on page 416 is that each node starts by approximating the mean with its locally known value. Now always two nodes inform each other about their estimates and set their new approximation to be that mean of the old and the received on. This way, the aggregate is approached by iteratively refining the estimations.
   The transformation of the local aggregate calculation rule α to the distributed one is not obvious. Instead of doing it by hand, we can just use the local rule to create sample data sets and then apply the approach of the second use case.

Network Model and Simulation

For gossip-based aggregation protocols, Jelasity et al. [1048] assume a topology where all nodes can potentially communicate with each other. In this fully connected overlay network, communication can be regarded as fault-free. Taking a look at the basic algorithm scheme of such protocols introduced as Algorithm 24.1 on page 416, we see that the data exchange happens once every δ time units at a randomly picked point in time. Even though being asynchronous in reality, it will definitely happen in this time span. That is, we may simplify the model to a synchronous network model where all communication happens simultaneously.

Another aspect of communication is how the nodes select their partners for the data exchange. It is a simple fact that the protocol can only converge to the correct value if each node has, maybe over multiple hops and calculations, been able to receive information from all other nodes. Imagine a network $\mathbf{N}$ consisting of $m = \text{numNodes}(\mathbf{N}) = 4$ nodes p, q, r, and t. If the communication partners are always (p, q) and (r, t), the data dissemination is insufficient since p will never be able to incorporate the knowledge of the states of r and t. On the other hand, one data exchange between q and r will allow the protocol to work since p would later on indirectly receive the required information from q.

Besides this basic fact, Jelasity et al. [1048] have shown that different forms of pair selection influence the convergence speed of the protocol. Correct protocols will always converge if complete data dissemination is guaranteed. Knowing that, we should choose a partner selection method that leads to fast convergence because we then can safe protocol steps in

the evaluation process. The pair building should be deterministic, because randomized selection schemes lead to slow convergence [1048], and, more importantly, will produce different outcomes in each test and make comparing the different evolved protocols complicated (as discussed in Section 1.3.4 on page 55). Therefore, choosing a deterministic selection scheme seems to be the best approach. Perfect matching according to Jelasity et al. [1048] means that each node is present in exactly one pair per protocol cycle, i. e., always takes part in the data exchange. If different pairs are selected in each cycle, the convergence speed will increase. It can further be increased by selecting (different) pairs in a way that disseminates the data fastest.

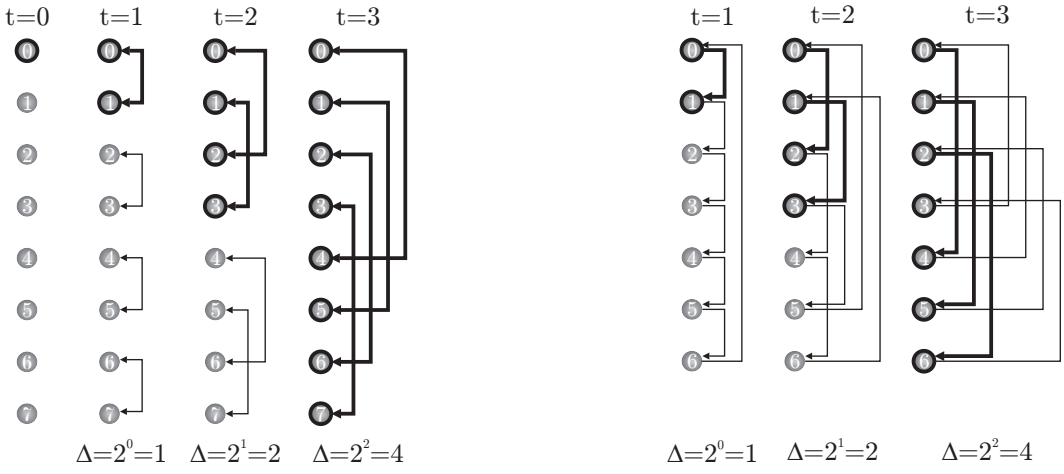


Fig. 24.5.a: pair-based

Fig. 24.5.b: general

Figure 24.5: Optimal data dissemination strategies.

From these ideas, we can derive a deterministic pair selection mechanism with best-case convergence. Therefore, we first need to set the number of nodes in the simulated network $\text{numNodes}(\mathbf{N}) = m = 2^d$ as a power of two. In each protocol step t with $t = 1, 2, \ldots$, we compute a value $\Delta = 2^{t \bmod d}$. Then we build pairs in the form $(i, i + \Delta)$, where i and $i + \Delta$ are the indices identifying the nodes. This setup equals a butterfly graph and is optimal, as you can see in Fig. 24.5.a. The data from node 0 (marked with a thick border) spreads in the first step to node 1. In the second step, it reaches node 2 directly and node 3 indirectly through node 1. Remember, if the average protocol would use this pair selection scheme, node 3 would compute its new estimate at step 2 since

$$s_3(t=2) = \frac{s_3(t=1) + s_1(t=1)}{2} + \frac{\frac{s_3(t=0)+s_2(t=0)}{2} + \frac{s_0(t=0)+s_1(t=0)}{2}}{2} \tag{24.6}$$

In the third protocol step, the remaining four nodes receive knowledge of the information from node 0 and the data is disseminated over the complete network. Now the cycle would start over again and node 0 would communicate with node 1.

This pair selection method is bounded to networks of the size $m = 2^d$. We can generalize this approach by breaking up the strict pair-communication limitation. Therefore, we set $d = \lceil \log_2 m \rceil$ while still leaving $\Delta = 2^{t \bmod d}$ and define that a node i sends its data to the node $(i + \Delta) \bmod m$ for all i as illustrated in Fig. 24.5.b. This general communication rule abandons the strict pair-based data exchange but leaves any other feature of the aggregation protocols, like the working of the update method, untouched. We should again visualize that this rule is only defined so we can construct simulations where the protocols need as few

as possible steps to converge to the correct value in order to spare us computation time. Another important aspect also becomes obvious here: The time that an aggregation protocol needs to converge will always depend on the number of nodes in the (simulated) network.

Node Model and Simulation

As important as modeling the network is the model of the nodes it consists of. In Figure 24.6, we illustrate an abstraction especially suitable for fast simulation of aggregation protocols. A node p executing a gossip-based aggregation protocol receives input in form of the locally

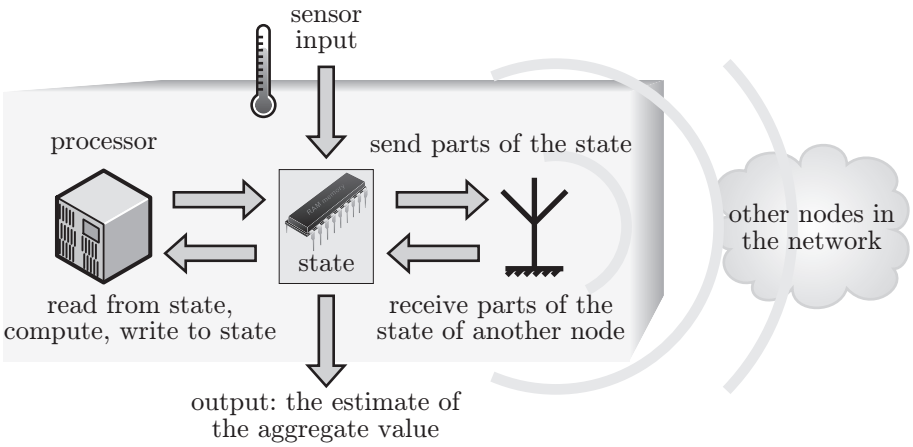


Figure 24.6: The model of a node capable to execute a proactive aggregation protocol.

known value (for example a sensor reading) and also in form of messages containing data from other nodes in the network. The output of p consists of the local approximation of the aggregate value and the information sent to its partners in the network. The computation is done by a processor which updates the local state by executing the update function. The local state s_p of p can most generally be represented as vector $\mathbf{s}_p \in \mathbb{R}^n$ of the dimension n, where n is the number of memory cells available on a node.

Like Jelasity et al. [1048], we until now have considered the states to be scalars. Generalizing them to vectors allows us to specify or evolve more complicated protocols. The state vector contains the approximation of the aggregate value at the position $i : 1 \leq i \leq n$. If the state only consists of a single number, the messages between two nodes will always include this single number and hence, the complete state.

A state vector not only serves as a container for the aggregate, but also as memory capable of accumulating information. It is probably unnecessary or unwanted to exchange the complete state during the communication. Therefore, we specify an index list e containing the indices of the elements to be sent and a list r with the indices of the elements that shall receive the values of the incoming messages. For a proper communication between the nodes, the length of e and r must be equal and each index must occur at most once in e and also at most once in r. Whenever a node p receives a message from node q, the following assignment will be done, with $\mathbf{s}_{p[i]}$ being the i^{th}th component of the vector:

$$\mathbf{s}_{p[r_j]} \longleftarrow \mathbf{s}_{q[e_j]} \; \forall j = 0 \ldots \text{len}(r) - 1 \tag{24.7}$$

In the original form of gossip-based aggregation protocols, the state is initialized with a static input value which is stepwise refined to approximate the aggregate value [1048]. In our

model, this restriction is no longer required. We specify an index I pointing at the element of the state vector that will receive the input. This allows us to grow protocols for static and for volatile input data. In the latter case, the inputs are refreshed in each protocol step. A node p would then perform

$$\mathbf{s}_{p[I]}(t) \longleftarrow \text{getInput}(p,t) \tag{24.8}$$

The function $\text{getInput}(p,t)$ returns the input value of node p at time step t. With this definition, the state vectors $\mathbf{s}_p$ become time-dependent, written as $\mathbf{s}_p(t)$. Finally, update is now designed as a map $\mathbb{R}^n \mapsto \mathbb{R}^n$ to return the new state vector.

$$\mathbf{s}_p(t+1) = \text{update}(\mathbf{s}_p(t)) \tag{24.9}$$

In the network simulation, we can put the state vectors of all nodes together to a single $n \times m$ matrix $S(t)$. The column k of this matrix contains the state vector $\mathbf{s}_k$ of the node k.

$$S(t) = (\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_m) \tag{24.10}$$

$$S_{[j,k]} = \mathbf{s}_k[j] \tag{24.11}$$

This notation is used in Algorithm 24.2 and Algorithm 24.3. In Algorithm 24.2 we specify how the model definitions just discussed can be used to build a network simulation for gossip-based, proactive aggregation protocols. Here we also apply the general optimal communication scheme explained in Section 24.1.2. In the practical realization, we can spare creating a new matrix $S(t)$ in each time step t by initial using two matrices S_1, S_2 which we simple swap in each turn.

Evaluation and Objective Values

The models described before are the basis of the evaluation of the aggregation protocols that we breed. In general, there are two functional features that we want to develop in the artificial evolution:

1. We want to grow aggregation protocols where the deviation between the local estimates and the global aggregate is as small as possible, ideally 0.
2. This deviation can surely not be 0 after the first iteration at $t = 1$, because the nodes do not know all data at that time. However, the way how received data is incorporated into the local state of a node can very well influence the speed of convergence to the wanted value. Therefore, we want to find protocols that converge as quickly as possible.

In all use cases discussed in Section 24.1.2, we either already know the correct aggregation values y_i or the local aggregate function $\alpha : \mathbb{R}^m \mapsto \mathbb{R}$ that calculates them from data vectors of the length m. The objective is to find a distributed protocol that computes the same aggregates in a network where the data vector is distributed over m nodes. In our model, the estimates of the aggregate value can be found at the positions $S_{[O,\cdot]}\star \equiv \mathbf{s}_{k[O]}\ \forall k \in [1 \ldots m]$ in the state matrix or the state vectors respectively.

The deviation $\varepsilon(k,t)$ of the local approximation of a node k from the correct aggregate value $y(t)$ at a point in time t denotes its estimation error.

$$y(t) = \alpha\Big(\big(\text{getInput}(1,t),..,\text{getInput}(m,t)\big)^T\Big) \tag{24.12}$$

$$\varepsilon(k,t) = y(t) - S_{[O,k]}(t) = y(t) - \mathbf{s}_{k[O]} \tag{24.13}$$

We have already argued that the mean square error is an appropriate quality function for symbolic regression (see Equation 23.6). Analogously, the mean of the squares of the errors ε over all simulated time steps and all simulated nodes is a good criterion for the utility of an

Algorithm 24.2: simulateNetwork(m, T)

Input: m: the number of nodes in the simulation
Input: T: the maximum number of simulation steps
Input: [implicit] update: the update function
Input: [implicit] I: the index for the input values
Input: [implicit] O: the index for the output values
Input: [implicit] e: the index list for the send values
Input: [implicit] r: the index list for the receive values
Data: d: communication step base according to Fig. 24.5.b on page 420
Data: k: a node index
Data: $S(t)$: the simulation state matrix at time step t
Data: Δ: the communication partner offset
Data: p: the communication partner node

```
1  begin
2      d ⟵ ⌈log_2 m⌉
3      S(0) ⟵ createMatrix(n × m)
       // initialize with local values
4      S(0)[·,k] ⟵ getInputk0
5      t ⟵ 1
6      while t ≤ T do
7          S(t) ⟵ copyMatrix(S(t − 1))
8          Δ ⟵ 2^(t mod d)
           // perform communication according to Fig. 24.5.b on page 420
9          k ⟵ 1
10         while k ≤ m do
11             p ⟵ (k + Δ) mod m
12             foreach j ∈ [1..len(r)] do  S(t)[r_j,p] ⟵ S(t − 1)[e_j,k]
13             k ⟵ k + 1
           // set (possible) new input values and perform update
14         k ⟵ 1
15         while k ≤ m do
16             S(t)[I,k] ⟵ getInput(k, t)
17             S(t)[i_j,k] ⟵ update(S(t)[·,k])
               // ≡ s_k(t) ⟵ update(s_k(t))
18             k ⟵ k + 1
19         t ⟵ t + 1
20 end
```

aggregation protocol. It is even related to both functional aspects subject to optimization: The larger it is, the greater is the deviation of the estimates from the correct value. If the convergence speed of the protocol is low, these deviations will become smaller more slowly by time. Hence, the mean square error will also be higher. For any evolved update function u we define[3]:

$$f_1(u, e, r) = \left. \frac{1}{T * m} \sum_{t=1}^{T} \sum_{k=1}^{m} \varepsilon(k, t)^2 \right| u, e, r \tag{24.14}$$

This rather mathematical definition is realized indirectly in Algorithm 24.3, which returns the value of f_1 for an evolved update method u. It also applies the fast, convergence-friendly communication scheme discussed in Section 24.1.2. Its realization in the Distributed Genetic Programming Framework [2177] software allows us to evaluate even complex distributed protocols in very short time: A protocol can be tested on 16 nodes for 300 protocol steps less than 5 milliseconds on a normal, 3GHz off-the-shelf PC.

[3] where $\cdot|u, e, r$ means "passing u, e, r as input to Algorithm 24.3"

Algorithm 24.3: $f_1(u, e, r) \longleftarrow$ evaluateAggregationProtocol(u, m, T)

Input: u: the evolved protocol update function to be evaluated
Input: m: the number of nodes in the simulation
Input: T: the maximum number of simulation steps
Input: [implicit] update: the update function
Input: [implicit] I: the index for the input values
Input: [implicit] O: the index for the output values
Input: [implicit] e: the index list for the send values
Input: [implicit] r: the index list for the receive values
Data: d: communication step base according to Fig. 24.5.b on page 420
Data: k: a node index
Data: $S(t)$: the simulation state matrix at time step t
Data: Δ: the communication partner offset
Data: p: the communication partner node
Data: res: the variable accumulating the square errors
Output: $f_1(u, e, r)$: the sum of all square errors (deviations from the correct aggregate) over all time steps

```
1  begin
2      d ⟵ ⌈log_2 m⌉
3      S(0) ⟵ createMatrix(n × m)
       // initialize with local values
4      S(0)[·,k] ⟵ getInputk0
5      t ⟵ 1
6      while t ≤ T do
7          S(t) ⟵ copyMatrix(S(t − 1))
8          ...
           // perform communication according to Fig. 24.5.b on page 420
9          ...
           // set (possible) new input values and perform update
10         k ⟵ 1
11         while k ≤ m do
12             S(t)[I,k] ⟵ getInput(k, t)
               // u is the evolved update-function and thus, used here
13             S(t)[·,k] ⟵ u(S(t)[·,k])
14             res ⟵ res + (y(t) − S(t)[O,k])^2
               // ≡ res ⟵ res + (α(i(t)) − S(t)[o,k])^2
15             k ⟵ k + 1
16         t ⟵ t + 1
17     return res
18 end
```

Input Data

In Algorithm 24.3 we use sample α values in order to determine the errors ε. In two of our initial use cases, we need to create these values before the evaluation process, either with an existing protocol or with a known aggregate function α. Here we will focus on the latter case.

If transforming a local aggregate function α to a distributed aggregation protocol, we need to create sample data vectors for the getInput(k, t)-method. Here we can differentiate between *static* and *dynamic* input data: for static input data, we just need to create the samples for $t = 0$ since getInput$(k, 0) =$ getInput$(k, 1) = \ldots agGetInputbkT \; \forall k \in [1..n]$. If we have dynamic inputs on the other hand, we need to ensure that at least some elements of the input vectors $\mathbf{x}(t) =$ getInput $\cdot\, t$ will differ, i. e., $\exists t_1, t_2 : \mathbf{x}(t_1) \neq \mathbf{x}(t_2)$. If this difference is too large, an aggregation protocol cannot converge. It should be noted that it would

be wrong to assume that we can measure this difference in terms of the sample data $\mathbf{x}$ – restrictions like $0.9 < \left|\frac{\mathbf{x}[i](t)}{\mathbf{x}[i](t+1)}\right| < 1.\overline{1}$ are useless, because their impact on the value of α is unknown. Instead, we must limit the variations in terms of the aggregation results, like

$$0.9 < \left|\frac{\alpha(\mathbf{x}(t))}{\alpha(\mathbf{x}(t))}\right| < 1.\overline{1} \tag{24.15}$$

In both, the static and the dynamic case, we need to create multiple input datasets, distinguished by adding a dataset index to $\mathbf{x}(t)$: $\mathbf{x}(1,t), \mathbf{x}(2,t), \ldots, \mathbf{x}(l,t)$. Only if $\alpha(\mathbf{x}(1,t)) \neq \alpha(\mathbf{x}(2,t)) \neq \cdots \neq \alpha\mathbf{x}(l,t)$ we can assure that the result are not just overfitted protocols that simple always return one single learned value: the exact α of the sample data. The single $\mathbf{x}(i,t)$ should differ in magnitude, sign, and distribution since this will lead to large differences in the α-values:

$$\left(\left|\frac{\alpha(\mathbf{x}(i,t))}{\alpha(\mathbf{x}(j,t))}\right| \ll 1\right) \vee \left(\left|\frac{\alpha(\mathbf{x}(i,t))}{\alpha(\mathbf{x}(j,t))}\right| \gg 1\right) \; \forall\, i \neq j \tag{24.16}$$

We use z such data sets to perform z runs of Algorithm 24.3 and compute the true value of f_1 as arithmetic mean of the single results.

$$f_1(u,e,r) = \frac{1}{z}\sum_{i=1}^{z} underCondition f_f(u,e,r)\mathbf{x}_i \tag{24.17}$$

Of course, for each protocol that we evaluate we will use the same sample data sets because otherwise the results would not be comparable. It should be noted that overfitting can furthermore be prevented by changing the sample vectors in each generation. In this experiment series, generating the test data was too time consuming so we did not apply this measure.

Volatile Input Data

The specification of getInput(k,t) which returns the input value of node k at time $t \in [0..T]$ allows us to evolve aggregation protocols for static and such for volatile input. Traditional aggregation protocols are only able to deal with constant inputs [1048]. These protocols have good convergence properties, as illustrated in Fig. 24.7.a. They always converge to the correct results but will simple ignore changes in the input data (see Fig. 24.7.b).

They would need to be restarted in a real application from time to time in order to provide up-to-date approximations of the aggregate. This approach is good if the input values in the real application that we evolve the protocols for change slowly. If these inputs are volatile, the estimations of these protocols become more and more imprecise. The fact that an aggregation protocol needs a certain number of cycles to converge is an issue especially in larger or mobile sensor networks. One way to solve this problem is to increase the data rate of the network accordingly and to restart the protocols more often. If this is not feasible, because of, for example, energy restrictions in a low-power sensor network application prohibit increasing the network traffic, dynamic aggregation protocols may help.

They represent a sliding average of the approximated parameter and are able to cope with changing input data. In each protocol step, they will incorporate their old state, the received information, and the current input data into the calculations. A dynamic distributed average protocol like the one illustrated in Figure 24.8 is a weighted sum of the old estimate, the received estimate, and the current value. The weights in the sum can be determined by the Genetic Programming process according to the speed with which the inputs change. In order to determine this speed for the simulations, a few real sample measurements would suffice to produce customized protocols for each application situation. However, the incorporation of the current input value is also the drawback of such an approach, since it cannot fully converge to the correct result anymore.

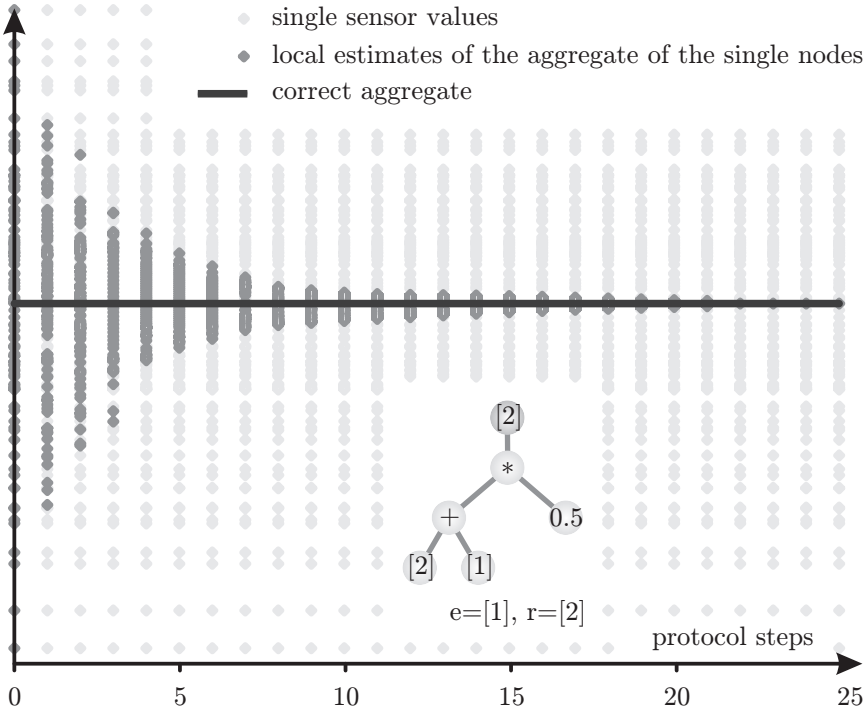


Fig. 24.7.a: with constant inputs

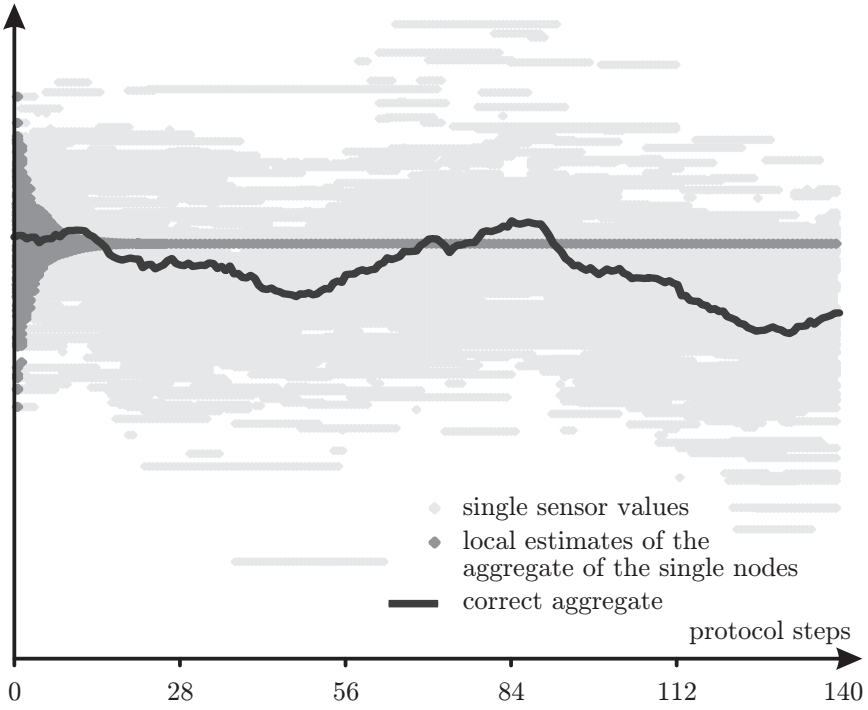


Fig. 24.7.b: with volatile inputs

Figure 24.7: The behavior of the distributed average protocol in different scenarios.

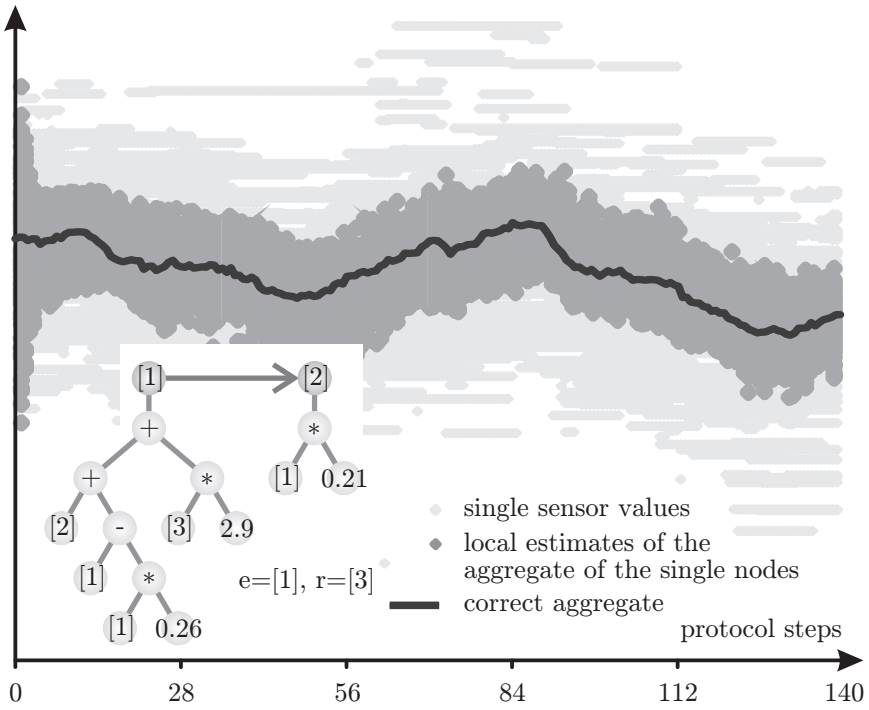


Figure 24.8: A dynamic aggregation protocol for the distributed average.

Phenotypic Representations of Aggregation Protocols

We have to find a proper representation for gossip-based aggregation protocols. Such a protocol consists of two parts: the evolved update function and a specification of the properties of the state vector – the variables I, O, r, and e.

Representation for the update Function

The function update as defined in the context of our basic model for aggregation protocols receives the state vectors $\mathbf{s}_k(t) \in \mathbb{R}^n$ of time step t as input. It returns the new state vectors $\mathbf{s}_k(t+1) \in \mathbb{R}^n$ of time step $t+1$. This function is indeed an algorithm by itself which can be represented as a list of tuples $l = (\ldots, (u_j, v_j), \ldots)$ of mathematical expressions u_j and vector element indices v_j. This list l is processed sequentially for $j = 1, 2, \ldots, \text{len}(l)$. In each step j, the result of the expression u_j is computed and assigned to the ${v_j}^{th}$ element of the old state vector $\mathbf{s}(t-1)$. In the simplest case, l will have the length $\text{len}(l) = 1$. One example for this is the well-known distributed average protocol illustrated in Figure 24.9: In the single formula, the first element of $\mathbf{s}_{[1]}(t)$, [1], is assigned to $0.5 * ([1] + [2])$ which is the average of its old value and the received information. Here, the value of the first element is send to the partner and the received message is stored in the second element, i. e., $r = [2], e = [1]$. The terminal set of the expressions now does not contain the simple variable x anymore but all elements of the state vectors. Finally, after all formulas in the list have been computed and their return values are assigned to the corresponding memory cells, the modified old state vector $\mathbf{s}_k(t)$ becomes the new one $\mathbf{s}k(t+1)$. Fig. 24.9.b shows a more complicated protocol where *update* consists of $\text{len}(l) = 4$ formulas $((u_1, 1), (u_2, 2), (u_3, 3), (u_4, 2))$. We will not elaborate deeper on these examples but just note that both are valid results of Genetic Programming – a more elaborate discussion of them can be found in Section 24.1.2 on page 430, Section 24.1.2 on page 431, and [2187, 2180].

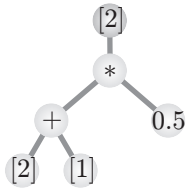


Fig. 24.9.a: distributed average

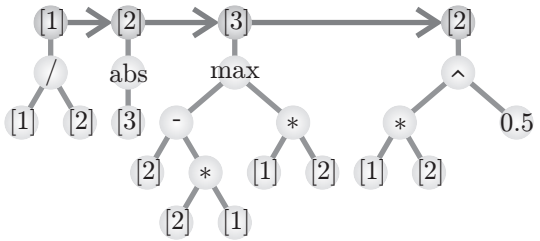


Fig. 24.9.b: square root of the distributed average

Figure 24.9: Some examples for the formula series part of aggregation protocols.

The important point is that we are able to provide a notation for the first part of the aggregation protocol specification that is compatible to normal symbolic regression and which thus can be evolved using standard operators.

Besides this sequence of formulas computed repetitively in a cycle, we also need an additional sequence that is executed only once, in the initialization phase. It is needed for some other protocols than the distributed minimum, maximum, and average, which cannot assume the approximation of the estimate to be the current input value. Here, another sequence of instructions is needed which transforms the input value into an estimate which then can be exchanged with other nodes and used as basis for subsequence calculations. This additional sequence is evolved and treated exactly in the same way as the set of formulas used inside the protocol cycle.

Experiments have shown that it is useful though to initialize all state elements in the first time step with the input values. Therefore, both Algorithm 24.2 and Algorithm 24.3, initially perform $S(0)_{[\cdot,k]} \longleftarrow \text{getInput}(k, 0)$ instead of $S(0)_{[i,k]} \longleftarrow \text{getInput}(k, 0)$. In all other time steps, only $S(t)_{[i,k]}$ is updated.

Straightforwardly, we can specify a non-functional objective function f_2 that returns the number of expressions in both sets and, hence, puts pressure into the direction of small protocols with less computational costs.

Representation for I, O, e, and r

Like the update function, the parameters of the data exchange, r and e, become subject to evolution. I and O are only single indices; we can assume them to be fixed as $I = 1$ and $O = 2$. Allowing them to be changed will only result in populations of many incompatible protocols. Although we could do the same with e and r, there is a very good reason to make them variable. If e and r are built during the evolutionary process, different protocols with different message lengths ($\text{len}(e_1) \neq \text{len}(e_2)$) can emerge. Hence, we can introduce a non-functional objective function f_3 that puts pressure into the direction of minimal message lengths. The results of Genetic Programming will thus be optimal not only in accuracy of the results but only in terms of communication costs.

For the lists e and r there are three possible representations. We can use either a bit string of the fixed length $2n$ which contains two bits for each element of $\mathbf{s}$: the first bit determines if the value of the element should be sent, the second bit denotes if an incoming element should be stored there. String genomes of a fixed length are explained in detail in Section 3.4 on page 147. By doing so, we implicitly define some restrictions on the message structure since we need to define an order on the elements inside. If $n = 4$, a bit string `01011010` will be translated into $e = (3, 4)$ and $r = (1, 2)$. It is not possible to obtain something like $e = (3, 4)$ and $r = (2, 1)$.

The second encoding scheme is to use two variable-length integer strings which represent e and r directly. Such genomes are introduced in Section 3.5 on page 149. Now the latter

case becomes possible. If the lengths of the two strings differ, for example for reproduction reasons, the length of the shorter one is used solely.

The third approach would be to, again, evolve one single string z. This string is composed of pairs $z = ((e_1, r_1), (e_2, r_2), \ldots, (r_l, r_l))$. The second and the third approach are somewhat equivalent,

In principle, all three methods are valid and correct since the impossibility of some message structures in the first method does not necessarily imply that certain protocol functionality cannot evolve. The standard reproduction operators for string genomes, be they of fixed or variable length, can be applied.

When we closely examine our abstract protocol representation, we will see that it will work with epidemic [1047] or SPIN-based [914] communication too, although we developed it for a gossip-based communication model.

Reproduction Operators

As already pointed out when elaborating on the representation schemes for the two parts of the aggregation protocols, well-known reproduction operators can be reused here.

1. The formulas in the protocol obey strictly a tree form, where the root always has two child nodes, the formula sequences for the protocol cycle and the initialization, which, in turn, may have arbitrarily many children: the formulas themselves. A formula is a tree node which has stored one number which identifies the vector element its results will be written to. It has exactly one child node, the mathematical expression which is a tree of other expressions. We elaborate on tree-shaped genomes in Section 4.3 on page 162.
2. The communication behavior is described as either one fixed-length bit string or two variable-length integer strings.

New protocols are created by first building a new formula tree and then combining it with one (or two, according to the applied coding scheme) newly created string chromosomes. We define the mutation operation as follows: If an aggregation protocol is mutated, with 80% probability its formula tree is modified and with 20% probability its message pattern. When performing a recombination operation, a new protocol is constructed by recombining the formula tree as well as the message definition of both parents with the default means.

Results from Experiments

In Table 24.1, we list the configuration of the Genetic Programming algorithm applied to breed aggregation protocols.

Parameter	Short	Description
Problem Space	$\mathbb{X}$	The space of aggregation programs. (see Section 24.1.2)
Objective Functions	$\boldsymbol{F}$	$F = \{f_1, f_2, f_3\}$, see Algorithm 24.3, Section 24.1.2, Section 24.1.2
Search Space	$\mathbb{G}$	(basically) identical with the problem space, i. e., $\mathbb{G} = \mathbb{X}$.
Search Operations	$\boldsymbol{Op}$	mutation and crossover (see Section 24.1.2)
GPM	**gpm**	not needed
Optimization Algorithm	***alg***	elitist Genetic Programming
Comparison Operator	***cm***	$\text{cmp}_{F,agg}$ (see Equation 24.18)

Population Size	***ps***	$ps = 4096$
Maximum Archive Size	***as***	The size of the archive with the best known individuals was limited to $as = 64$. (see Definition 2.4)
Steady-State	***ss***	The algorithms were generational (not steady-state) ($ss = 0$). (see Section 2.1.6)
Fitness Assignment Algorithm	***fa***	For fitness assignment in the evolutionary algorithm, Pareto ranking was used. (see Section 2.3.3)
Selection Algorithm	***sel***	A binary ($k = 2$) tournament selection was applied. (see Section 2.4.4)
Convergence Prevention	***cp***	No additional means for convergence prevention were used, i. e., $cp = 0$. (see Section 2.4.8)
Number of Training Cases	***tc***	The number of training cases used for evaluating the objective functions were $tc = 22$, where each run is granted 28 cycles in the static and 300 cycles in the dynamic case.
ct	***ct***	The training cases were fixed, i. e., $ct = 0$.

Table 24.1: The settings of the Aggregation-Genetic Programming experiments.

In the simulations, 16 virtual machines were running, each holding a state vector $\mathbf{s}$ with five elements. From all experiments, those with a tiered prevalence comparison performed best and, hence, will be discussed in this section. Tiered prevalence comparison is similar to a Pareto optimization which is performed level-wise. When comparing two solution candidates x_1 and x_2, initially, the objective values of the first objective function f_1 are considered only. If one of the solution candidates has here a better value than the other, it wins. If both values are equal, we compare the second objective values in the same way, and so on. The comparator function defined in Equation 24.18 gives correctness (f_1) precedence to protocol size (f_2). Its result indicates which of the two individuals won – a negative number denotes the victory of x_1, a positive one that x_2 is better. The tiered structure of $\text{cmp}_{F,agg}$ leads to optimal sets with few members that most often (but not always) have equal objective values and only differ in their phenotypes.

$$\text{cmp}_{F,agg}(x_1, x_2) = \begin{cases} -1 \text{ if } (f_1(x_1) < f_1(x_2)) \vee \\ \quad ((f_1(x_1) = f_1(x_2)) \wedge (f_2(x_1) < f_2(x_2))) \vee \\ \quad ((f_1(x_1) = f_1(x_2)) \wedge (f_2(x_1) = f_2(x_2)) \wedge \\ \quad\quad (f_3(x_1) < f_3(x_2))) \\ 1 \text{ if } (f_1(x_2) < f_1(x_1)) \vee \\ \quad ((f_1(x_2) = f_1(x_1)) \wedge (f_2(x_2) < f_2(x_1))) \vee \\ \quad ((f_1(x_2) = f_1(x_1)) \wedge (f_2(x_2) = f_2(x_1)) \wedge \\ \quad\quad (f_3(x_2) < f_3(x_1))) \\ 0 \text{ otherwise} \end{cases} \tag{24.18}$$

We do not need more than five memory cells in our experiments. The message size was normally one or two in all test series and if it was larger, it converged quickly to a minimum. The objective function f_3 that minimizes it thus shows no interesting behavior. It can be assumed that it will have equal characteristics like f_2 in larger problems.

Average – static

With this configuration, protocols for simple aggregates like minimum, maximum, and average can be obtained in just a few generation steps. We have used the distributed average protocol which computes $\alpha_{avg} = \overline{x}$ in many of the previous examples, for instance, in Section 24.1.2 on page 416, Section 24.1.2 on page 425, and in Fig. 24.9.a. The evolution of a static version such an algorithm is illustrated in Figure 24.10. The graphic shows how

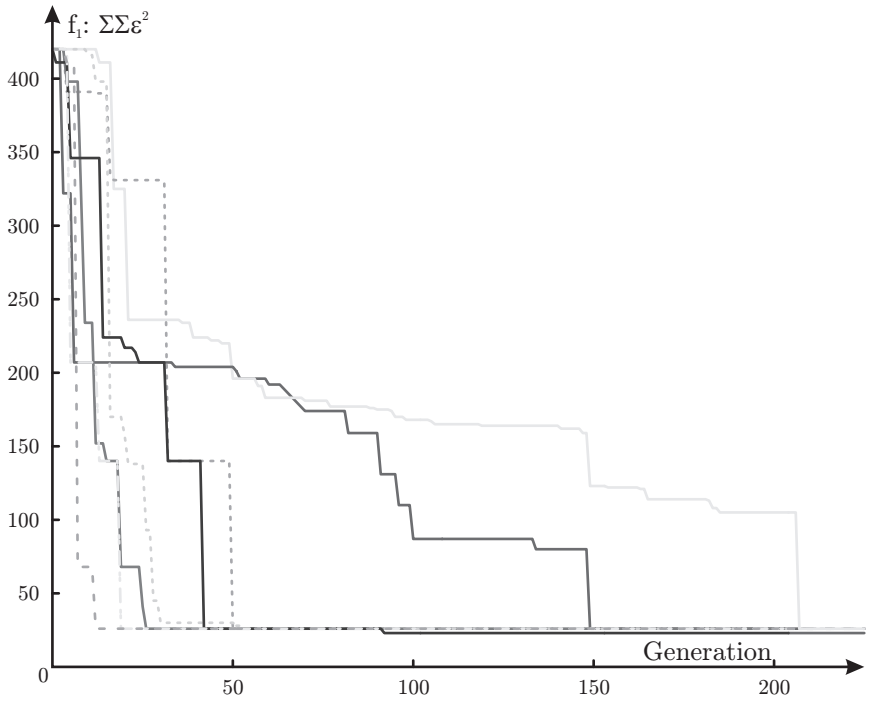


Figure 24.10: The evolutionary progress of the static *average* protocol.

the objective values of the first objective function (the mean square error sum) improve with the generations in twelve independent runs of the evolutionary algorithm. All runs did converge to the optimal solution previously discussed, most of them very quickly in less than 50 generations.

Figure 24.11 reveals the inter-relation between the first and second objective function for two randomly picked runs. Most often, when the accurateness of the (best known) protocols increases, the number of formula expressions also rises. These peaks in f_2 are always followed by a recession caused by stepwise improvement of the protocol efficiency by eliminating unnecessary expressions. This phenomenon is rooted in the tiered comparison that we chose: A larger but more precise protocol will always beat a smaller, less accurate one. If two protocols have equal precision, the smaller one will prevail.

Root-Of-Average – static

In our past research, we used the evolution of the *root-of-average* protocol as benchmark problem [2180]. Here, a distributed average protocol for the aggregate function α_{ra} is to be evolved:

$$\alpha_{ra}(\mathbf{x}) = \sqrt{|\overline{x}|} \; \forall x \in \mathbf{x} \tag{24.19}$$

One result of these experiments has already been sketched in Fig. 24.9.b. Figure 24.12 is a plot of eleven independent evolution runs. It also shows a solution found after only 84 generations in the quickest experiment. The values of the first objective function f_1, denoting the mean square error, improve so quickly in all runs at the beginning that a logarithmic scale is needed to display them properly. This contrasts with the simple average protocol evolution where the measured fitness is approximately proportional to the number of generations. The reason is the underlying aggregate function which is more complicated and thus, harder to approximate. Therefore, the initial errors are much higher and even small changes in the protocols can lead to large gains in accurateness.

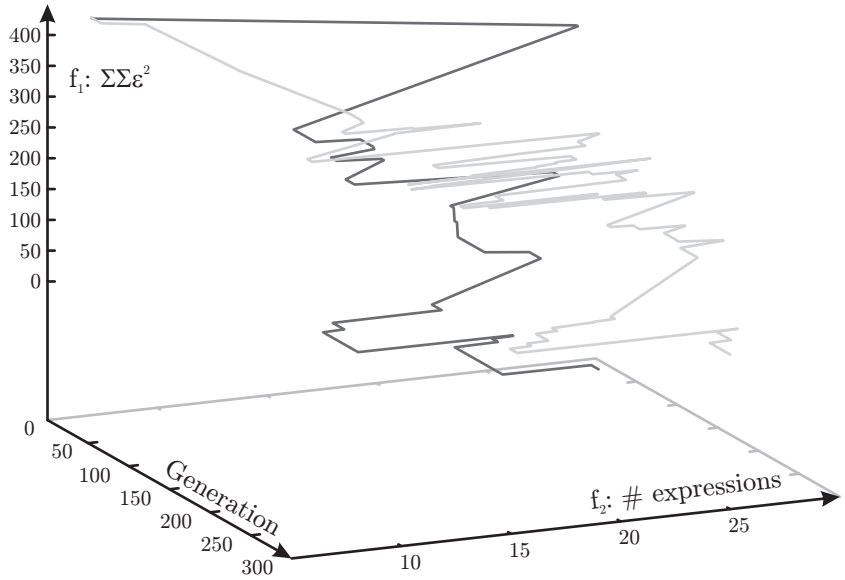


Figure 24.11: The relation of f_1 and f_2 in the static *average* protocol.

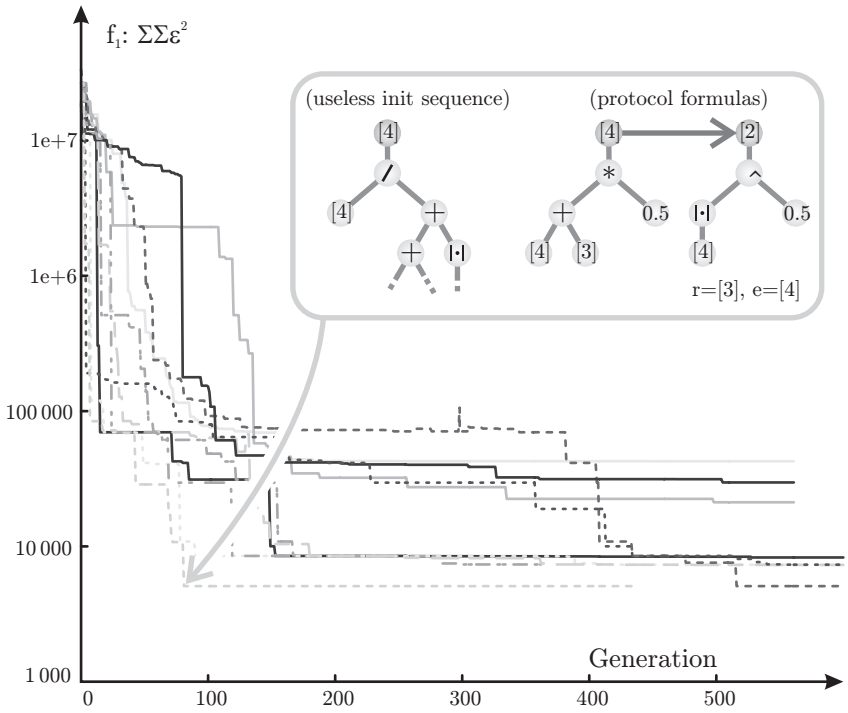


Figure 24.12: The evolutionary progress and one grown solution of the static *root-of-average* protocol.

The example solution contains a useless initialization sequence. In the experiments, it paradoxically did not vanish during the later course of the evolution although the secondary (non-functional) objective function f_2 puts pressure into the direction of smaller protocols. For the inter-relation between the first and second objective function, the same observations can be made than in the average protocol. Improvements in f_1 often cause an increase in f_2 which is followed by an almost immediate decrease, as pictured in Figure 24.13 for the 84-generation solution.

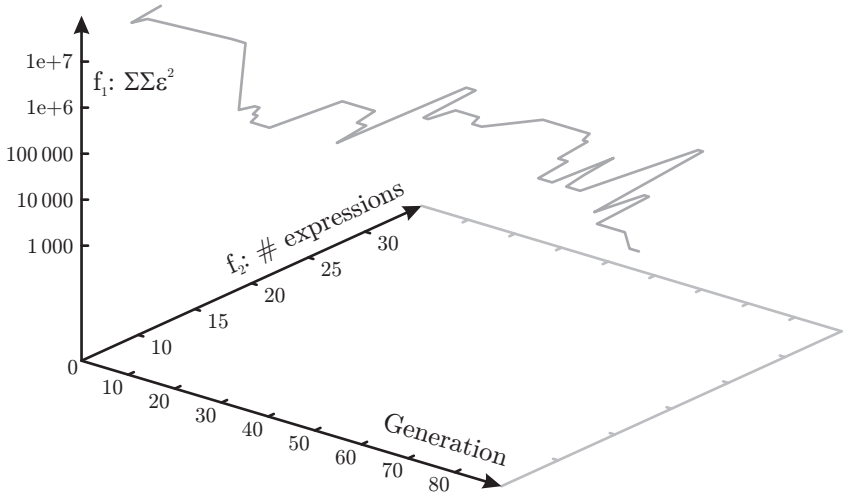


Figure 24.13: The relation of f_1 and f_2 in the static *root-of-average* protocol.

Average – dynamic

After verifying our approach for conventional aggregation protocols with static input data, it is time to test it with dynamically changing inputs. This may turn out be a useful application and is more interesting, since creating protocols for this scenario by hand is more complicated.

So we first repeat the "average" experiment for two different scenarios with volatile input data. The first one is depicted with solid lines in Figure 24.14. Here, the *true* values of the aggregate $\alpha(\mathbf{x}(t))$ can vary in each protocol step by 1% and in one simulation by 50% in total. In the second scenario, denoted by dashed lines, these volatility measures are increased to 3% and 70% respectively. The different settings have a clear impact on the results of the error functions – the more unsteady the protocol inputs, the higher will f_1 be, as Figure 24.14 clearly illustrates. The evolved solution exhibits very simple behavior: In each protocol step, a node first computes the average of its currently known value and the new sensor input. Then, it sets the new estimate to the average of this value and the value received from its partner node. Each node sends its current sensor value. This robust basic scheme seems to work fine in a volatile environment. The course of the evolutionary process itself has not changed significantly. Also the interactions between of f_1 and f_2 stay the same, as shown in Figure 24.15.

Root-Of-Average – dynamic

Now we repeat the second experiment, evolving a protocol that computes the square root of the average, with dynamic input data. Here we follow the same approach as for the dynamic average protocol: Tests are run with the same two volatility settings as in Section 24.1.2.

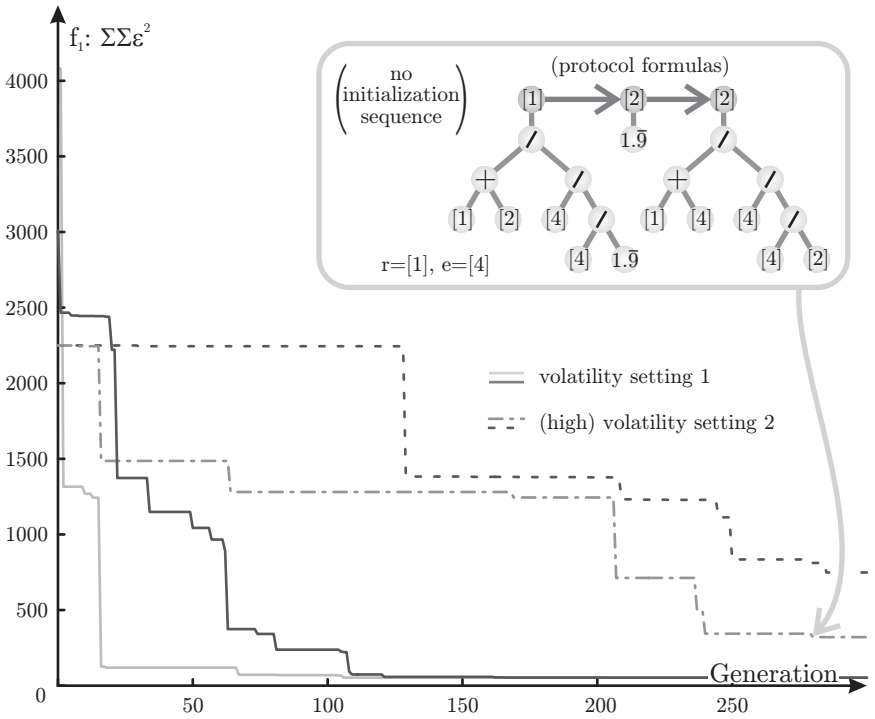


Figure 24.14: The evolutionary progress of the dynamic *average* protocol.

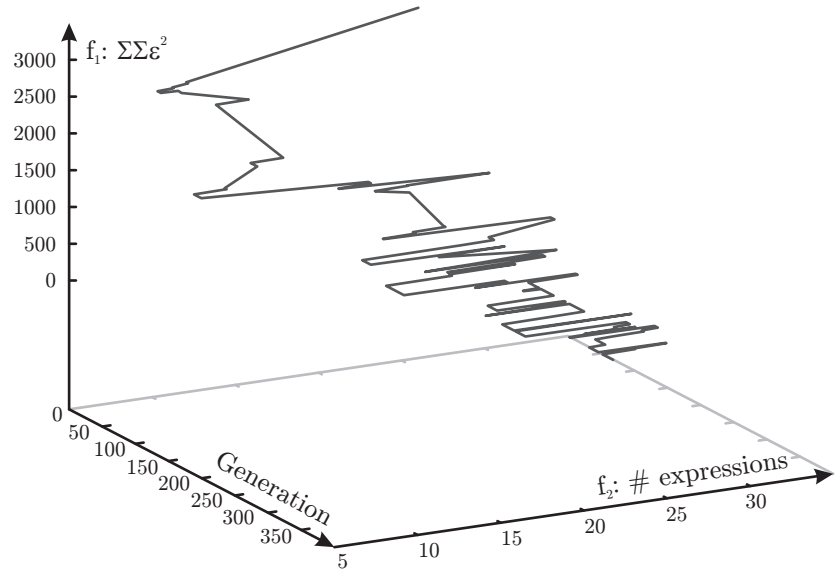


Figure 24.15: The relation of f_1 and f_2 in the dynamic *average* protocol.

Figure 24.16 shows how f_1 changes by time. Like in Figure 24.12, we have to use a logarithmic scaling for f_1 to illustrate it properly. For the tests with the slower changing data (solid lines), an intermediate solution is included because the final results were too complicated to be sketched here. The evolutions with the highly dynamic input dataset however did not yield functional aggregation protocols. From this we can follow that there is a threshold of volatility from which on Genetic Programming is no longer able to breed stable formulas.

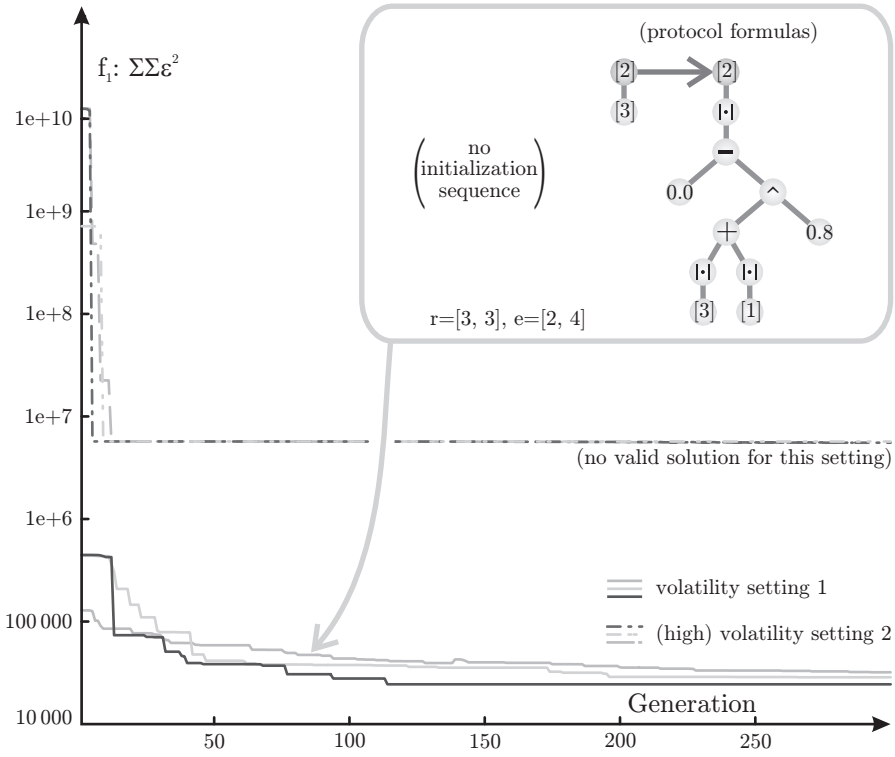


Figure 24.16: The evolutionary progress and one grown solution of the dynamic *root-of-average* protocol.

The relation of f_1 and f_2, outlined in Figure 24.17, complies with our expectations. In every experiment run, increasing f_1 is usually coupled to a deterioration of f_2 which means that the protocol formulas become larger and include more sub-expressions. This is followed by a recreation span where the formulas are reduced in size. After a phase of rest, where the new protocol supposable spreads throughout the population, the cycle starts all over again until the end of the evolution.

Conclusions

In this chapter, we have illustrated how Genetic Programming can be utilized for the automated synthesis of aggregation protocols. The transition to the evolution of protocols for dynamically changing input data is a step towards a new direction. Especially in applications like large-scale sensor networks, it is very hard for a software engineer to decide which protocol configuration is best. With our evolutionary approach, different solutions could be evolved for different volatility settings which can then be selected by the network according to the current situation.

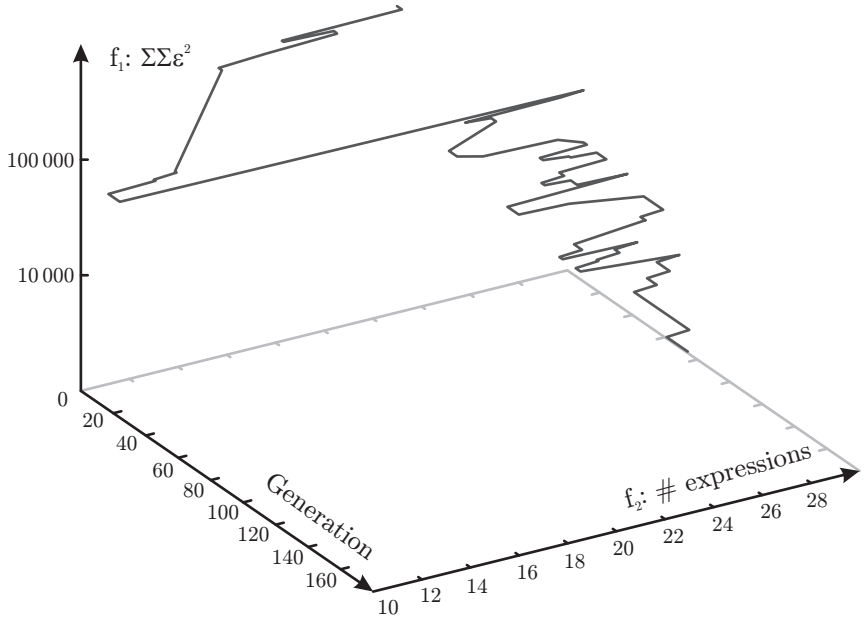


Figure 24.17: The relation of f_1 and f_2 in the dynamic *root-of-average* protocol.

Part III

Sigoa – Implementation in Java

25

Introduction

Today, there exist many different optimization frameworks. Some of them are dedicated to special purposes like spacecraft design [755] or trading systems [2302]. Others provide multi-purpose functionality like GALib [2140], Evolutionary Objects (EO) [1114, 1336] or the Java evolutionary computation library (ECJ) [1327].

In this part of the book, we want to introduce a new approach in global optimization software, called Sigoa, the *Simple Interface for Global Optimization Algorithms*[1]. Based on this library, we want to demonstrate how the optimization algorithms discussed in the previous chapters can be implemented.

We decided to use Java [837, 838, 317] as programming language and runtime system for this project since it is a very common, object-oriented, and platform independent. You can find more information on Java technology either directly at the website http://java.sun.com/ [accessed 2007-07-03] or in books like

1. *Javabuch* by Krüger [1217], the best German Java learning resource in my opinion, is online available for download at http://www.javabuch.de/ [accessed 2007-07-03].
2. For the English reader, *Thinking in Java* by Eckel [618] would be more appropriate – its free, third edition is online available at http://www.mindview.net/Books/TIJ/ [accessed 2007-07-03].
3. As same as interesting are the O'Reilly books *Java in a Nutshell* [687] and *Java Examples in a nutshell* [686] by Flanagan, and *Learning Java* by Niemeyer and Knudsen [1534].
4. *Java ist auch eine Insel* by [2071] – another good resource written in German, is also online available at http://www.galileocomputing.de/openbook/javainsel6/ [accessed 2007-07-03].

The source code of the binaries and the source files of the software described in this book part can be found online at http://www.sigoa.org/. It is not only open-source, but licensed very liberally under the LGPL (see appendix Chapter B on page 581) which allows for the integration of Sigoa into all kinds of systems, from educational to commercial, without any restrictions. Sigoa has features that aim to support optimizing complicated types of individuals which require time-consuming simulation and evaluation procedures.

Genetic Programming of real distributed algorithms is one example for such problems. In order to determine such an algorithm's fitness, you need to simulate the algorithm[2]. The evolution progresses step by step, so at first, we will not have any algorithm that works properly for a specified problem. Some of the solution candidates whatsoever will be able to perform some of the *sub-tasks* of the problem, or will maybe solve it partly. Since they may work on some of the inputs while failing to process other inputs correctly, a single simulation run will not be sufficient. We rather execute the algorithms multiple times and then use the minimum, median, average, or maximum objective values encountered. In the case of growing

[1] Sigoa http://www.sigoa.org/

[2] See Section 4.10 on page 219 for further discussions.

distributed algorithms, it is again not sufficient to simulate one processor. Instead, we need to simulate a network of many processors in order to determine the objective values[3]. Hence, it is simple to imagine that such a process may take some time. There are many other examples of optimization problems that involve complicated and time-consuming processes or simulations.

A framework capable of partially dealing with such aspects in an elegant manner has already been developed by the author in the past, see Weise [2175], Weise and Geihs [2176, 2177]. With the Sigoa approach, we use our former experiences to create a software package that has a higher performance and is way more versatile: One of the key features of Sigoa is the separation of specification from implementation, which allows heavyweight implementations as required for the evolution of the distributed algorithms as well as creating lightweight optimization algorithms which do not need simulations at all – like numerical minimization or such and such. This clear division not only allows for implementing all the optimization algorithms introduced in the previous parts but is good basis for including new, experimental methods that may have not been discussed yet.

Before starting with the specification of the Sigoa approach, we performed a detailed study on different global optimization methods and evolutionary algorithms. Additionally, we used the lessons learned from designing the DGPF system to write down the following major requirements:

25.1 Requirements Analysis

25.1.1 Multi-Objectivity

Almost all real-world problems involve contradicting objectives. A distributed algorithm evolved should, for example, be efficient and yet simple. It should consume not much memory and involve as little as possible communication between different processors but on the other hand should ensure proper functionality and be robust against lost or erroneous messages. The first requirement of an optimization framework is thus multi-objectivity.

25.1.2 Separation of Specification and Implementation

It should easily be possible to adapt the optimization framework to other problems or problem domains. The ability to replace the solution candidate representation forms is therefore necessary. Furthermore, the API must allow the implementation of all optimization algorithms discussed in the previous chapters in an easy and elegant manner. It should further be modular, since most of the optimization algorithms also consist of different sub-algorithms, as we have seen for example in Chapter 2 on page 95.

From this requirement we deduce that the software architecture used for the whole framework should be component based. Each component should communicate with the others only through clearly specified interfaces. This way, each module will be exchangeable and may be even represented by proxies or such and such, granting a maximum of extensibility. If we define a general interface for selection, we could modify the SPEA-algorithm (see **??** on page ??) which originally uses tournament selection to use another selection algorithm.

Hence, we will define Java-interfaces for all parts of optimization algorithms such as fitness assignment or clustering methods used for pruning the optimal sets. By doing so, we reach a separation of the specification from the implementation. For all interfaces we will provide a reference implementation which can easily be exchanged, allowing for different levels of complexity in the realizations.

[3] In Section 24.1.2 on page 414 you can find good example for this issue.

25.1.3 Separation of Concerns

An optimization system consists not only of the optimization algorithms themselves. It needs interfaces to simulators. If it is distributed, there must be a communication subsystem. Even if the optimization system is not distributed, we will most likely make use of parallelism since the processors inside of nowadays off-the-shelf PCs already offer supportive hyper-threading or dual-core technology [570, 1891]. If Sigoa is utilized by multiple other software systems which transfer optimization tasks to it, security issues arise. These aspects are orthogonal to the mathematical task of optimizing and should therefore be specified at different places and clearly be separated from the pure algorithms. Best practice commands to already consider such aspects in the earliest software design phase of every project and thus, also in the Sigoa library.

25.1.4 Support for Pluggable Simulations and Introspection

In most real-world scenarios, simulations are needed to evaluate the objective values of the solution candidates. If we use the framework for multiple problem domains, we will need to exchange these simulations or even want to rely on external modules. In some cases, the value of an objective function is an aggregate of everything what happened during the simulation. Therefore, they need a clear insight into what is going. Since we separate the objective functions from the simulations by clearly specified interfaces (as discussed in Section 25.1.3), these interfaces need to provide this required functionality of introspection.

In the use case of evolving a distributed algorithm, we can visualize the combination with the separation of concerns and introspective simulations: Besides working correctly, a distributed algorithm should use as few messages as possible or at least has stable demands considering the bandwidth on the communication channel. We therefore could write an objective function which inspects the number of messages underway in the simulation and computes a long-term average and variance. The simulation itself then does not need to be aware of that; it simple has to offer the functionality of counting the messages currently in transmission. The catch is that we can now replace the objective function by another one that maybe puts the pressure a little bit differently, leading to better results, without modifying the simulation. On the other hand, we can also use different simulation models – for example one where transmission errors can occur and one where this is not the case – without touching the objective function.

25.1.5 Distribution utilities

As already said, there are many applications where the simulations are very complicated and therefore, our architecture should allow us to distribute the arising workload to a network of many computers. The optimization process then can run significantly faster because many optimization techniques (especially evolutionary algorithms) are very suitable for parallel and distributed execution as discussed in Chapter 18 on page 299.

25.2 Architecture

We want to design the Sigoa optimization system based on these requirements. In this book part, we have assigned different chapters to the classes of different components of Sigoa and their sub-algorithms. By specifying interfaces for all aspects of optimization and implementing them elsewhere, the versatility to exchange all components is granted, so customized optimizers can be built to obtain the best results for different problem domains. Furthermore, interfaces allow us to implement components in different levels of detail: there may be applications where the evaluation of objective functions involves massive simulations

(like genetic programming) and applications, where the simple evaluation of mathematical functions enough (like numerical minimizing). In the latter case, using a system that provides extended support for simulations may result in performance degeneration since a lot of useless work is performed. If the mechanism that computes the objective values could be exchanged, an optimal approach can be used in each case.

As result from these considerations, we divide the Sigoa architecture in `org.sigoa` into two main packages: `org.sigoa.spec` contains the specifications and `org.sigoa.refimpl` a reference implementation. Figure 25.1 illustrates this top-level package hierarchy.

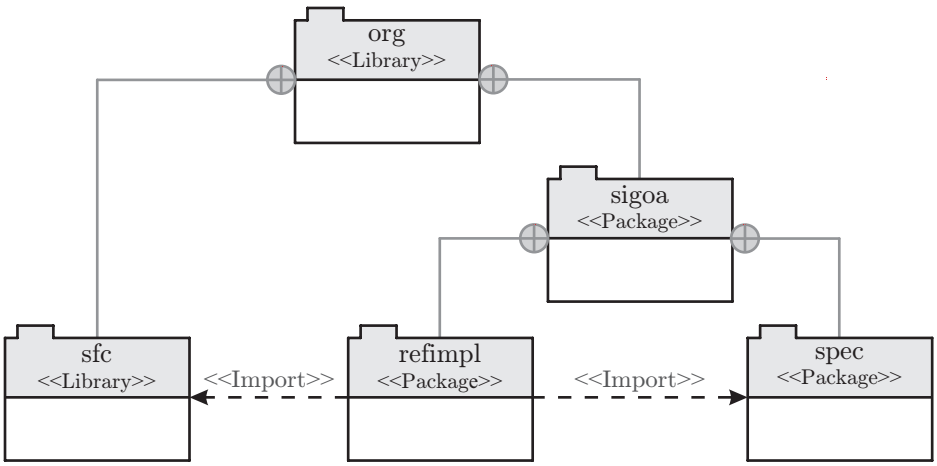


Figure 25.1: The top-level packages of the Sigoa optimization system.

The specification of the functionality of Sigoa is given by interfaces and a few basic utility classes only. It is independent from any library or other software system and does not require prerequisites. The interfaces can therefore also be implemented as wrappers that bridge to other, existing optimizing systems. Most of these specification interfaces inherit from `java.io.Serializable` and hence can be serialized using the Java Object Serialization mechanisms[4]. This way, we provide the foundation for creating snapshots of a running optimization algorithm which would allows for starting, stopping, restarting, and migrating them.

The reference implementation uses an additional software package called Sfc, the *Java Software Foundation Classes* – a LGPL-licensed open-source library available under the same URL as Sigoa that provides some useful classes for tasks that are needed in many applications like enhanced IO, XML support, extended and more efficient implementations of the Java Collection Framework[5]-interfaces and so on. This utility collection is not directly linked to optimization algorithms but provides valuable services that ease the implementation of the Sigoa components.

The package hierarchy of the reference implementation is identical to the one of the specifications. The package `org.sigoa.spec.gp.reproduction`, for example, contains the definition of mutation and crossover operations whereas the package `org.sigoa.refimpl.gp.reproduction` contains the reference implementation of these operations.

25.3 Subsystems

As illustrated in Figure 25.2, the Sigoa framework is constituted by nine subsystems:

[4] `http://java.sun.com/javase/6/docs/technotes/guides/serialization/` [accessed 2007-07-03]
[5] `http://java.sun.com/javase/6/docs/technotes/guides/collections/` [accessed 2007-07-03]

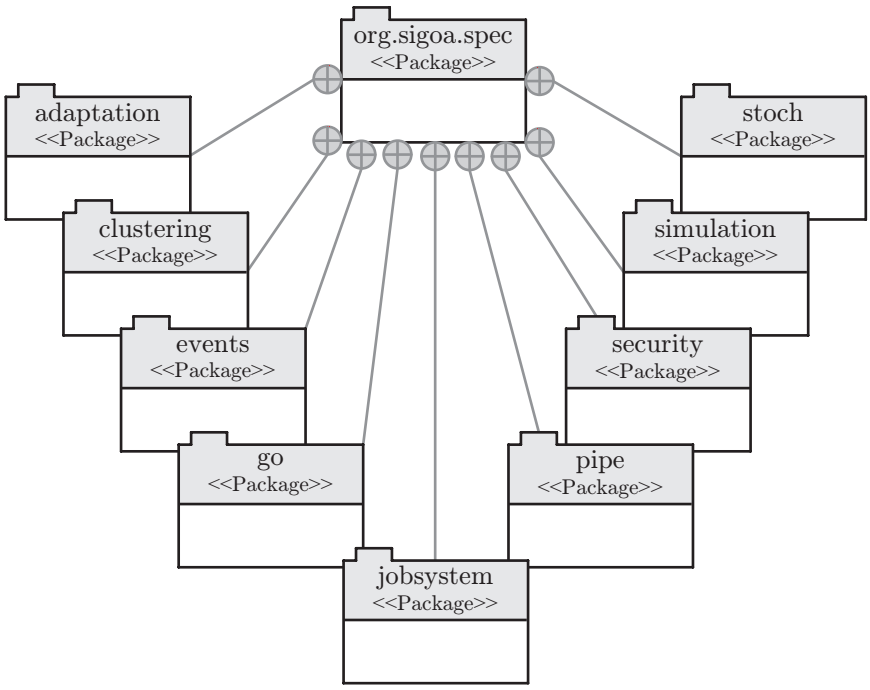


Figure 25.2: The subsystem specification of the optimization framework.

1. The `adaptation` package contains mechanisms that allow components to adapt themselves to a given situation based on rules. This can be used for example by optimization algorithms in order to adjust their parameters. A very simple application is the termination criterion[6]: a rule could be defined that terminates an algorithm after a given time.
2. In the `clustering`-package, interfaces for clustering-algorithms (as defined in Chapter 29 on page 535) are specified. Clustering algorithms can be used to reduce a large set of solution candidates to a smaller one without loss of generality.
3. One way for optimization algorithms to report their status and statistics to the outside world would be via events. As already said, we do not treat the optimization process as a mere mathematical procedure – it will always be part of some application. As such, not only the final results are interesting but also status messages and statistic evaluations of its progress. The `events` package defines interfaces for events that can be generated and may contain such information.
4. The largest subsystem is the `go` package, where all components and sub-algorithms for global optimization are specified. Here you can find the interface specifications that cover the all the algorithmic and mathematical functionality of global optimization algorithms.
5. In the `jobsystem` package, we place the specification of the means to *run* optimization algorithms. An optimizer may be parallelized or run sequentially and therefore may use multiple threads. The algorithm itself should be separated from the parallelism issues. Applying the definitions of the `jobsystem` package, optimizers may divide their work into parallelizable pieces which can be executed as *jobs*. Such jobs are then handled by the job system, which decides if they should be run in different threads or performed sequentially. This way, it also possible to manage multiple optimization algorithms in parallel and to specify which one will be assigned to how many processors. The implementations of the

[6] see Section 1.3.4 on page 54

job system specifications could also perform accounting and performance monitoring of the work load.

6. The concept of pipes defined in the `pipe` package is a very mighty approach for realizing optimization. It does not only allow separating the different components of an optimizer completely – totally new components, like statistic monitors can also easily be inserted into a system with minimum changes.
7. The job system enables Sigoa to handle multiple optimization requests at once. Since it is a plain component interface, these requests may come from anywhere, maybe even from a web service interface built on top of it. It must somehow be ensured that such requests do not interfere or even perform harmful or otherwise malicious actions. Therefore, a security concept is mandatory. In the `security` package we specify simple interfaces that build on the Java Security Technology[7].
8. The behavior of solution candidates is often simulated in order to determine their objective values. The `simulation` package provides interfaces that specify how simulations can be accessed, made available, and are managed.
9. Stochastic evaluations are a useful tool for optimization algorithms. As already said, the application using the Sigoa system may regularly need information about the progress, which normally can only be given in form of some sort of statistical evaluation. This data may also be used by the optimization algorithms themselves or by adaptation rules. Furthermore, most the global optimization methods discussed here are randomized algorithms. They thus need random number generators as introduced in Section 28.9 on page 526.

[7] `http://java.sun.com/javase/6/docs/technotes/guides/security` [accessed 2007-07-03]

26

Examples

But before we are going into detail about the different packages and utilities of the Sigoa software, we will present some application examples. These give a straightforward insight into the usage and customization of the Sigoa components which most probably is good enough to apply them to other problems. A more specific discussion of the Sigoa packages following after this chapter then rounds up the view on this novel optimization system.

26.1 The 2007 DATA-MINING-CUP

As an application example for genetic algorithms, the 2007 Data-Mining-Cup Contest, has been introduced in Section 22.1.2 on page 374. We strongly recommend reading this section first. We there have discussed the basic principles behind the challenge and the structure of one possible solution to it. Here we will only show how these ideas can be realized easily using the Sigoa framework.

The objective of the contest is to classify a set of 50 000 data vectors containing 20 features (from which only 17 are relevant) each into one of the three groups **A**, **B**, and **N**. In order to build classifiers that do so, another 50 000 datasets with already known classifications are available as training data. Thus, let us start by representing the three possible classification results using a simple Java `enum` like in Listing 26.1.

Our approach in Section 22.1.2 was to solve the task using a modified version of Learning Classifier Systems C. For the contest, a function $P(C)$ denoting the profit that could be gained with a classifier C was already defined (see Equation 22.1). Thus, we simple strip the LCSs from their *learning* capability and directly maximize the profit directly.

26.1.1 The Phenotype

The problem space $\mathbb{X}$ was thus composed of mere classifier systems, the phenotypes of a genetic algorithm. They consist of a set of rules `m_rules` (the single classifiers) which we can

```
1 public enum EClasses {
2   /** class A */
3   A,
4   /** class B */
5   B,
6   /** class N */
7   N;
8 }
```

Listing 26.1: The enum `EClasses` with the possible DMC 2007 classifications.

represent as `byte` arrays containing the conditions and rule outputs `m_results` (instances of `EClasses`).

Listing 26.2 illustrates how the method `classify` works which classifies a set of 17 relevant features (stored in the array `data`) into one of the three possible `EClasses` instances. It iterates over all the rules in `m_rules` and checks if rule `m_rules[i]` fits perfectly according to the definitions in Table 22.2 on page 378. If so, the corresponding classification `m_results[i]` is returned. `classify` keeps also track on the rule which has the fewest violated conditions in the variables `lec` and `leci`. If no perfectly matching rule could be found, the $\frac{1}{5}$-threshold mentioned in Section 22.1.2 is checked: if `lec <= 3`, the classification `m_results[leci]` belonging to the rule with the least violations is returned. Otherwise, the class **N** represented by `EClasses.N` is assigned to the data sample.

So this is basically what a *phenotype* can look like in Sigoa – you can clearly see that, except from implementing `java.io.Serializable`, no further requirements are imposed on its structure. The method `classify` is not mandatory, it is will just be part of the evaluation in this particular optimization problem; other problems may need totally other functionality.

26.1.2 The Genotype and the Embryogeny

The *genotype* that belongs to the phenotypic individual representations is a variable-length a bit string. Such genomes have been discussed in Section 3.5 on page 149 extensively. In Figure 22.4, we have introduced the genotype-phenotype mapping in this particular application: since there are four possible conditions and 17 conditions plus three possible classifications (**A**, **B**, and **N**) per rule, we need $17 * 2 + 2 = 36$ bits to encode a single classifier which is the *granularity*, i. e., the gene size of our genome. A classifier system in turn may consist of an arbitrary number of such classifiers.

In Sigoa, we can represent variable-length as well as fixed-length bit strings as byte arrays (`byte[]`) for which predefined creation, mutation, and crossover operators exist. Therefore, we do not have to deal with the reproduction operations directly and can concentrate on the translation of a genotype $g \in$ `byte[]` into a corresponding phenotype which is an instance of `ClassifierSystem`. Such translations is called genotype-phenotype mapping (see Section 3.8 on page 155) or artificial embryogeny (discussed in Section 3.8) for which Sigoa offers a core interface `IEmbryogeny` (see **??** on page ??) and a reference implementation `Embryogeny` (see **??** on page ??) along with an extension for bitstrings, `BitStringEmbryogeny` (see **??** on page ??) which provides special streams for input and output of structured data from and to bit strings. We simply need to extend this class by providing (at least) the transformation function gpm from genotypes to phenotypes and (optionally) vice versa. Listing 26.3 shows this extension in form of the class `ClassifierEmbryogeny`. The constant `CLASSIFIER_EMBRYOGENY` provides a globally shared instance of this new embryogeny.

26.1.3 The Simulation

So now we need to find out how an evolved classifier system C behaves. Therefore we can use the provided test datasets or better, only a good share of them while saving the rest in order to check if our classifier system generalize well. For these training sets, we built a matrix $M(C)$ where the columns denote the classification results delivered by C and the rows contain the true classes. For determining the zero-based indices into this matrix we use the method `ordinal()` of the `EClasses` enum, i. e., $m_{2,1}$ would represent those elements in class **N** that were miss-classified into group **B** – 2799 in the example matrix M_{ex} of Equation 26.1. From M_{ex}, we can furthermore read that 1087 of the samples belonging to class **B** were correctly classified whereas 777 were assigned to class **A** and 1462 to class **N**.

$$M_{ex}(C) = \begin{pmatrix} 4062 & 856 & 3794 \\ 777 & 1087 & 1462 \\ 5484 & 2799 & 29\,679 \end{pmatrix} \tag{26.1}$$

```
1  public class ClassifierSystem extends JavaTextable implements
       Serializable {
2   ...
3    private final byte[][] m_rules;
4    private final EClasses[] m_results;
5    ...
6    public ClassifierSystem(final byte[][] rules, final EClasses[]
       results) {
7      super();
8      this.m_results = results;
9      this.m_rules   = rules; }
10   ...
11   public EClasses classify(final byte[] data) {
12     byte[][] rules;
13     byte[]   rule;
14     int      i, j, ec, lec, leci;
15
16     rules = this.m_rules;
17     lec   = Integer.MAX_VALUE;
18     leci  = 0;
19
20     main: for (i = (rules.length - 1); i >= 0; i--) {
21       rule = rules[i];
22       ec = 0;
23       for (j = (rule.length - 1); j >= 0; j--) {
24         switch (rule[j]) {
25         case 0: {
26           if (data[j] != 0)
27             if ((++ec) > 3) continue main;
28           break;
29         }
30         case 1: {
31           if (data[j] < 1) // != 1
32             if ((++ec) > 3) continue main;
33           break;
34         }
35         case 2: {
36           if (data[j] <= 1)// <= 0)
37             if ((++ec) > 3) continue main;
38           break;
39         }
40         }
41       }
42       if (ec <= 0) return this.m_results[i];
43       if (ec < lec) {
44         lec  = ec;
45         leci = i;
46       }
47     }
48     if (lec <= 3) return this.m_results[leci];
49     return EClasses.N;
50   }
51 ...
52 }
```

Listing 26.2: The structure of our DMC 2007 classifier system.

```
1  public class ClassifierEmbryogeny extends
       BitStringEmbryogeny<ClassifierSystem> {
2    /** the classes */
3    private static final EClasses[] CLASSES = EClasses.values();
4    /** The globally shared instance */
5    public static final IEmbryogeny<byte[], ClassifierSystem>
         CLASSIFIER_EMBRYOGENY = new ClassifierEmbryogeny();
6  ..
7    /** This method is supposed to compute an instance of
8     * the phenotype from an instance of the genotype.
9     * @param genotype The genotype instance to breed a
10    *                 phenotype from.
11    * @return The phenotype hatched from the genotype. */
12   @Override
13   public ClassifierSystem hatch(final byte[] genotype) {
14     int                  i, j, c;
15     byte[][]             rules;
16     byte[]               rule;
17     EClasses[]           results;
18     BitStringInputStream bis;
19
20     if (genotype == null)
21       throw new NullPointerException();
22     c       = ((genotype.length * 8) / 36);
23     rules   = new byte[c][17];
24     results = new EClasses[c];
25     bis     = this.acquireBitStringInputStream();
26     bis.init(genotype);
27
28     for (i = 0; i < c; i++) {
29       rule = rules[i];
30       for (j = 0; j < 17; j++) {
31         rule[j] = (byte) (bis.readBits(2));
32       }
33       results[i] = (CLASSES[bis.readBits(2) % 3]);
34     }
35     this.releaseBitStringInputStream(bis);
36     return new ClassifierSystem(rules, results);
37   }
38 ...
39 }
```

Listing 26.3: The embryogeny component of our DMC 2007 contribution.

From such matrices, we can easily compute the profit function $P(C)$ as well as other features, like how many **A**s, **B**s, and **N**s were classified incorrectly. What we basically do here is to simulate the behavior of the classifiers. And for simulations, Sigoa provides the interface `ISimulation` (see ?? on page ??) and its standard implementation `Simulation` (see ?? on page ??). This default implementation just needs to be extended so it uses the training samples, which we load somewhere else (in a class called `Datasets`), and computes M. Therefore, overriding the method `beginIndividual` is sufficient and other changes are not needed.

Listing 26.4 shows the most important code of the new class `ClassificationSimulation`. In order to allow us to publish the new simulation in the simulation manager of the optimization job system, we also provide a globally shared factory in form of an `ISimulationProvider`-instance with the constant `PROVIDER` in line 3.

```
1  public class ClassificationSimulation extends
       Simulation<ClassifierSystem> {
2    /** the shared provider for this simulation */
3    public static final ISimulationProvider PROVIDER = new
         SimulationProvider(ClassificationSimulation.class);
4    /** the matrix M(C) */
5    private final int[][] m_classifications;
6  ...
7    public ClassificationSimulation() {
8      super();
9      this.m_classifications = new int[3][3]; }
10
11   /** Here the matrix M(C) is computed
12    * @param what The classifier C to be simulated. */
13   @Override
14   public void beginIndividual(final ClassifierSystem what)
15     {
16     int     i;
17     int[][] x = this.m_classifications;
18     super.beginIndividual(what);
19     for (i = (x.length - 1); i >= 0; i--)
20       Arrays.fill(x[i], 0);
21     for (i = (DATA.length - 1); i >= 0; i--)
22       x[CLASSES[i].ordinal()][ what.classify(DATA[i]).ordinal()]++;
23   }
24 ...
25   /** Compute the profit value P(C). */
26   public int getProfit() {
27     int[][] data = this.m_classifications;
28     return (3 * data[0][0]) + (6 * data[1][1]) - (data[2][0] +
           data[2][1] + data[0][1] + data[1][0]);
29   }
30 }
```

Listing 26.4: The simulation for testing the DMC 2007 classifier system.

26.1.4 The Objective Functions

On the foundation of the new simulation for classifier system, we can define the objective functions that should guide the evolution. In Section 22.1.2 on page 379 we have introduced the two most important objective functions: one that minimizes $f_1(C) = -P(C)$ and hence, maximizes the profit, and $f_2(C) = |C|$ which minimizes the number of rules in the classifier system.

All objective functions in Sigoa are instances of the interface `IObjectiveFunction` (see ?? on page ??). They can be derived from its default implementation `ObjectiveFunction` (see ?? on page ??) which implements the basic functionality so only the real mathematical computations need to be added.

In Listing 26.5, we implement f_1. Therefore, the method `endEvaluation` needs to be overridden. Here we store negated profit into a state record which is used by the optimization system to compute the objective value and to store it into the individual records later on.

The only remaining question is: How will the optimizer system know that our objective function needs an instance of `ClassificationSimulation` and that it has to call its method `beginIndividual` beforehand? The answer is relatively simple: In line 3, we have defined an instance of `SimulationProvider` for the `ClassificationSimulation`. This provider will later be introduced to the optimization job system. It uses `ClassificationSimulation.class` as identifier per default. With the method `getRequiredSimulationId` on line 16, we tell the

```
1  public class ProfitObjectiveFunction extends
       ObjectiveFunction<ClassifierSystem, ObjectiveState, Serializable,
       ClassificationSimulation> {
2    ...
3    /** This method is called after any simulation/
4     * evaluation is performed. It stores the negated
5     * profit -P(C) into the state-variable - that's all.*/
6    @Override
7    public void endEvaluation(final ClassifierSystem individual, final
         ObjectiveState state, final Serializable staticState, final
         ClassificationSimulation simulation) {
8      state.setObjectiveValue(-simulation.getProfit());
9    }
10 ..
11   /**
12    * Obtain the ID of the required simulator.
13    * @return The ID=class of our simulator */
14   @Override
15   public Serializable getRequiredSimulationId() {
16     return ClassificationSimulation.class;
17   }
18 }
```

Listing 26.5: The profit objective function $f_1(C) = -P(C)$ for the DMC 2007.

```
1  public class SizeObjectiveFunction extends
       ObjectiveFunction<ClassifierSystem, ObjectiveState, Serializable,
       ISimulation<ClassifierSystem>> {
2    /** This method is called after any simulation/
3     * evaluation is performed. It stores the size of
4     * the \ClassS\ |C| into the state-
5     * variable - that's all. */
6    @Override
7    public void endEvaluation(final ClassifierSystem individual, final
         ObjectiveState state, final Serializable staticState, final
         ISimulation<ClassifierSystem> simulation) {
8      state.setObjectiveValue(Math.max(individual.getSize(), 3));
9    }
10 }
```

Listing 26.6: The size objective function $f_2(C) = |C|$ for the DMC 2007.

job system that we need a simulation which is made available by an provider with exactly this ID. Before passing the simulation to our objective function, the job system will call its `beginIndividual` method which, in turn, builds the matrix $M(C)$ holding the information needed for its `getProfit` method. Now we can query the profit from this simulation.

For the secondary objective function f_2 defined in Listing 26.6, we do not need any simulation. Instead, we directly query the number of rules in the classifier system via the method `getSize`. In Listing 26.2, we have omitted this routine for space reasons, it simply returns `m_rules.length`. Again, this value is stored into the state record passed in from the evaluator component of the job system which will then do the rest of the work.

26.1.5 The Evolution Process

Now the work is almost done, we just need to start the optimization process. Listing 26.7 presents a `main`-method which is called on startup of a Java program and does so. Therefore, we first have to decide which global optimization algorithm should be used and pick `ElitsitEA`[1], an elitist evolutionary algorithm (per default steady-state) with a population size of $10 * 1024$ and mutation and crossover rates of 0.4 in line 8.

Then we construct an `IOptimizationInfo`-record with all the information that will guide the evolution[2]. Part of this information is how the solution candidates should be evaluated. For this, we use an instance of `Evaluator`[3] (line 15) which is equipped with a `List` containing the two new objective functions from 10 and 12. We furthermore tell the system to perform a pure Pareto-optimization as discussed in Section 1.2.2 on page 31 by passing the globally shared instance of `ParetoComparator`[4] (line 16) into the info record. We then define that our embryogeny component should be used to translate the bit string genotypes into `ClassifierSystem` phenotypes in line 17. These genotypes are produced by the default reproduction operators for variable-length bit string genomes[5] added in lines 18 to 20. All of them are created with a *granularity* of 36 which means that it is ensured that all genotypes have sizes of multiples of 36 bits and that all crossover operations only occur at such 36 bit boundaries.

After this is done, we instantiate `SimulationManager`[6] and publish the new simulation that we have developed in Section 26.1.3 on page 446 by adding its provider to the simulation manager in line 27. The job system created in line 28 allows the evaluator to access the simulation manager, an instance of the interface `ISimulationManager`[7]. The evaluator will then ask its objective functions which simulations they need – in our case a `ClassificationSimulation` – and then query the simulation manager to provide them.

In line 28, we decided to use a multi-processor job system which is capable of transparently parallelizing the optimization process. The different types of job systems which are instances of the interface `IJobSystem` specified in **??** on page ?? are discussed in **??** on page ??. We add our optimizer to the system in line 29 and finally start it in 30. Since we have added no termination criterion, the system will basically run forever in this example.

In order to get information on its progress, we have provided two special output pipes (see **??** on page ??) in lines 23 and 24 to the optimizer's non-prevalence pipe. Through this pipe, in each generation all non-prevailed (in our case, non-dominated) individuals will be written and thus, pass our two pipes. In each generation, new text files with information about them are created. The first one, the `IndividualPrinterPipe`, uses the directory *c* and creates files that start with a *c* followed by the current generation index. It writes down the full information about all individuals. From these files, we can later easily reconstruct the complete individuals and, for instance, integrate them into the real applications. The second printer pipe, an instance of `ObjectivePrinterPipe`, stores only the objective values in a comma-separated-values format. The output files are put into the directories *bo* and also start with *bo* followed by the generation index. Such files are especially useful for getting a quick overview on how the evolution progresses. Later, they may also read into spread sheets or graphical tools in order to produce fancy diagrams like Figure 22.5 on page 380.

---

[1] see **??** on page ??

[2] `IOptimizationInfo` is discussed in **??** on page ??, its reference implementation `OptimizationInfo` in **??** on page ??.

[3] The class `Evaluator`, discussed in **??** on page ??, is the default implementation of the interface `IEvaluator` specified in **??** on page ??.

[4] The class `ParetoComparator`, elaborated on in **??** on page ??, implements the interface `IComparator` defined in **??** on page ??.

[5] These operations are introduced in **??** on page ?? implement the interfaces `ICreator`, `IMutator`, and `ICrossover` specified in **??** on page ??.

[6] see **??** on page ??

[7] see **??** on page ??

```
public static void main(String[] args) {
    EA<byte[], ClassifierSystem> opt;
    IOptimizationInfo<byte[], ClassifierSystem> oi;
    IJobSystem s;
    SimulationManager m;
    List<IObjectiveFunction<ClassifierSystem, ?, ?,
        ISimulation<ClassifierSystem>>> l;

    opt = new ElitistEA<byte[], ClassifierSystem>(10 * 1024, 0.4d,
        0.4d);

    l = new ArrayList<IObjectiveFunction<ClassifierSystem, ?, ?,
        ISimulation<ClassifierSystem>>>();
    l.add(new ProfitObjectiveFunction());
    l.add(new SizeObjectiveFunction());

    oi  = new OptimizationInfo<byte[], ClassifierSystem>(
            new Evaluator<ClassifierSystem>(l),
            ParetoComparator.PARETO_COMPARATOR,
            ClassifierEmbryogeny.CLASSIFIER_EMBRYOGENY,
            new VariableLengthBitStringCreator(36),
            new VariableLengthBitStringMutator(36),
            new VariableLengthBitStringNPointCrossover(36),
            null);

    opt.addNonPrevailedPipe(new IndividualPrinterPipe<byte[],
        ClassifierSystem>(new FileTextWriterProvider(new
        File("c"),"c"), false));
    opt.addNonPrevailedPipe(new ObjectivePrinterPipe<byte[],
        ClassifierSystem>(new FileTextWriterProvider(new File("bo"),
        "bo"), false));

    m = new SimulationManager();
    m.addProvider(ClassificationSimulation.PROVIDER);
    s = new MultiProcessorJobSystem(m);
    s.executeOptimization(opt, new JobInfo<byte[],
        ClassifierSystem>(oi));
    s.start();
  }
```

Listing 26.7: A `main` method that runs the evolution for the 2007 DMC.

Part IV

Background

27

Set Theory

Set theory[1] [550, 880, 1967] is an important part of the mathematical theory. Numerous other disciplines like algebra, analysis and topology are based up on it. Since set theory (and the topics to follow) is not the topic of this book but a mere prerequisite, this chapter (and the ones to follow) will just briefly introduce it. We make heavy use of wild definitions and in some cases even use roughly cut stuff short. More information on the topics discussed can be retrieved from the literature references.

Set theory can be divided into naïve set theory[2] and axiomatic set theory[3]. The first approach, the naïve set theory, is inconsistent and therefore not regarded in this book.

Definition 27.1 (Set). A set is a collection of objects considered as a whole[4]. The objects of a set are called elements or members. They can be anything, from numbers and vectors, to complex data structures, algorithms, or even other sets. Sets are conventionally denoted with capital letters, A, B, C, etc. while their elements are usually referred to with small letters a, b, c.

27.1 Set Membership

The expression $a \in A$ means that the element a is a member of the set A while $y \notin A$ means that y is not a member of A. There are three common forms to define sets:

1. With their elements in braces: $A = \{1, 2, 3\}$ defines a set A containing the three elements 1, 2, and 3. $\{1, 1, 2, 3\} = \{1, 2, 3\}$ since the curly braces only denote set membership.
2. The same set can be specified using logical operators to describe its elements: $\forall b \in \mathbb{N} : (b \geq 1) \wedge (b < 4) \Leftrightarrow b \in B$.
3. A shortcut for the previous form is to denote the logical expression in braces, like $C = \{(c \geq 1) \wedge (c < 4), c \in \mathbb{N}\}$.

The cardinality of a set A is written as $|A|$ and stands for the count of elements in the set.

27.2 Relations between Sets

Two sets A and B are said to be equal, written $A = B$, if they have the same members. They are not equal ($A \neq B$) if either a member of A is not an element of B or an element

[1] http://en.wikipedia.org/wiki/Set_theory [accessed 2007-07-03]
[2] http://en.wikipedia.org/wiki/Naive_set_theory [accessed 2007-07-03]
[3] http://en.wikipedia.org/wiki/Axiomatic_set_theory [accessed 2007-07-03]
[4] http://en.wikipedia.org/wiki/Set_%28mathematics%29 [accessed 2007-07-03]

of B is not a member of A. If all elements of the set A are also elements of the set B, A is called subset of B and B is the superset of A. We write $A \subset B$ if A is a (true) subset of but not equal to B. $A \subseteq B$ means the A is a subset of B and may be equal to B. If A is no subset of but may be equal to B, $A \not\subset B$ is written. $A \not\subseteq B$ means that A is neither a subset of nor equal to B.

$$A = B \equiv x \in A \Leftrightarrow x \in B \tag{27.1}$$
$$A \neq B \equiv (\exists x : x \in A \wedge x \notin B) \vee (\exists y : y \in B \wedge y \notin A) \tag{27.2}$$
$$A \subseteq B \equiv x \in A \Rightarrow x \in B \tag{27.3}$$
$$A \subset B \equiv A \subseteq B \wedge \exists y : y \in B \wedge y \notin A \tag{27.4}$$
$$A \not\subseteq B \equiv \exists x : x \in A \wedge x \notin B \tag{27.5}$$
$$A \not\subset B \equiv (A = B) \vee (\exists x : x \in A \wedge x \notin B) \tag{27.6}$$

27.3 Special Sets

Special sets used in the context of this book are

1. The empty set $\emptyset = \{\}$ contains no elements ($|\emptyset| = 0$).
2. The natural numbers[5] $\mathbb{N}$ include all whole numbers bigger than 0. ($\mathbb{N} = \{1, 2, 3, ..\}$)
3. The natural numbers including 0 ($\mathbb{N}_0$) include all whole numbers bigger than or equal to 0. ($\mathbb{N}_0 = \{1, 2, 3, ..\}$)
4. $\mathbb{Z}$ is the set of all integers, positive and negative. ($\mathbb{Z} = \{.., -2, -1, 0, 1, 2, ..\}$)
5. The rational numbers[6] $\mathbb{Q}$ are defined as $\mathbb{Q} = \left\{ \frac{a}{b} : a, b \in \mathbb{Z}, b \neq 0 \right\}$.
6. All real numbers[7] $\mathbb{R}$ include all rational and rational numbers (such as $\sqrt{2}$).
7. $\mathbb{R}^+$ denotes the positive real numbers including 0: ($\mathbb{R}^+ = [0, \infty)$).
8. $\mathbb{C}$ is the set of complex numbers[8]. When needed in the context of this book, we abbreviate the immaginary unit with i, and the real and imaginary parts of a complex number z with Rez and Imz.

$$\mathbb{N} \subset \mathbb{N}_0 \subset \mathbb{Z} \subset \mathbb{Q} \subset \mathbb{R} \subset \mathbb{C} \tag{27.7}$$
$$\mathbb{N} \subset \mathbb{N}_0 \subset \mathbb{R}^+ \subset \mathbb{R} \subset \mathbb{C} \tag{27.8}$$

For these numerical sets, special subsets, so called intervals, can be specified. $[1, 5)$ is a set which contains all the numbers starting from (including) 1 up to (exclusively) 5. $(1, 5]$ on the other hand contains all numbers bigger than 1 and up to inclusively 5. In order to avoid ambiguities, such sets will always used in a context where it is clear if the numbers in the set are natural or real.

27.4 Operations on Sets

Let us now define the possible unary and binary operations on sets, some of which are illustrated in Figure 27.1.

[5] http://en.wikipedia.org/wiki/Natural_numbers [accessed 2008-01-28]
[6] http://en.wikipedia.org/wiki/Rational_number [accessed 2008-01-28]
[7] http://en.wikipedia.org/wiki/Real_numbers [accessed 2008-01-28]
[8] http://en.wikipedia.org/wiki/Complex_number [accessed 2008-01-29]

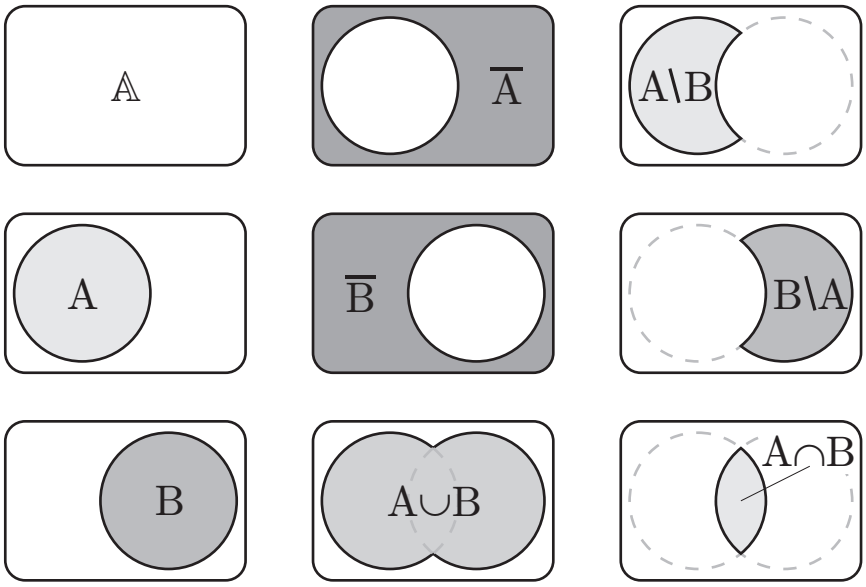


Figure 27.1: Set operations performed on sets A and B inside a set $\mathbb{A}$

Definition 27.2 (Set Union). The union[9] C of two sets A and B is written as $A \cup B$ and contains all the objects that are element of at least one of the sets.

$$C = A \cup B \Leftrightarrow ((c \in A) \vee (c \in B) \Leftrightarrow (c \in C)) \tag{27.9}$$
$$A \cup B = B \cup A \tag{27.10}$$
$$A \cup \emptyset = A \tag{27.11}$$
$$A \cup A = A \tag{27.12}$$
$$A \subseteq A \cup B \tag{27.13}$$

Definition 27.3 (Set Intersection). The intersection[10] D of two sets A and B, denoted by $A \cap B$, contains all the objects that are elements of both of the sets. If $A \cap B = \emptyset$, meaning that A and B have no elements in common, they are called *disjoint*.

$$D = A \cap B \Leftrightarrow ((d \in A) \wedge (d \in B) \Leftrightarrow (d \in D)) \tag{27.14}$$
$$A \cap B = B \cap A \tag{27.15}$$
$$A \cap \emptyset = \emptyset \tag{27.16}$$
$$A \cap A = A \tag{27.17}$$
$$A \cap B \subseteq A \tag{27.18}$$

Definition 27.4 (Set Difference). The difference E of two sets A and B, $A \setminus B$, contains the objects that are element of A but not of B.

$$E = A \setminus B \Leftrightarrow ((e \in A) \wedge (e \notin B) \Leftrightarrow (e \in E)) \tag{27.19}$$
$$A \setminus \emptyset = A \tag{27.20}$$
$$\emptyset \setminus A = \emptyset \tag{27.21}$$
$$A \setminus A = \emptyset \tag{27.22}$$
$$A \setminus B \subseteq A \tag{27.23}$$

[9] http://en.wikipedia.org/wiki/Union_%28set_theory%29 [accessed 2007-07-03]
[10] http://en.wikipedia.org/wiki/Intersection_%28set_theory%29 [accessed 2007-07-03]

Definition 27.5 (Set Complement). The complementary set $\overline{A}$ of the set A in a set $\mathbb{A}$ includes all the elements which are in $\mathbb{A}$ but not element of A:

$$A \subseteq \mathbb{A} \Rightarrow \overline{A} = \mathbb{A} \setminus A \tag{27.24}$$

Definition 27.6 (Cartesian Product). The Cartesian product[11] P of two sets A and B, denoted $P = A \times B$ is the set of all ordered pairs (a, b) whose first component is an element from A and the second is an element of B.

$$P = A \times B \Leftrightarrow P = \{(a, b) : a \in A, b \in B\} \tag{27.25}$$

$$A^n = \underbrace{A \times A \times .. \times A}_{n \text{ times}} \tag{27.26}$$

Definition 27.7 (Countable Set). A set S is called countable[12] if there exists an injective function[13] $\exists f : S \mapsto \mathbb{N}$.

Definition 27.8 (Uncountable Set). A set is uncountable if it is not countable, i. e., no such function exists for the set. $\mathbb{N}$, $\mathbb{Z}$, and $\mathbb{Q}$ are countable, $\mathbb{R}$ and $\mathbb{R}^+$are not.

Definition 27.9 (Power Set). The power set[14] $\mathcal{P}(A)$ of the set A is the set of all subsets of A.

$$\forall p \in \mathcal{P}(A) \Leftrightarrow p \subseteq A \tag{27.27}$$

27.5 Tuples

Definition 27.10 (Type). A type is a set of values that a variable, constant, function, or similar entity may take on.

We can, for instance, specify the type $T = \{1, 2, 3\}$. A variable x which is an instance of this type then can take on the values 1, 2, or 3.

Definition 27.11 (Tuple). A tuple[15] is an ordered, finite sequence of elements, where each element is an instance of a certain type.

To each position in i a tuple t, a type T_i is assigned. The element $t_{[i]}$ at a position i must then be an element of T_i. Other than sets, tuples may contain the same element twice. Since every item of a tuple may be of a different type, $(Monday, 23, \{a, b, c\})$ is a valid tuple.

In the context of this book, we define tuples with parenthesis like (a, b, c) whereas sets are specified using braces $\{a, b, c\}$.

Definition 27.12 (Tuple Type). To formalize this relation, we define the tuple type T. T specifies the basic sets for the elements of its tuples. If a tuple t meets the constraints imposed to its values by T, we can write $t \in T$ which means that the tuple t is an instance of T.

$$T = (T_1, T_2, ..T_n), n \in \mathbb{N} \tag{27.28}$$

$$t = (t_1, t_2, ..t_n) \in T \Leftrightarrow t_i \in T_i \; \forall 0 \leq i < n \wedge \text{len}(t) = \text{len}(T) \tag{27.29}$$

[11] http://en.wikipedia.org/wiki/Cartesian_product [accessed 2007-07-03]
[12] http://en.wikipedia.org/wiki/Countable_set [accessed 2007-07-03]
[13] see definition of function on page 462
[14] http://en.wikipedia.org/wiki/Axiom_of_power_set [accessed 2007-07-03]
[15] http://en.wikipedia.org/wiki/Tuple [accessed 2007-07-03]

27.6 Lists

Definition 27.13 (List). Lists[16] are abstract data types which can be regarded as special tuples. They are sequences where every item is of the same type.

Other than our discussions on set theory, the following text about the data structure *list* is strictly local to this book and not to be understood as a general mathematical theory. All the functions and operations defined on lists in this book are only given in order to allow for a clear and well-defined notation in the other parts of the book, when specifying optimization algorithms, for instance. They are not founded on related work by any other scientist.

We introduce functions that will add elements to or remove elements from lists; that sort lists or search within them. Like tuples, lists can be defined using parenthesis in this book. The single elements of a list are accessed by their index written in brackets ($(a,b,c)_{[1]} = b$) where the first element has the index 0 and the last element has the index $n-1$ (while n is the count of elements in the list: $n = \text{len}((a,b,c)) = 3$). The empty list is abbreviated with $()$.

Definition 27.14 (createList**).** The $l = \text{createList}(n, q)$ method creates a new list l of the length n filled with the item q.

$$l = \text{createList}(n,q) \Leftrightarrow \text{len}(l) = n \wedge \forall 0 \leq i < n \Rightarrow l_{[i]} = q \tag{27.30}$$

Definition 27.15 (insertListItem**).** The function $m = \text{insertListItem}(l, i, q)$ creates a new list m by inserting one element q in a list l at the index $0 \leq i \leq \text{len}(l)$. By doing so, it shifts all elements located at index i and above to the right by one position.

$$\begin{aligned} m = \text{insertListItem}(l,i,q) \Leftrightarrow\ & \text{len}(m) = \text{len}(l) + 1 \wedge m_{[i]} = q \wedge \\ & \forall j : 0 \leq j < i \Rightarrow m_{[j]} = l_{[j]} \wedge \\ & \forall j : i \leq j < \text{len}(l) \Rightarrow m_{[j+1]} = l_{[j]} \end{aligned} \tag{27.31}$$

Definition 27.16 (addListItem**).** The addListItem function is a shortcut for inserting one item at the end of a list:

$$\text{addListItem}(l,q) \equiv \text{insertListItem}(l, \text{len}(l), q) \tag{27.32}$$

Definition 27.17 (deleteListItem**).** The function $m = \text{deleteListItem}(l, i)$ creates a new list m by removing the element at index $0 \leq i < \text{len}(l)$ from the list l ($\text{len}(l) \geq i + 1$).

$$\begin{aligned} m = \text{deleteListItem}(l,i) \Leftrightarrow\ & \text{len}(m) = \text{len}(l) - 1 \wedge \\ & \forall j : 0 \leq j < i \Rightarrow m_{[j]} = l_{[j]} \\ & \forall j : i < j < \text{len}(l) \Rightarrow m_{[j-1]} = l_{[j]} \end{aligned} \tag{27.33}$$

Definition 27.18 (deleteListRange**).** The method $m = \text{deleteListRange}(l, i, c)$ creates a new list m by removing c elements beginning at index $0 \leq i < \text{len}(l)$ from the list l ($\text{len}(l) \geq i + c$).

$$\begin{aligned} m = \text{deleteListRange}(l,i,c) \Leftrightarrow\ & \text{len}(m) = \text{len}(l) - c \wedge \\ & \forall j : 0 \leq j < i \Rightarrow m_{[j]} = l_{[j]} \wedge \\ & \forall j : i + c \leq j < \text{len}(l) \Rightarrow m_{[j-c]} = l_{[j]} \end{aligned} \tag{27.34}$$

Definition 27.19 (appendList**).** The function $\text{appendList}(l_1, l_2)$ is a shortcut for adding all the elements of a list l_2 to a list l_1. We define it recursively as:

$$\text{appendList}(l_1,l_2) \equiv \begin{cases} l_1 & \text{if } \text{len}(l_2) = 0 \\ \text{appendList}(\text{addListItem}(l_1, l_{2[0]}), \text{deleteListItem}(l_2, 0)) & \text{otherwise} \end{cases} \tag{27.35}$$

[16] http://en.wikipedia.org/wiki/List_%28computing%29 [accessed 2007-07-03]

Definition 27.20 (countOccurences**).** The function countOccurences(x, l) returns the number of occurrences of the element x in the list l.

$$\text{countOccurences}(x, l) = |\{i \in 0 \ldots \text{len}(l) - 1 : l_{[i]} = x\}| \tag{27.36}$$

Definition 27.21 (subList**).** The method subList(l, i, c) extracts c elements from the list l beginning at index i and returns them as a new list.

$$\text{subList}(l, i, s) \equiv \text{deleteListRange}(\text{deleteListRange}(l, 0, i), c, |l| - i - c) \tag{27.37}$$

Definition 27.22 (Sorting Lists). It is often useful to have sorted lists[17]. Thus we define the functions $S = \text{sortList}_a(U, \text{cmp})$ and $S = \text{sortList}_d(U, \text{cmp})$ which sort a list U in ascending or descending order using a comparator function $\text{cmp}(u_1, u_2)$.

$$S = \text{sortList}_a(U, \text{cmp}) \tag{27.38}$$

$$\forall u \in U \; \exists i \in [0, \text{len}(U) - 1] : S_{[i]} = u \tag{27.39}$$

$$\text{len}(S) = \text{len}(U) \tag{27.40}$$

$$\forall 0 \leq i < \text{len}(U) - 1 \Rightarrow \text{cmp}(S_{[i]}, S_{[i+1]}, \leq) \, 0 \tag{27.41}$$

For $S = \text{sortList}_d(U, \text{cmp})$, only Equation 27.41 changes, the rest stays valid:

$$S = \text{sortList}_d(U, s) \tag{27.42}$$

$$\forall 0 \leq i < \text{len}(U) - 1 \Rightarrow \text{cmp}(S_{[i]}, S_{[i+1]}, \geq) \, 0 \tag{27.43}$$

The concept of comparator functions has been introduced in Definition 1.15 on page 38. $\text{cmp}(u_1, u_2)$ returns a negative value if u_1 is smaller than u_2, a positive number if u_1 is greater than u_2, and 0 if both are equal. Comparator functions are very versatile, they from the foundation of the sorting mechanisms of the Java framework [838, 837], for instance. In global optimization, they are perfectly suited to represent the Pareto dominance or prevalence relations introduced in Section 1.2.2 on page 31 and Section 1.2.4. Sorting according to a specific function f of only one parameter can easily be performed by building the comparator $\text{cmp}(u_1, u_2) \equiv (f(u_1) - f(u_2))$. Thus, we will furthermore synonymously use the sorting predicate also with unary functions f.

$$\text{sortList}(U, f) \equiv \text{sortList}(U, \text{cmp}(u_1, u_2) \equiv (f(u_1) - f(u_2))) \tag{27.44}$$

A list U can be sorted in $\mathbf{O}(\text{len}(U) \log \text{len}(U))$ time complexity. For concrete examples of sorting algorithms, see [1163, 446, 1850].

Definition 27.23 (Searching in Unsorted Lists). Searching an element u in an unsorted list U means walking through it until either the element is found or the end of the whole list has been scanned, which corresponds to complexity $\mathbf{O}(\text{len}(U))$.

$$\text{searchItem}_u(u, U) = \begin{cases} i : U_{[i]} = u & \text{if } u \in U \\ -1 & \text{otherwise} \end{cases} \tag{27.45}$$

Definition 27.24 (Searching in Sorted Lists). Searching an element s in sorted list S means to perform a binary search[18] returning the index of the element if it is contained in S. If $s \notin S$, a negative number is returned indicating the position where the element could be inserted into the list without violating its order. The function $\text{searchItem}_{as}(s, S,)$searches in an ascending sorted list, $\text{searchItem}_{ds}(s, S,)$searches in a descending sorted list. Searching in a sorted list is done in $\mathbf{O}(\log \text{len}(S))$ time. For concrete algorithm examples, again see [1163, 446, 1850].

[17] http://en.wikipedia.org/wiki/Sorting_algorithm [accessed 2007-07-03]
[18] http://en.wikipedia.org/wiki/Binary_search [accessed 2007-07-03]

$$\text{searchItem}_{as}(s,S) = \begin{cases} i : S_{[i]} = s & \text{if } s \in S \\ (-i-1) : (\forall j \geq 0, j < i \Rightarrow S_{[j]} \leq s) \wedge & \text{otherwise} \\ \quad (\forall j < \text{len}(S), j \geq i \Rightarrow S_{[j]} > s) & \end{cases} \tag{27.46}$$

$$\text{searchItem}_{ds}(s,S) = \begin{cases} i : S_{[i]} = s & \text{if } s \in S \\ (-i-1) : (\forall j \geq 0, j < i \Rightarrow S_{[j]} \geq s) \wedge & \text{otherwise} \\ \quad (\forall j < \text{len}(S), j \geq i \Rightarrow S_{[j]} < s) & \end{cases} \tag{27.47}$$

Definition 27.25 (removeListItem**).** The function removeListItem(l, q) finds one occurrence of an element q in a list l by using the appropriate search algorithm and deletes it (returning a new list m).

$$m = \text{removeListItem}(l,q) \Leftrightarrow \begin{cases} l & \text{if searchItem}(q,l) < 0 \\ \text{deleteListItem}(l, \text{searchItem}(q,l)) & \text{otherwise} \end{cases} \tag{27.48}$$

We can further define transformations between sets and lists which will implicitly be used when needed in this book. It should be noted that "setToList" is not the inverse function of listToSet.

$$\begin{aligned} B = \text{setToList}(\textit{set } A) \Rightarrow\ & \forall a \in A\ \exists i : B_{[i]} = a \wedge \\ & \forall i \in [0, \text{len}(B)-1] \Rightarrow B_{[i]} \in A \wedge \\ & \text{len}(\text{setToList}(A)) = |A| \end{aligned} \tag{27.49}$$

$$\begin{aligned} A = \text{listToSet}(\textit{list } B) \Rightarrow\ & \forall i \in [0, \text{len}(B)-1] \Rightarrow B_{[i]} \in A \wedge \\ & \forall a \in A\ \exists i \in [0..\text{len}(B)-1] : B_{[i]} = a \wedge \\ & |\text{listToSet}(B)| \leq \text{len}(B) \end{aligned} \tag{27.50}$$

27.7 Binary Relations

Definition 27.26 (Binary Relation). A binary[19] relation[20] R is defined as an ordered triple (A, B, P) where A and B are arbitrary sets, and P is a subset of the Cartesian product $A \times B$ (see Equation 27.25). The sets A and B are called the domain and codomain of the relation and P is called its graph. The statement $(a, b) \in P : a \in A \wedge b \in B$ is read "a is R-related to b" and is written as $R(a, b)$. The order of the elements in each pair of P is important: If $a \neq b$, then $R(a, b)$ and $R(b, a)$ both can be `true` or `false` independently of each other.

Some types and possible properties of binary relations are listed below and illustrated in Figure 27.2. A binary relation can be [673]:

1. Left-total if
$$\forall a \in A\ \exists b \in B : R(a,b) \tag{27.51}$$
2. Surjective[21] or right-total if
$$\forall b \in B\ \exists a \in A : R(a,b) \tag{27.52}$$

[19] http://en.wikipedia.org/wiki/Binary_relation [accessed 2007-07-03]
[20] http://en.wikipedia.org/wiki/Relation_%28mathematics%29 [accessed 2007-07-03]
[21] http://en.wikipedia.org/wiki/Surjective [accessed 2007-07-03]

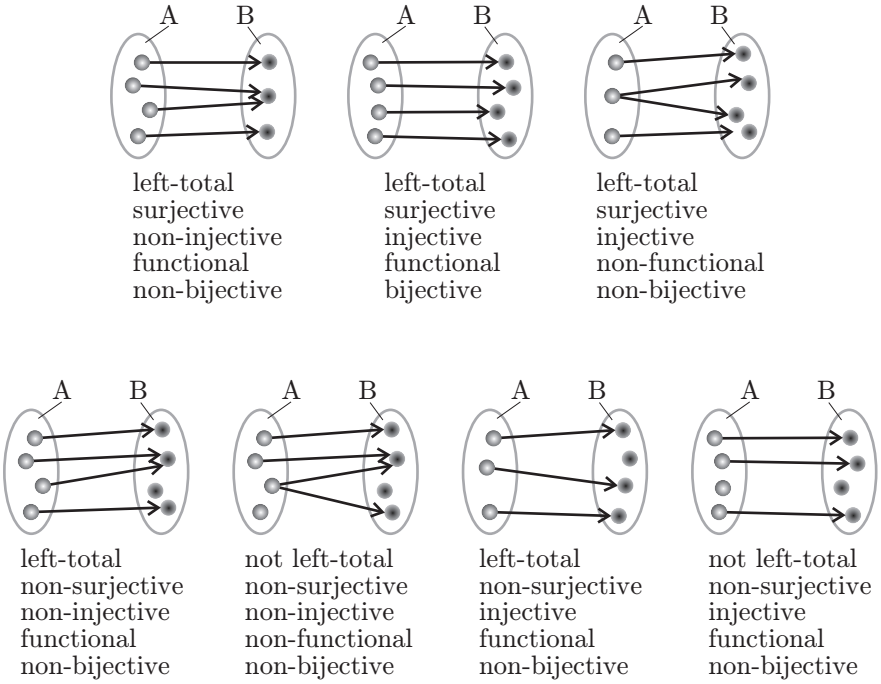


Figure 27.2: Properties of a binary relation R with domain A and codomain B.

3. Injective[22] if
$$\forall a_1, a_2 \in A, b \in B : R(a_1, b) \wedge R(a_2, b) \Rightarrow a_1 = a_2 \tag{27.53}$$
4. Functional if
$$\forall a \in A, b_1, b_2 \in B : R(a, b_1) \wedge R(a, b_2) \Rightarrow b_1 = b_2 \tag{27.54}$$
5. Bijective[23] if it is left-total, right-total and functional.
6. Transititve[24] if
$$\forall a \in A,\ \forall b \in B,\ \forall c \in A \cap B\ :\ R(a, c) \wedge R(c, b) \Rightarrow R(a, b) \tag{27.55}$$

27.7.1 Functions

Definition 27.27 (Function). A function f is a binary relation with the property that for an element x of the domain[25] X there is no more than one element y in the codomain Y such that x is related to y. This uniquely determined element y is denoted by $f(x)$. In other words, a function is a functional binary relation and we can write:

$$\forall x \in X, y_1, y_2 \in Y : f(x, y_1) \wedge f(x, y_2) \Rightarrow y_1 = y_2 \tag{27.56}$$

A function maps each element of X to one element in Y. The domain X is the set of possible input values of f and the codomain Y is the set its *possible* outputs. The set of all *actual* outputs $\{f(x) : x \in X\}$ is called range. This distinction between range and codomain can be made obvious with a small example. The sine function can be defined as a mapping from the real numbers to the real numbers $\sin : \mathbb{R} \mapsto \mathbb{R}$, making $\mathbb{R}$ its codomain. Its actual range however is just the real interval $[-1, 1]$.

[22] http://en.wikipedia.org/wiki/Injective [accessed 2007-07-03]
[23] http://en.wikipedia.org/wiki/Bijective [accessed 2007-07-03]
[24] http://en.wikipedia.org/wiki/Transitive_relation [accessed 2007-07-03]
[25] http://en.wikipedia.org/wiki/Domain_%28mathematics%29 [accessed 2007-07-03]

Monotonicity

Real functions are monotone, i. e., have the property of monotonicity[26], if they preserve a given order[27].

Definition 27.28 (Monotonically Increasing). A function $f : X \mapsto Y$ that maps a subset of the real numbers $X \subseteq \mathbb{R}$ to a subset of the real numbers $Y \subseteq \mathbb{R}$ is called monotonic, monotonically increasing, increasing, or non-decreasing, if and only if Equation 27.57 holds.

$$\forall x_1 < x_2,\ x_1, x_2 \in X \Rightarrow f(x_1) \leq f(x_2) \tag{27.57}$$

Definition 27.29 (Monotonically Decreasing). A function $f : X \mapsto Y$ that maps a subset of the real numbers $X \subseteq \mathbb{R}$ to a subset of the real numbers $Y \subseteq \mathbb{R}$ is called monotonically decreasing, decreasing, or non-increasing, if and only if Equation 27.58 holds.

$$\forall x_1 < x_2,\ x_1, x_2 \in X \Rightarrow f(x_1) \geq f(x_2) \tag{27.58}$$

27.7.2 Order Relations

All of us have learned the meaning and the importance of order since the earliest years in school. The alphabet is ordered, the natural numbers are ordered, the marks on our school reports are ordered, and so on. Matter of fact, we come into contact with orders even way before entering school by learning to distinguish things according to their size, for instance.

Order relations[28] are another type of binary relations which is used to express the order amongst the elements of a set A. Since order relations are imposed on single sets, both their domain and their codomain are the same (A, in this case). For such relations, we can define an additional number of properties which can be used to characterize and distinguish the different types of order relations:

1. Antisymmetrie:
$$R(a_1, a_2) \wedge R(a_2, a_1) \Rightarrow a_1 = a_2\ \forall a_1, a_2 \in A \tag{27.59}$$
2. Asymmetrie
$$R(a_1, a_2) \Rightarrow \neg R(a_2, a_1)\ \ \forall a_1, a_2 \in A \tag{27.60}$$
3. Reflexivenss
$$R(a, a)\ \ \forall a \in A \tag{27.61}$$
4. Irreflexivenss
$$\nexists a \in A : R(a, a) \tag{27.62}$$

All order relations are transitive[29], and either antisymmetric or symmetric and either reflexive or irreflexive:

Definition 27.30 (Partial Order). A binary relation R defines a (*non-strict*, *reflexive*) partial order if and only if it is reflexive, antisymmetric, and transitive.

The $\leq$ and $\geq$ operators, for instance, represent non-strict partial orders on the set of the complex numbers $\mathbb{C}$. Partial orders that correspond to the $>$ and $<$ comparators are called *strict*. The Pareto dominance relation introduced in Definition 1.13 on page 31 is another example for such a strict partial order.

Definition 27.31 (Strict Partial Order). A binary relation R defines a *strict* (or *irreflexive*) partial order if it is irreflexive, asymmetric, and transitive.

[26] http://en.wikipedia.org/wiki/Monotonic_function [accessed 2007-08-08]
[27] Order relations are discussed in Section 27.7.2.
[28] http://en.wikipedia.org/wiki/Order_relation [accessed 2007-07-03]
[29] See Equation 27.55 on the facing page for the definition of transitivity.

Definition 27.32 (Total Order). A total order[30] (or linear order, simple order) R on the set A is a partial order which is complete/total.

$$R(a_1, a_2) \vee R(a_2, a_1) \ \forall a_1, a_2 \in A \tag{27.63}$$

The real numbers $\mathbb{R}$ for example are totally ordered whereas on the set of complex numbers $\mathbb{C}$, only (strict or reflexive) partial (non-total) orders can be defined because it is continuous in two dimensions.

27.7.3 Equivalence Relations

Another important class of relations are equivalence relations[31] [2093, 2141] which are often abbreviated with $\equiv$ or $\sim$, i. e., $a_1 \equiv a_2$ and $a_1 \sim a_2$ mean $R(a_1, a_2)$ for the equivalence relation R imposed on the set A and $a_1, a_2 \in A$. Unlike order relations, equivalence relations are *symmetric*, i. e.,

$$R(a_1, a_2) \Rightarrow R(a_2, a_1) \ \forall a_1, a_2 \in A \tag{27.64}$$

Definition 27.33 (Equivalence Relation). The binary relation R defines an equivalence relation on the set A if and only if it is reflexive, symmetric, and transitive.

Definition 27.34 (Equivalence Class). If an equivalence relation R is defined on a set A, the subset $A' \subseteq A$ of A is an equivalence class[32] if and only if $\forall a_1, a_2 \in A' \Rightarrow R(a_1, a_2)$ $(a_1 \sim a_2)$.

[30] http://en.wikipedia.org/wiki/Total_order [accessed 2007-07-03]
[31] http://en.wikipedia.org/wiki/Equivalence_relation [accessed 2007-07-28]
[32] http://en.wikipedia.org/wiki/Equivalence_class [accessed 2007-07-28]

28

Stochastic Theory and Statistics

In this chapter we give a rough introduction into stochastic theory[1] [1720, 1264, 1043, 1044], which subsumes

1. probability[2] theory[3], the mathematical study of phenomena characterized by randomness or uncertainty, and
2. statistics[4], the art of collecting, analyzing, interpreting, and presenting data.

28.1 General Information

28.1.1 Books

Some books about (or including significant information about) Stochastic Theory and Statistics are:

Kallenberg [1084]: *Foundations of modern probability*
Rényi [1720]: *Probability Theory*
Tijms [2041]: *Understanding Probability: Chance Rules in Everyday Life*
Feller [649]: *An Introduction to Probability Theory and Its Applications*
Kallenberg [1085]: *Probabilistic symmetries and invariance principles*
Jaynes [1043, 1044]: *Probability Theory: The Logic of Science*
Lawler [1264]: *Introduction to Stochastic Processes*
Casella and Berger [350]: *Statistical Inference*
Lowry [1310]: *Concepts and Applications of Inferential Statistics*
Lowry [1313]: *VassarStats: Web Site for Statistical Computing*
Siegel and Castellan Jr. [1878]: *Nonparametric Statistics for The Behavioral Sciences*
Sheskin [1866]: *Handbook of Parametric and Nonparametric Statistical Procedures*
Bhattacharyya and Johnson [205]: *Statistical Concepts and Methods*
Bortz, Lienert, and Boehnke [252]: *Verteilungsfreie Methoden in der Biostatistik*
Polasek [1652]: *Schließende Statistik – Einführung in die Schätz- und Testtheorie für Wirtschaftswissenschaftler*
Edgington [619]: *Randomization tests*
Harlow, Mulaik, and Steiger [898]: *What If There Were No Significance Tests?*
Dallal [478]: *The Little Handbook of Statistical Practice*

[1] http://en.wikipedia.org/wiki/Stochastic [accessed 2007-07-03]
[2] http://en.wikipedia.org/wiki/Probability [accessed 2007-07-03]
[3] http://en.wikipedia.org/wiki/Probability_theory [accessed 2007-07-03]
[4] http://en.wikipedia.org/wiki/Statistics [accessed 2007-07-03]

Heath [912]: *An introduction to experimental design and statistics for biology*
Kanji [1091]: *100 Statistical Tests*
Neyman and Pearson [1523]: *Joint Statistical Papers*
Lindley and Scott [1290]: *New Cambridge Statistical Tables*
Rice [1727]: *Mathematical Statistics and Data Analysis*
Panik [1607]: *Advanced Statistics from an Elementary Point of View*
Kay [1105]: *Fundamentals of Statistical Signal Processing, Volume I: Estimation Theory*
Box, Hunter, and Hunter [263]: *Statistics for Experimenters: Design, Innovation, and Discovery*
Fisher [682]: *The design of experiments*
Cox and Reid [460]: *The Theory of the Design of Experiments*
Fisher [684]: *Statistical methods and scientific inference*
Fisher [680]: *Statistical Methods for Research Workers*
Casella and Berger [351]: *Statistical Inference*
Robert and Casella [1744]: *Monte Carlo Statistical Methods*
Liu [1294]: *Monte Carlo Strategies in Scientific Computing*
Yates [2288]: *The Design and Analysis of Factorial Experiments*
Snyder and Miller [1914]: *Random Point Processes in Time and Space*
Devroye [556]: *Non-Uniform Random Variate Generation*
Poor [1668]: *An Introduction to Signal Detection and Estimation*
Van Trees [2100]: *Detection, Estimation, and Modulation Theory, Part I*
Simon [1882]: *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*
Kleinbaum, Kupper, and Muller [1150]: *Applied regression analysis and other multivariable methods*
Draper and Smith [595]: *Applied regression analysis*
Fox [739]: *Applied Regression Analysis, Linear Models, and Related Methods*
Banks [134]: *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice*
Mackeown [1339]: *Stochastic Simulation in Physics*
Osborne and Rubinstein [1587]: *A Course in Game Theory*
Fudenberg and Tirole [752]: *Game Theory*
Kindermann and Snell [1139]: *Markov Random Fields and Their Applications*
Bennett [178]: *The Collected Papers of R.A. Fisher*

28.2 Probability

Probability theory is used to determine the likeliness of the occurrence of an event under ideal mathematical conditions. [1084, 1085]

Definition 28.1 (Random Experiment). Random experiments can be repeated arbitrary often, their results cannot be predicted.

Definition 28.2 (Elementary Event). The possible outcomes of random situations are called elementary events or samples ω.

Definition 28.3 (Sample Space). The set of all possible outcomes (elementary events, samples) of a random situation is the sample space $\Omega = \{\omega_i : i \in 1..N = |\Omega|\}$.

When throwing dice[5], for example, the sample space will be $\Omega = \omega_1, \omega_2, \omega_3, \omega_4, \omega_5, \omega_6$ whereas ω_i means that the number i was thrown.

Definition 28.4 (Random Event). A random event A is a subset of the sample space Ω ($A \subseteq \Omega$). If $\omega \in A$ occurs, then A is occurs too.

[5] Throwing a dice is discussed as example for stochastic extensively in Section 28.6 on page 497.

Definition 28.5 (Certain Event). The certain event is the random event will always occur in each repetition of a random experiment. Therefore, it is equal to the whole sample space Ω.

Definition 28.6 (Impossible Event). The impossible event will never occur in any repetition of a random experiment, it is defined as $\emptyset$.

Definition 28.7 (Conflicting Events). Two conflicting events A_1 and A_2 can never occur together in a random experiment. Therefore, $A_1 \cap A_2 = \emptyset$.

28.2.1 Probabily as defined by Bernoulli (1713)

In some idealized situations, like throwing ideal coins or ideal dice, all elementary events of the sample space have the same probability de Laplace [523].

$$P(\omega) = \frac{1}{|\Omega|} \; \forall \omega \in \Omega \tag{28.1}$$

Equation 28.1 is also called the Laplace-assumption. If it holds, the probability of an event A can be defined as:

$$P(A) = \frac{\textit{number of possible events in favor of } A}{\textit{number of possible events}} = \frac{|A|}{|\Omega|} \tag{28.2}$$

For many random experiments of this type, we can use combinatorical[6] approaches in order to determine the number of possible outcomes. Therefore, we want to shortly outline the mathematical concepts of factorial numbers, combinations, and permutations.[7]

Definition 28.8 (Factorial). The factorial[8] $n!$ of a number $n \in \mathbb{N}_0$ is the product of n and all natural numbers smaller than it. It is a specialization of the Gamma function for positive integer numbers, see Section 28.10.1 on page 532.

$$n! = \prod_{i=1}^{n} i \tag{28.3}$$

$$0! = 1 \tag{28.4}$$

In combinatorial mathematics[9], we often want to know in how many ways we can arrange $n \in \mathbb{N}$ elements from a set Ω with $M = |\Omega| \geq n$ elements. We can distinguish between *combinations*, where the order of the elements in the arrangement plays no role, and *permutations*, where it is important. (a, b, c) and (c, b, a), for instance, denote the same combination but different permutations of the elements $\{a, b, c\}$. We furthermore distinguish between arrangements where each element of Ω can occurred at most once (without repetition) and arrangements where the same elements may occur multiple time (with repetition).

Combinations

The number of possible combinations[10] $C(M, n)$ of $n \in \mathbb{N}$ elements out of a set Ω with $M = |\Omega| \geq n$ elements without repetition is

[6] http://en.wikipedia.org/wiki/Combinatorics [accessed 2007-07-03]
[7] http://en.wikipedia.org/wiki/Combinations_and_permutations [accessed 2007-07-03]
[8] http://en.wikipedia.org/wiki/Factorial [accessed 2007-07-03]
[9] http://en.wikipedia.org/wiki/Combinatorics [accessed 2008-01-31]
[10] http://en.wikipedia.org/wiki/Combination [accessed 2008-01-31]

$$C(M,n) = \binom{M}{n} = \frac{M!}{n!\,(M-n)!} \tag{28.5}$$

$$\binom{M}{n} = \frac{M}{n} * \frac{M-1}{n-1} * \frac{M-2}{n-2} * .. * \frac{M-n+1}{1} \tag{28.6}$$

$$C(M+1,n) = C(M,n) + C(M,n-1) = \binom{M+1}{n} = \binom{M}{n} + \binom{M}{n-1} \tag{28.7}$$

If the elements of Ω may repeatedly occur in the arrangements, the number of possible combinations becomes

$$\frac{M+n-1!}{n!\,(M-1)!} = \binom{M+n-1}{n} = \binom{M+n-1}{n-1} = C(M+n-1,n) = C(M+n-1,n-1) \tag{28.8}$$

Permutations

The number of possible permutations[11] $Perm(M,n)$ of $n \in \mathbb{N}$ elements out of a set Ω with $M = |\Omega| \geq n$ elements without repetition is

$$Perm(M,n) = (M)_n = \frac{M!}{(M-n)!} \tag{28.9}$$

If an element from Ω can occur more than once in the arrangements, the number of possible permutations is

$$M^n \tag{28.10}$$

28.2.2 The Limiting Frequency Theory of von Mises

If we repeat a random experiment multiple times, the number of occurrences of a certain event should somehow reflect its probability. The more often we perform the experiment, the more reliable will the estimations of the event probability become. We can express this relation using the notation of *frequency*.

Definition 28.9 (Absolute Frequency). The number $H(A,n)$ denoting how often an event A occurred during n repetitions of a random experiment is its absolute frequency[12].

Definition 28.10 (Relative Frequency). The relative frequency $h(A,n)$ of an event A is its absolute frequency normalized to the total number of experiments n. The relative frequency has the following properties:

$$h(A,n) = \frac{H(A,n)}{n} \tag{28.11}$$

$$0 \leq h(A,n) \leq 1 \tag{28.12}$$

$$h(\Omega,n) = 1 \; \forall n \in \mathbb{N} \tag{28.13}$$

$$A \cap B = \emptyset \Rightarrow h(A \cup B,n) = \frac{H(A,n) + H(B,n)}{n} = h(A,n) + h(B,n) \tag{28.14}$$

According to von Mises [2120], the (statistical) probability $P(A)$ of an event A computing the limit of its relative frequency $h(A,n)$ as n approaching infinity. This is the limit of the quotient of the number of elementary events favoring A and the number of all possible elementary events for infinite many repetitions. [2120, 2121]

$$P(A) = \lim_{n\to\infty} h(A,n) = \lim_{n\to\infty} \frac{n_A}{n} \tag{28.15}$$

[11] http://en.wikipedia.org/wiki/Permutations [accessed 2008-01-31]
[12] http://en.wikipedia.org/wiki/Frequency_%28statistics%29 [accessed 2007-07-03]

28.2.3 The Axioms of Kolmogorov

Definition 28.11 (σ-algebra). A subset S of the power set $\mathcal{P}(\Omega)$ is called σ-algebra[13], if the following axioms hold:

$$\Omega \in S \tag{28.16}$$
$$\emptyset \in S \tag{28.17}$$
$$A \in S \Leftrightarrow \overline{A} \in S \tag{28.18}$$
$$A \in S \wedge B \in S \Rightarrow (A \cup B) \in S \tag{28.19}$$

From these axioms others can be deduced, for example:

$$A \in S \wedge B \in S \Rightarrow \overline{A} \in S \wedge \overline{B} \in S \tag{28.20}$$
$$\Rightarrow \overline{A} \cup \overline{B} \in S$$
$$\Rightarrow \overline{\overline{A} \cup \overline{B}} \in S \tag{28.21}$$
$$\Rightarrow A \cap B \in S$$
$$A \in S \wedge B \in S \Rightarrow (A \cap B) \in S \tag{28.22}$$

Definition 28.12 (Probability Space). A probability space (or random experiment) is defined by the triple (Ω, S, P) whereas

1. Ω is the sample space, a set of elementary events,
2. S is a σ-algebra defined on Ω, and
3. $P\omega$ defines a probability measure[14] that determines the probability of occurrence for each event $\omega \in \Omega$. (Kolmogorov [1169] axioms[15])

Definition 28.13 (Probability). A mapping P which maps a real number to each elementary event $\omega \in \Omega$ is called probability measure if and only if the σ-algebra S on Ω holds:

$$\forall A \in S \Rightarrow 0 \leq P(A) \leq 1 \tag{28.23}$$
$$P(\Omega) = 1 \tag{28.24}$$
$$\forall \textit{disjoint } A_i \in S \Rightarrow P(A) = P\left(\bigcup_{\forall i} A_i\right) = \sum_{\forall i} P(A_i) \tag{28.25}$$

From these axioms, it can be deduced that:

$$P(\emptyset) = 0 \tag{28.26}$$
$$P(A) = 1 - P\left(\overline{A}\right) \tag{28.27}$$
$$P\left(A \cap \overline{B}\right) = P(A) - P(A \cap B) \tag{28.28}$$
$$P(A \cup B) = P(A) + P(B) - P(A \cap B) \tag{28.29}$$

28.2.4 Conditional Probability

Definition 28.14 (Conditional Probability). The conditional probability[16] $P(A\,|B)$ is the probability of some event A, given the occurrence of some other event B. $P(A\,|B)$ is read "the probability of A, given B".

[13] http://en.wikipedia.org/wiki/Sigma-algebra [accessed 2007-07-03]
[14] http://en.wikipedia.org/wiki/Probability_measure [accessed 2007-07-03]
[15] http://en.wikipedia.org/wiki/Kolmogorov_axioms [accessed 2007-07-03]
[16] http://en.wikipedia.org/wiki/Conditional_probability [accessed 2007-07-03]

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \tag{28.30}$$

$$P(A \cap B) = P(A|B)\, P(B) \tag{28.31}$$

Definition 28.15 (Statistical Independence). Two events A and B are (statistically) independent if and only if $P(A \cap B) = P(A)\, P(B)$ holds. If two events A and B are statistically independent, we can deduce:

$$P(A \cap B) = P(A)\, P(B) \tag{28.32}$$

$$P(A\,|B\,) = P(A) \tag{28.33}$$

$$P(B\,|A\,) = P(B) \tag{28.34}$$

28.2.5 Random Variable

Definition 28.16 (Random Variable). The function X which relates the sample space Ω to the real numbers $\mathbb{R}$ is called random variable[17] in the probability space (Ω, S, P).

$$X : \Omega \mapsto \mathbb{R} \tag{28.35}$$

Using such a random variable, we can replace the sample space Ω with the new sample space Ω_X. Furthermore, the σ-algebra S can be replaced with a σ-algebra S_X, which consists of subsets of Ω_X instead of Ω. Last but not least, we replace the probability measure P which relates the $\omega \in \Omega$ to the interval $[0, 1]$ by a new probability measure P_X which relates the real numbers $\mathbb{R}$ to this interval.

Definition 28.17 (Probability Space of a Random Variable). Is $X : \Omega \mapsto \mathbb{R}$ a random variable, then the probability space of X is defined as the triplet

$$(\Omega_X, S_X, P_X) \tag{28.36}$$

One example for such a new probability measure would be the probability that a random variable X takes on a real value which is smaller or equal a value x:

$$P_X(X \leq x) = P(\{\omega : \omega \in \Omega \wedge X(\omega) \leq x\}) \tag{28.37}$$

28.2.6 Cumulative Distribution Function

Definition 28.18 (Cumulative Distribution Function). If X is a random variable of a probability space $(\Omega_X = \mathbb{R}, S_X, P_X)$, we call the function $F_X : \mathbb{R} \mapsto [0, 1]$ the (cumulative) distribution function[18] (CDF) of the random variable Xif it fulfills Equation 28.38.

$$F_X := \underbrace{P_X(X \leq x)}_{\textit{definition rnd. var.}} \equiv \underbrace{P(\{\omega : \omega \in \Omega \wedge X(\omega) \leq x\})}_{\textit{definition probability space}} \tag{28.38}$$

A cumulative distribution function F_X has the following properties:

1. $F_X(X)$ is normalized:

$$\underbrace{\lim_{x \to -\infty} F_X(x) = 0}_{\textit{impossible event}}, \quad \underbrace{\lim_{x \to +\infty} F_X(x) = 1}_{\textit{certain event}} \tag{28.39}$$

[17] http://en.wikipedia.org/wiki/Random_variable [accessed 2007-07-03]
[18] http://en.wikipedia.org/wiki/Cumulative_distribution_function [accessed 2007-07-03]

2. $F_X(X)$ is monotonously[19] growing:

$$F_X(x_1) \leq F_X(x_2) \ \forall x_1 \leq x_2 \tag{28.40}$$

3. $F_X(X)$ is (right-sided) continuous[20]:

$$\lim_{h \to 0} F_X(x+h) = F_X(x) \tag{28.41}$$

4. The probability that the random variable X takes on values in the interval $x_0 \leq X \leq x_1$ can be computed using the CDF:

$$P(x_0 \leq X \leq x_1) = F_X(x_1) - F_X(x_0) \tag{28.42}$$

5. The probability that the random variable X takes on the value of a single random number x:

$$P(X = x) = F_X(x) - \lim_{h \to 0} F_X(x-h) \tag{28.43}$$

We can further distinguish between sample spaces Ω which contain at most countable infinite many elements and such that are continuums. Hence, we there are discrete[21] and continuous[22] random variables.

Definition 28.19 (Discrete Random Variable). A random variable X (and its probability measure $P_X(X)$ respectively) is called discrete if it takes on at most countable infinite many values. Its cumulative distribution function $F_X(X)$ therefore has the shape of a stairway.

Definition 28.20. A random variable X (and its probability measure P_X respectively) is called continuous if it can take on uncountable infinite many values and its cumulative distribution function $F_X(X)$ is also continuous.

28.2.7 Probability Mass Function

Definition 28.21 (Probability Mass Function). The probability mass function[23] (PMF) f_X is defined discrete distributions only and assigns a probability to each value a discrete random variable X can take on.

$$f_X : \mathbb{Z} \mapsto [0,1] \ : \ f_X(x) := P_X(X = x) \tag{28.44}$$

Therefore, we can specify the relation between the PMF and its corresponding(discrete) CDF in Equation 28.45 and Equation 28.45. We can further define the probability of an event A in Equation 28.47 using the PMF.

$$P_X(X \leq x) = F_X(x) = \sum_{i=-\infty}^{x} f_X(x) \tag{28.45}$$

$$P_X(X = x) = f_X(x) = F_X(x) - F_X(x-1) \tag{28.46}$$

$$P_X(A) = \sum_{\forall x \in A} f_X(x) \tag{28.47}$$

[19] http://en.wikipedia.org/wiki/Monotonicity [accessed 2007-07-03]
[20] http://en.wikipedia.org/wiki/Continuous_function [accessed 2007-07-03]
[21] http://en.wikipedia.org/wiki/Discrete_random_variable [accessed 2007-07-03]
[22] http://en.wikipedia.org/wiki/Continuous_probability_distribution [accessed 2007-07-03]
[23] http://en.wikipedia.org/wiki/Probability_mass_function [accessed 2007-07-03]

28.2.8 Probability Density Function

The probability density function[24] (PDF) is the counterpart of the PMF for continuous distributions. The PDF *does not* represent the probabilities of the single values of a random variable. Since a continuous random variable can take on uncountable many values, each distinct value itself has the probability 0. If we, for instance, picture the current temperature outside as (continuous) random variable, the probability that it takes on the value 18 for 18°C is zero. It will never be *exactly* 18°C outside, we can at most declare with a certain probability that we have a temperature between 17.99999°C and 18.00001°C.

Definition 28.22 (Probability Density Function). If a random variable X is continuous, its probability density function f_X is defined as

$$f_X : \mathbb{R} \mapsto [0, \infty) \ : \ F_X(x) = \int_{-\infty}^{+\infty} f_X(\xi)\, d\xi \ \forall x \in \mathbb{R} \tag{28.48}$$

28.3 Stochastic Properties

Each random variable X which conforms to a probability distribution F_X may have certain properties such as a maximum and a mean value, a variance, and a value which will be taken on by X most often. If the cumulative distribution function F_X of X is known, these values can usually be computed directly from its parameters. On the other hand, it is possible that we only know the values $A_{[i]}$ which X took on during some random experiments. From this set of sample data A, we can *estimate* the properties of the underlying (possibly unknown) distribution of X using statistical methods (with a certain error, of course).

In the following, we will elaborate on the properties of a random variable $X \in \mathbb{R}$ both from the viewpoint of knowing the PMF/PDF $f_X(x)$ and the CDF $F_X(x)$ as well as from the statistical perspective, where only a sample A of past values of X is known. In the latter case, we define the sample as a list A with the length $n = \text{len}(A)$ and the elements $A_{[i]} : i \in [0, n-1]$.

28.3.1 Count, Min, Max and Range

The most primitive features of a random distribution are the minimum, maximum, and the range of its values, as well as the number of values $A_{[i]}$ in a data sample A.

Definition 28.23 (Count). $n = \text{len}(A)$ is the number of elements in the data sample A.

This item count is only defined for data samples, not for random variables, since random variables represent experiments which can infinitely be repeated and thus stand for infinitely many values. The number of items should not be mixed up with the possible number of different values the random variable may take on. A data sample A may contain the same value a multiple times. When throwing a dice seven times, one may throw $A = (1, 4, 3, 3, 2, 6, 1)$, for example[25].

Definition 28.24 (Minimum). There exists no smaller element in the sample data A than the minimum sample $\check{a} \equiv \min(A)$ when speaking statistics. From the perspective of the cumulative distribution function F_X, the minimum is the lower boundary $\check{x}$ of the random variable X(or negative infinity, if no such boundary exists). Both definitions are fully compliant to Definition 1.10 on page 25.

$$\min(A) \equiv \check{a} \in A : \ \forall a \in A \Rightarrow \check{a} \leq A \tag{28.49}$$

$$\check{x} = \min(X) \Leftrightarrow F_X(\check{x}) > 0 \wedge F_X(x) \geq F_X(\check{x}) \ \forall x \in \mathbb{R} \tag{28.50}$$

[24] http://en.wikipedia.org/wiki/Probability_density_function [accessed 2007-07-03]

[25] Throwing a dice is discussed as example for stochastic extensively in Section 28.6 on page 497.

Definition 28.25 (Maximum). In statistically evaluated sample data, exists element bigger than the maximum $\hat{a} = \max(A)$ in A. The value $\hat{x}$ is the upper boundary of the values a random variable X may take on (or positive infinity, if X is unbounded). This definition is compliant with Definition 1.9 on page 25.

$$\max(A) \equiv \hat{a} \in A : \forall a \in A \Rightarrow \hat{a} \geq A \tag{28.51}$$

$$\hat{x} = \max(X) \Leftrightarrow F_X(\hat{x}) \geq F_X(x) \ \forall x \in \mathbb{R} \tag{28.52}$$

Definition 28.26 (Range).
The range $\text{range}(A)$ of the sample data A is the difference of the maximum $\max(A)$ and the minimum $\min(A)$ of A and therefore represents the span covered by the data. If a random variable X is limited in both directions, it has a finite range $\text{range}(X)$, otherwise this range is infinite too.

$$\text{range}(A) = \hat{a} - \check{a} = \max(A) - \min(A) \tag{28.53}$$

$$\text{range}(X) = \hat{x} - \check{x} = \max(X) - \min(X) \tag{28.54}$$

28.3.2 Expected Value and Arithmetic Mean

The expected value EX and the $\overline{a}$ are basic measures for random variables and data samples that help us to estimate the regions where their values will be distributed around.

Definition 28.27 (Expected Value). The expected value[26] of a random variable X is the sum of the probability of each possible outcome of the random experiment multiplied by the outcome value. It is abbreviated by EX or μ. For discrete distributions it can be computed using Equation 28.55 and for continuous ones Equation 28.56 holds.

$$EX = \sum_{x=-\infty}^{\infty} x f_X(x) \tag{28.55}$$

$$EX = \int_{-\infty}^{\infty} x f_X(x)\, dx \tag{28.56}$$

If the expected value EX of a random variable X is known, the following statements can be derived the expected values of some related random variables as follows:

$$Y = a + X \Rightarrow EY = a + EX \tag{28.57}$$

$$Z = bX \Rightarrow EZ = bEX \tag{28.58}$$

Definition 28.28 (Sum). The $\text{sum}(A)$ represents the sum of all elements in a set of data samples A. This value does, of course, only exist in statistics.

$$\text{sum}(A) = \sum_{i=0}^{n-1} A_{[i]} \tag{28.59}$$

Definition 28.29 (Arithmetic Mean). The arithmetic mean[27] $\overline{a}$ is the sum of all elements in the sample data A divided by the total number of values. In the spirit of the limiting frequency method of von Mises [2120], it is an estimation of the expected value $\overline{a} \approx EX$ of the random variable X that produced the sample data A.

$$\overline{a} = \frac{\text{sum}(A)}{n} = \frac{1}{n}\sum_{i=0}^{n-1} A_{[i]} = \sum_{i=0}^{n-1} h(A_{[i]}, n) \tag{28.60}$$

[26] http://en.wikipedia.org/wiki/Expected_value [accessed 2007-07-03]
[27] http://en.wikipedia.org/wiki/Arithmetic_mean [accessed 2007-07-03]

28.3.3 Variance and Standard Deviation

The variance[28] [677] is a measure of statistical dispersion. It illustrates how close the results of a random variable or the elements a in a data sample A are to their expected value EX or their arithmetical mean $\overline{a}$.

Definition 28.30 (Variance of a Random Variable). The variance $D^2X \equiv \text{var}(X) \equiv \sigma^2$ of a random variable X is defined as

$$\text{var}(X) = D^2X = E\Big[(X - EX)^2\Big] = E\big[X^2\big] - (EX)^2 \tag{28.61}$$

The variance of a discrete random variable X can be computed using Equation 28.62 and for continuous distributions, Equation 28.63 will hold.

$$\begin{aligned} D^2X &= \sum_{x=-\infty}^{\infty} f_X(x)\,(x - EX)^2 = \sum_{x=-\infty}^{\infty} x^2 f_X(x) - \left[\sum_{x=-\infty}^{\infty} x f_X(x)\right]^2 \\ &= \left[\sum_{x=-\infty}^{\infty} x^2 f_X(x)\right] - (EX)^2 \end{aligned} \tag{28.62}$$

$$\begin{aligned} D^2X &= \int_{-\infty}^{\infty} (x - EX)^2\,dx = \int_{-\infty}^{\infty} x^2 f_X(x)\,dx - \left[\int_{-\infty}^{\infty} x f_X(x)\,dx\right]^2 \\ &= \left[\int_{-\infty}^{\infty} x^2 f_X(x)\,dx\right] - (EX)^2 \end{aligned} \tag{28.63}$$

If the variance D^2X of a random variable X is known, we can derive the variances of some related random variables as follows:

$$Y = a + X \Rightarrow D^2Y = D^2X \tag{28.64}$$

$$Z = bX \Rightarrow D^2Z = b^2D^2X \tag{28.65}$$

Definition 28.31 (Sum of Squares). The function sumSqrs(A) is only defined for statistical data and represents the sum of the squares of all elements in the data sample A.

$$\text{sumSqrs}(A) = \sum_{i=0}^{n-1} (A_{[i]})^2 \tag{28.66}$$

Definition 28.32 (Variance Estimator). We define the (unbiased) estimator[29] s^2 of the variance of the random variable which produced the sample values A according to Equation 28.67. The variance is zero for all samples with $(n = \text{len}(A)) \leq 1$.

$$s^2 = \frac{1}{n-1}\sum_{i=0}^{n-1} (A_{[i]} - \overline{a})^2 == \frac{1}{n-1}\left(\text{sumSqrs}(A) - \frac{(\text{sum}(A))^2}{n}\right) \tag{28.67}$$

Definition 28.33 (Standard Deviation). The standard deviation[30] is the square root of the variance. The standard deviation of a random variable X is abbreviated with DX and σ, its statistical estimate is s.

$$DX = \sqrt{D^2X} \tag{28.68}$$

$$s = \sqrt{s^2} \tag{28.69}$$

The standard deviation is zero for all samples with $n \leq 1$.

[28] http://en.wikipedia.org/wiki/Variance [accessed 2007-07-03]
[29] see Definition 28.55 on page 499
[30] http://en.wikipedia.org/wiki/Standard_deviation [accessed 2007-07-03]

Definition 28.34 (Coefficient of Variation). The coefficient of variation[31] c_V of a random variable X is the ratio of the standard deviation by expected value of X. For data samples, its estimate $c_{\approx V}$ is defined as the ration of the estimate of the standard deviation and the arithmetic mean.

$$c_V = \frac{DX}{EX} = \frac{\sigma}{\mu} \tag{28.70}$$

$$c_{\approx V} = \frac{n}{\text{sum}(A)} \sqrt{\frac{\text{sumSqrs}(A) - \frac{(\text{sum}(A))^2}{n}}{n-1}} \tag{28.71}$$

Definition 28.35 (Covariance). The covariance[32] $\text{cov}(X, Y)$ of two random variables X and Y is a measure for how much they are related. It exists if the expected values EX^2 and EY^2 exist and is defined as

$$\text{cov}(X, Y) = E[X - EX] * E[Y - EY] \tag{28.72}$$

$$= E[X * Y] - EX * EY \tag{28.73}$$

$$\tag{28.74}$$

If X and Y are statistically independent, then their covariance is zero, since

$$E[X * Y] = EX * EY \tag{28.75}$$

Furthermore, the following formulas hold for the covariance

$$D^2X = \text{cov}(X, X) \tag{28.76}$$

$$D^2[X + Y] = \text{cov}(X + Y, X + Y) = D^2X + D^2Y + 2\text{cov}(X, Y) \tag{28.77}$$

$$D^2[X - Y] = \text{cov}(X - Y, X + Y) = D^2X + D^2Y - 2\text{cov}(X, Y) \tag{28.78}$$

$$\text{cov}(X, Y) = \text{cov}(Y, X) \tag{28.79}$$

$$\text{cov}(aX, Y) = a\,\text{cov}(Y, X) \tag{28.80}$$

$$\text{cov}(X + Y, Z) = \text{cov}(X, Z) + \text{cov}(Y, Z) \tag{28.81}$$

$$\text{cov}(aX + b, cY + d) = a\,c\,\text{cov}(X, Y) \tag{28.82}$$

28.3.4 Moments

Definition 28.36 (Moment). The k^{th} moment[33] $\mu'_k(c)$ about a value c is defined for a random distribution X as

$$\mu'_k(c) = E\Big[(X - c)^k\Big] \tag{28.83}$$

It can be specified for discrete (Equation 28.84) and continuous (Equation 28.85) probability distributions using Equation 28.55 and Equation 28.56 as follows.

$$\mu'_k(c) = \sum_{x=-\infty}^{\infty} f_X(x)\,(x - c)^k \tag{28.84}$$

$$\mu'_k(c) = \int_{-\infty}^{\infty} f_X(x)\,(x - c)^k\,dx \tag{28.85}$$

Definition 28.37 (Statistical Moment). The k^{th} statistical moment μ'_k of a random distribution is its k^{th} moment about zero, i. e., the expected value of its values raised to the k^{th} power.

$$\mu'_k = \mu'_k(0) = E\big[X^k\big] \tag{28.86}$$

Definition 28.38 (Central Moment). The k^{th} moment about the mean (or central moment)[34] is the expected value of the difference between elements and their expected value

[31] http://en.wikipedia.org/wiki/Coefficient_of_variation [accessed 2007-07-03]
[32] http://en.wikipedia.org/wiki/Covariance [accessed 2008-02-05]
[33] http://en.wikipedia.org/wiki/Moment_%28mathematics%29 [accessed 2008-02-01]
[34] http://en.wikipedia.org/wiki/Moment_about_the_mean [accessed 2007-07-03]

raised to the kth power.

$$\mu_k = E\Big[(X - EX)^k\Big] \tag{28.87}$$

Hence, the variance D^2X equals the s central moment μ_2.

Definition 28.39 (Standardized Moment). The kth standardized moment $\mu_{\sigma,k}$ is the quotient of the kth central moment and the standard deviation raised to the kth power.

$$\mu_{\sigma,k} = \frac{\mu_k}{\sigma^k} \tag{28.88}$$

28.3.5 Skewness and Kurtosis

The two other most important moments of random distributions are the skewness γ_1 and the kurtosis γ_2 and their estimates G_1 and G_2.

Definition 28.40 (Skewness). The skewness[35] γ_1, the third standardized moment, is a measure of asymmetry of a probability distribution. If $\gamma_1 > 0$, the right part of the distribution function is either longer or fatter (positive skew, right-skewed). If $\gamma_1 < 0$, the distribution's left part is longer or fatter.

$$\gamma_1 = \mu_{\sigma,3} = \frac{\mu_3}{\sigma^3} \tag{28.89}$$

For sample data A the skewness of the underlying random variable is approximated with the estimator G_1 where s is the estimated standard deviation. The sample skewness is only defined for sets A with at least three elements.

$$G_1 = \frac{n}{(n-1)(n-2)} \sum_{i=0}^{n-1} \left(\frac{A_{[i]} - \overline{a}}{s} \right)^3 \tag{28.90}$$

Definition 28.41 (Kurtosis). The excess kurtosis[36] γ_2 is a measure for the sharpness of a distribution's peak. A distribution with a high kurtosis has a sharper "peak" and fatter "tails", while a distribution with a low kurtosis has a more rounded peak with wider "shoulders". The normal distribution (see Section 28.5.2) has a zero kurtosis.

$$\gamma_2 = \mu_{\sigma,4} - 3 = \frac{\mu_4}{s^3} - 3 \tag{28.91}$$

For sample data A which represents only a subset of a greater amount of data, the sample kurtosis can be approximated with the estimator G_2 where s is the estimate of the sample's standard deviation. The kurtosis is only defined for sets with at least four elements.

$$G_2 = \left[\frac{n(n+1)}{(n-1)(n-2)(n-3)} \sum_{i=0}^{n-1} \left(\frac{A_{[i]} - \overline{a}}{s} \right)^4 \right] - \frac{3(n-1)^2}{(n-2)(n-3)} \tag{28.92}$$

28.3.6 Median, Quantiles, and Mode

Definition 28.42 (Median). The median $m = \text{med}(X)$ is the value right in the middle of a sample or distribution, dividing it into two equal halves. Therefore, the probability of drawing an element less than $\text{med}(X)$ is equal to the probability of drawing an element larger than m.

$$P(X \leq m) \geq \frac{1}{2} \wedge P(X \geq m) \geq \frac{1}{2} \wedge P(X \leq m) \leq P(X \geq m) \tag{28.93}$$

[35] http://en.wikipedia.org/wiki/Skewness [accessed 2007-07-03]
[36] http://en.wikipedia.org/wiki/Kurtosis [accessed 2008-02-01]

We can determine the median m of continuous and discrete distributions by solving Equation 28.94 and Equation 28.95 respectively.

$$\frac{1}{2} = \int_{-\infty}^{m} f_X(x)\,dx \tag{28.94}$$

$$\sum_{i=-\infty}^{m-1} f_X(x) \leq \frac{1}{2} \leq \sum_{i=m}^{\infty} f_X(x) \tag{28.95}$$

$$\tag{28.96}$$

If a sample A has an odd element count, the median m is the element in the middle, otherwise (in a set with an even element count there exists no single "middle"-element), the arithmetic mean of the two middle elements. The median represents the dataset in an unbiased manner. If you have, for example, the dataset $A = (1, 1, 1, 1, 1, 2, 2, 2, 500\,000)$, the arithmetic mean, biased by the large element 500 000 would be very high ($55556.\overline{7}$). The median however would be 1 and thus represents the sample better. The median of a sample can be computed as:

$$A_s \equiv \text{sortList}_a(A, >) \tag{28.97}$$

$$\text{med}(A) = \begin{cases} A_s\left[\frac{n-1}{2}\right] & \text{if } n = \text{len}(A) \textit{ is odd} \\ \frac{1}{2}\left(A_s\left[\frac{n}{2}\right] + A_s\left[\frac{n}{2}-1\right]\right) & \text{otherwise} \end{cases} \tag{28.98}$$

Definition 28.43 (Quantile). Quantiles[37] are points taken at regular intervals from a sorted dataset (or a cumulative distribution function). The q-quantiles divide a distribution function F_X or data sample A into q parts T_i with equal probability. They can be regarded as the generalized median, or vice versa, the median is the 2-quantile.

$$\forall\, x \in \mathbb{R}, i \in [0, q-1] \Rightarrow \frac{1}{q} \leq P(x \in T_i) \tag{28.99}$$

A sorted data sample is divided into q subsets of equal length by the q-quantiles. The cumulative distribution function of a random variable X is divided by the q-quantiles into q subsets of equal area. The quantiles are the boundaries between the subsets/areas. Therefore, the kth q-quantile is the value ζ so that the probability that the random variable (or an element of the data set) will take on a value less than ζ is at most $\frac{k}{q}$ and the probability that it will take on a value greater than or equal to ζ is at most $\frac{q-k}{q}$. There exist $q-1$ q-quantiles (k spans from 1 to $q-1$). The kth q-quantile $\text{quantile}_q^k(A)$ of a dataset A can be computed as:

$$A_s \equiv \text{sortList}_a(A, >) \tag{28.100}$$

$$\text{quantile}_q^k(A) = A_s[\lfloor \tfrac{k*n}{q} \rfloor] \tag{28.101}$$

For some special values of q, the quantiles have been given special names too (see Table 28.1).

Definition 28.44 (Interquartile Range). The interquartile range[38] is the range between the first and the third quartile and defined as $quantile_4^3(X) - quantile_4^1(X)$.

Definition 28.45 (Mode). The mode[39] is the value that most often occurs in a data sample or is most frequently assumed by a random variable. There exist unimodal distributions/samples that have one mode value and multimodal distributions/samples with multiple modes.

In [2119, 258] you can find further information of the relation between the mode, the mean and the skewness.

[37] http://en.wikipedia.org/wiki/Quantiles [accessed 2007-07-03]
[38] http://en.wikipedia.org/wiki/Inter-quartile_range [accessed 2007-07-03]
[39] http://en.wikipedia.org/wiki/Mode_%28statistics%29 [accessed 2007-07-03]

q	name
100	percentiles
10	deciles
9	noniles
5	quintiles
4	quartiles
2	median

Table 28.1: Special Quantiles

28.3.7 Entropy

Definition 28.46 (Information Entropy). The information entropy[40] $\mathrm{H}(X)$defined by Shannon [1858] is a measure of uncertainty for discrete probability mass functions f_X of random variables X or data sets A. It is defined in as follows. The $h(a,n)$ in Equation 28.103 denotes the relative frequency of the value a amongst the n samples in A.

$$\mathrm{H}(X) = \sum_{x=-\infty}^{\infty} f_X(x) \log_2 \left(\frac{1}{f_X(y)} \right) = -\sum_{x-\infty}^{\infty} f_X(x) \log_2 f_X(x) \tag{28.102}$$

$$\mathrm{H}(A) = -\sum_{\forall a \in A} h(a,n) \log_2 h(a,n) \tag{28.103}$$

Definition 28.47 (Differential Entropy). The differential (also called continuous) entropy $\mathrm{h}(X)$ is a generalization of the information entropy to continuous probability density functions f_X of random variables X. [1266]

$$\mathrm{h}(X) = -\int_{-\infty}^{\infty} f_X(x) \ln f_X(x)\, dx \tag{28.104}$$

28.3.8 The Law of Large Numbers

The law of large numbers (LLN) combines statistics and probability by showing that if an event e with the probability $P(e) = p$ is observed in n independent repetitions of a random experiment, its relative frequency $h(e,n)$ (see Definition 28.10) converges to its probability p if n becomes larger.

In the following, assume that the A is an infinite sequence of samples from equally distributed and pairwise independent random variables X_i with the (same) expected value EX. The weak law of large numbers states that the mean $\overline{a}$ of the sequence A converges to a value in $(EX - \varepsilon, EX + \varepsilon)$ for each positive real number $\varepsilon > 0, \varepsilon \in \mathbb{R}^+$.

$$\lim_{n\to\infty} P(|\overline{a} - EX| < \varepsilon) = 1 \tag{28.105}$$

In other words, the weak law of large numbers says that the sample average will converge to the expected value of the random experiment if the experiment is repeated many times.

According to the strong law of large numbers, the mean $\overline{a}$ of the sequence A even converges to the expected value EX of the underlying distribution for infinite large n

$$P\Big(\lim_{n\to\infty} \overline{a} = EX \Big) = 1 \tag{28.106}$$

The law of large numbers implies that the accumulated results of each random experiment will approximate the underlying distribution function if repeated infinitely (under the condition that there exists an invariable underlying distribution function).

[40] http://en.wikipedia.org/wiki/Information_entropy [accessed 2007-07-03]

28.4 Some Discrete Distributions

In this section we will introduce some common discrete distributions. Discrete probability distributions assign probabilities to the elements of a finite (or, at most, countable infinite) set of discrete events/outcomes of a random experiment.

Parts of the information provided in this and the following section have been obtained from Wikipedia [2219].

28.4.1 Discrete Uniform Distribution

The uniform distribution exists in a discrete[41] as well as in a continuous form. In this section we want to discuss the discrete form whereas the continuous form is elaborated on in Section 28.4.1.

All possible outcomes $\omega \in \Omega$ of a random experiment which obeys the uniform distribution have exactly the same probability. In the discrete uniform distribution, Ω has at most countable infinite elements (although normally being finite). The best example for this distribution is throwing an ideal dice. This experiment has six possible outcomes ω_i where each has the same probability $P(\omega_i) = \frac{1}{6}$. Throwing ideal coins and drawing one element out of a set of n possible elements are other examples where a discrete uniform distribution can be assumed. Table 28.2 contains the characteristics of the discrete uniform distribution. In Figure 28.1 you can find some example uniform probability mass functions and in Figure 28.2 we have outlined their according cumulative distribution functions.

parameter	definition	
parameters	$a, b \in \mathbb{Z}, a > b$	(28.107)
$\lvert\Omega\rvert$	$\lvert\Omega\rvert = r = \text{range} = b - a + 1$	(28.108)
PMF	$P(X = x) = f_X(x) = \begin{cases} \frac{1}{r} & \text{if } a \leq x \leq b, x \in \mathbb{Z} \\ 0 & \text{otherwise} \end{cases}$	(28.109)
CDF	$P(X \leq x) = F_X(x) = \begin{cases} 0 & \text{if } x < a \\ \lfloor \frac{x-a+1}{r} \rfloor & \text{if } a \leq x \leq b \\ 1 & \text{otherwise} \end{cases}$	(28.110)
mean	$EX = \frac{a+b}{2}$	(28.111)
median	$\text{med} = \frac{a+b}{2}$	(28.112)
mode	$\text{mode} = \emptyset$	(28.113)
variance	$D^2X = \frac{r^2-1}{12}$	(28.114)
skewness	$\gamma_1 = 0$	(28.115)
kurtosis	$\gamma_2 = -\frac{6(r^2+1)}{5(r^2-1)}$	(28.116)
entropy	$\mathrm{H}(X) = \ln r$	(28.117)
mgf	$M_X(t) = \frac{e^{at} - e^{(b+1)t}}{r(1-e^t)}$	(28.118)
char. func.	$\varphi_X(t) = \frac{e^{\mathrm{i}at} - e^{\mathrm{i}(b+1)t}}{r(1-e^{\mathrm{i}t})}$	(28.119)

Table 28.2: Parameters of the discrete uniform distribution.

[41] http://en.wikipedia.org/wiki/Uniform_distribution_%28discrete%29 [accessed 2007-07-03]

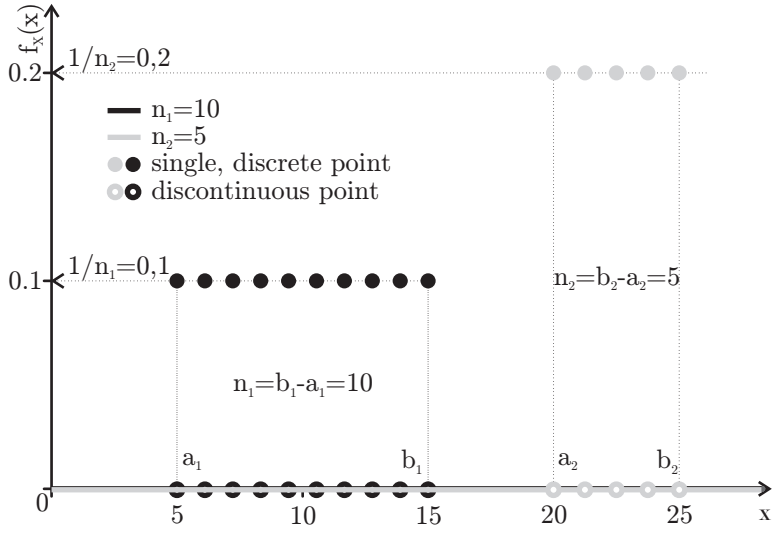


Figure 28.1: The PMFs of some discrete uniform distributions

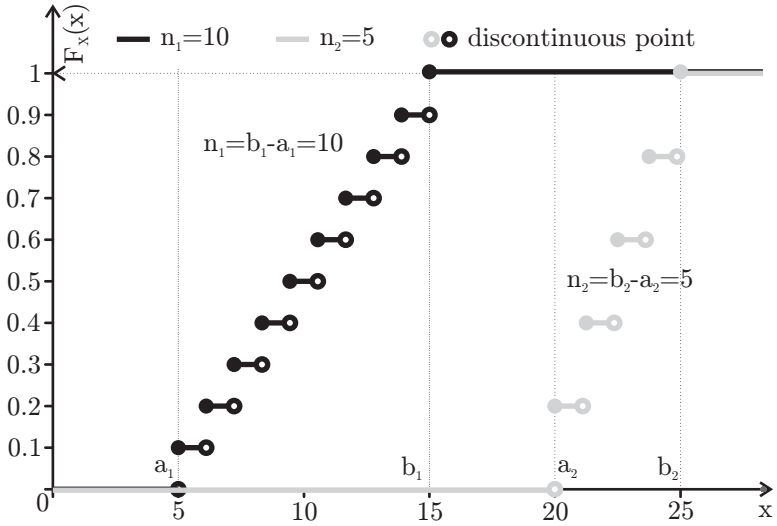


Figure 28.2: The CDFs of some discrete uniform distributions

28.4.2 Poisson Distribution π_λ

The Poisson distribution[42] π_λ [20] complies with the reference model *telephone switchboard*. It describes a process where the number of events that occur (independently of each other) in a certain time interval only depends on the duration of the interval and not of its position (*prehistory*). Events do not have any aftermath and thus, there is no mutual influence of non-overlapping time intervals (*homogeneity*). Furthermore, only the time when an even occurs is considered and not the duration of the event. In the telephone switchboard example, we would only be interested in the time at which a call comes in, not in the length of the call. In this model, no events occur in infinitely short time intervals. The features of the Poisson

[42] http://en.wikipedia.org/wiki/Poisson_distribution [accessed 2007-07-03]

distribution are listed in Table 28.3[43] and examples for its PDF and CDF are illustrated in Figure 28.3 and Figure 28.4.

parameter	**definition**	
parameters	$\lambda = \mu t > 0$	(28.120)
PMF	$P(X = x) = f_X(x) = \frac{(\mu t)^x}{x!} e^{-\mu t} = \frac{\lambda^x}{x!} e^{-\lambda}$	(28.121)
CDF	$P(X \leq x) = F_X(x) = \frac{\Gamma(\lfloor k+1 \rfloor, \lambda)}{\lfloor k \rfloor !} = \sum_{i=0}^{x} \frac{e^{-\lambda} \lambda^i}{i!!}$	(28.122)
mean	$EX = \mu t = \lambda$	(28.123)
median	$\text{med} \approx \lfloor \lambda + \frac{1}{3} - \frac{1}{5\lambda} \rfloor$	(28.124)
mode	$\text{mode} = \lfloor \lambda \rfloor$	(28.125)
variance	$D^2 X = \mu t = \lambda$	(28.126)
skewness	$\gamma_1 = \lambda^{-\frac{1}{2}}$	(28.127)
kurtosis	$\gamma_2 = \frac{1}{\lambda}$	(28.128)
entropy	$\mathrm{H}(X) = \lambda (1 - \ln \lambda) + e^{-\lambda} \sum_{k=0}^{\infty} \frac{\lambda^k \ln(k!)}{k!}$	(28.129)
mgf	$M_X(t) = e^{\lambda(e^t - 1)}$	(28.130)
char. func.	$\varphi_X(t) = e^{\lambda(e^{it} - 1)}$	(28.131)

Table 28.3: Parameters of the Poisson distribution.

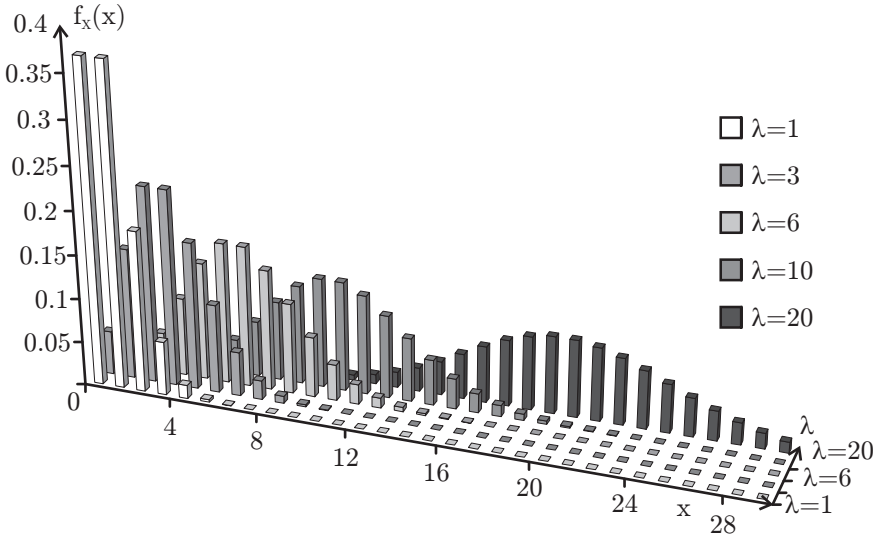


Figure 28.3: The PMFs of some Poisson distributions

Poisson Process

The Poisson process[44] [1914] is a process that obeys the Poisson distribution – just like the example of the telephone switchboard mentioned before. Here, λ is expressed as the product of the intensity μ and the time t. μ normally describes a frequency, for example $\mu = \frac{1}{\text{min}}$. Both, the expected value as well as the variance of the Poisson process are $\lambda = \mu t$. In

[43] The Γ in Equation 28.122 denotes the (upper) incomplete gamma function. More information on the gamma function Γ can be found in Section 28.10.1 on page 532.

[44] http://en.wikipedia.org/wiki/Poisson_process [accessed 2007-07-03]

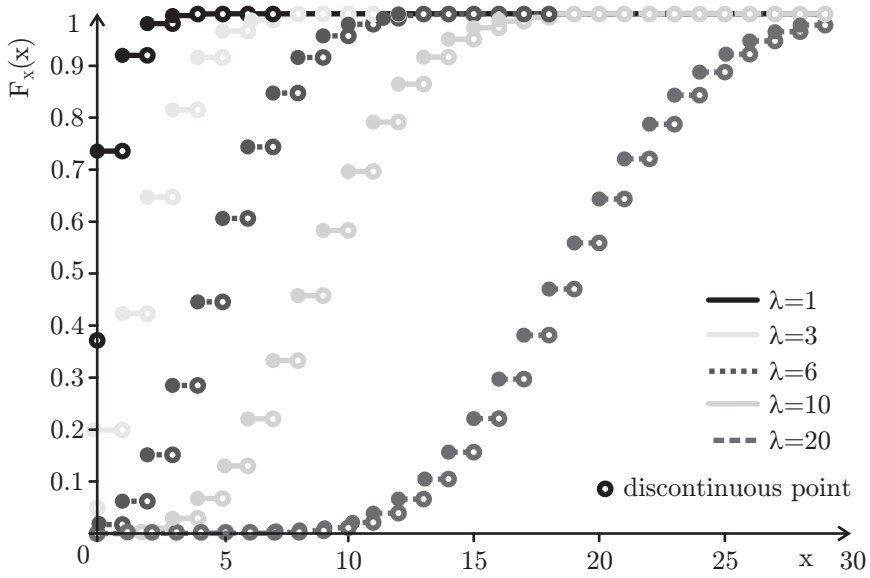


Figure 28.4: The CDFs of some Poisson distributions

Equation 28.132, the probability that k events occur in a Poisson process in a time interval of the length t is defined.

$$P(X_t = k) = \frac{(\mu t)^k}{k!} e^{-\mu t} = \frac{\lambda^k}{k!} e^{-\lambda} \tag{28.132}$$

The probability that in a time interval $[t, t + \Delta t]$

1. no events occur is $1 - \lambda \Delta t + \mathbf{o}(\Delta t)$.
2. exactly one event occurs is $\lambda \Delta t + \mathbf{o}(\Delta t)$.
3. multiple events occur $\mathbf{o}(\Delta t)$.

Here we use an infinitesimal version the small-**o** notation. [45] The statement that $f \in \mathbf{o}(\xi) \Rightarrow |f(x)| \ll |\xi(x)|$ is normally only valid for $x \to \infty$. In the infinitesimal variant, it holds for $x \to 0$. Thus, we can state that $\mathbf{o}(\Delta t)$ is much smaller than Δt. In principle, the above equations imply that in an infinite small time span either no or one event occurs, i. e., events do not arrive simultaneously:

$$\lim_{t \to 0} P(X_t > 1) = 0 \tag{28.133}$$

The Relation between the Poisson Process and the Exponential Distribution

It is important to know that the (time) distance between two events of the Poisson process is exponentially distributed (see Section 28.5.3 on page 489). The expected value of the number of events to arrive per time unit in a Poisson process is EX_{pois}, then the expected value of the time between two events $\frac{1}{EX_{pois}}$. Since this is the excepted value $EX_{exp} = \frac{1}{EX_{pois}}$ of the exponential distribution, its λ_{exp}-value is $\lambda_{exp} = \frac{1}{EX_{exp}} = \frac{1}{\frac{1}{EX_{pois}}} = EX_{pois}$. Therefore, the λ_{exp}-value of the exponential distribution equals the λ_{pois}-value of the Poisson distribution $\lambda_{exp} = \lambda_{pois} = EX_{pois}$. In other words, the time interval between (neighboring) events of the

[45] See Section 30.1.3 on page 550 and Definition 30.16 on page 551 for a detailed elaboration on the small-**o** notation.

Poisson process is exponentially distributed with the same λ value as the Poisson process, as illustrated in Equation 28.134.

$$X_i \sim \pi_\lambda \Leftrightarrow (t(X_{i+1}) - tX_i) \sim exp(\lambda) \ \forall i \in \mathbb{N} \tag{28.134}$$

28.4.3 Binomial Distribution B(n, p)

The binomial distribution[46] $B(n,p)$ is the probability distribution that describes the probability of the possible numbers successes of n independent experiments with the success probability p each. Such experiments is called Bernoulli experiments or Bernoulli trials. For $n = 1$, the binomial distribution is a Bernoulli distribution[47].

Table 28.4[48] points out some of the properties of the binomial distribution. A few examples for PMFs and CDFs of different binomial distributions are given in Figure 28.5 and Figure 28.6.

parameter	definition	
parameters	$n \in \mathbb{N}_0, 0 \leq p \leq 1,\ p \in \mathbb{R}$	(28.135)
PMF	$P(X = x) = f_X(x) = \binom{n}{x} p^x (1-p)^{n-x}$	(28.136)
CDF	$P(X \leq x) = F_X(x) = \sum_{i=0}^{\lfloor x \rfloor} f_X(x) = I_{1-p}(n - \lfloor x \rfloor, 1 + \lfloor x \rfloor)$	(28.137)
mean	$EX = np$	(28.138)
median	med *is one of* $\{\lfloor np \rfloor - 1, \lfloor np \rfloor, \lfloor np \rfloor + 1\}$	(28.139)
mode	mode $= \lfloor (n+1)p \rfloor$	(28.140)
variance	$D^2X = np(1-p)$	(28.141)
skewness	$\gamma_1 = \frac{1-2p}{\sqrt{np(1-p)}}$	(28.142)
kurtosis	$\gamma_2 = \frac{1-6p(1-p)}{np(1-p)}$	(28.143)
entropy	$\mathrm{H}(X) = \frac{1}{2} \ln (2\pi n\, e\, p\, (1-p)) + \mathbf{O}\left(\frac{1}{n}\right)$	(28.144)
mgf	$M_X(t) = (1 - p + pe^t)^n$	(28.145)
char. func.	$\varphi_X(t) = (1 - p + pe^{it})^n$	(28.146)

Table 28.4: Parameters of the Binomial distribution.

For $n \to \infty$, the binomial distribution approaches a normal distribution. For large n, $B(n,p)$ can therefore often be approximated with the normal distribution (see Section 28.5.2) $N(np, np(1-p))$. Whether this approximation is good or not can be found out by rules of thumb, some of them are:

$$np > 5 \wedge n(1-p) > 5$$
$$\mu \pm 3\sigma \approx np \pm 3\sqrt{np(1-p)} \in [0, n]$$

In case these rules hold, we still need to transform a continuous distribution to a discrete one. In order to do so, we add 0.5 to the x values, i. e., $F_{X,bin}(x) \approx F_{X,normal}(x + 0.5)$.

[46] http://en.wikipedia.org/wiki/Binomial_distribution [accessed 2007-10-01]
[47] http://en.wikipedia.org/wiki/Bernoulli_distribution [accessed 2007-10-01]
[48] I_{1-p} in Equation 28.137 denotes the regularized incomplete beta function.

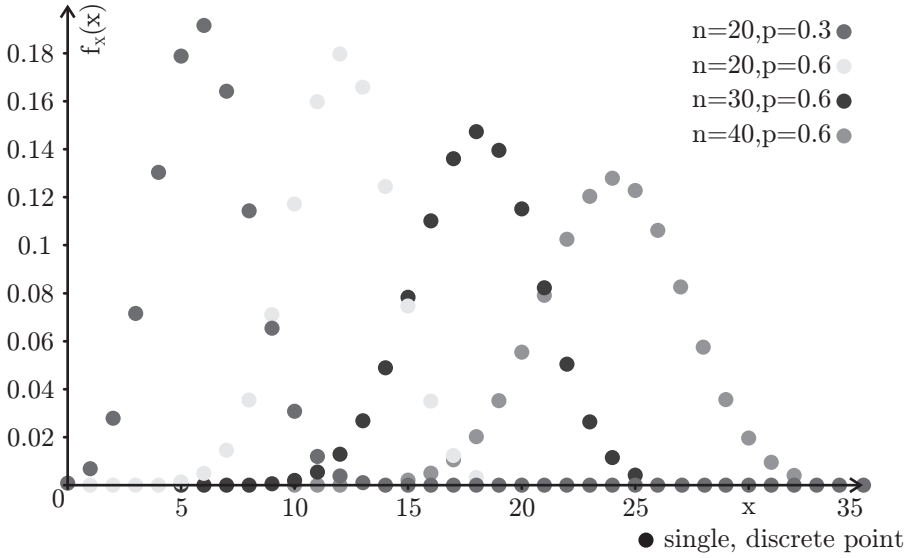


Figure 28.5: The PMFs of some binomial distributions

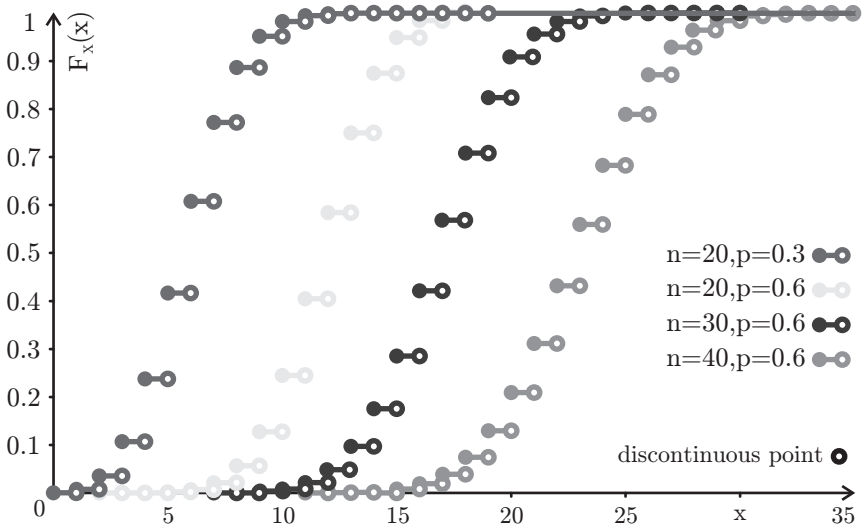


Figure 28.6: The CDFs of some binomial distributions

28.5 Some Continuous Distributions

In this section we will introduce some common continuous distributions. Unlike the discrete distributions, continuous distributions have an uncountable infinite large set of possible outcomes of random experiments. Thus, the PDF does not assign probabilities to certain events. Only the CDF makes statements about the probability of a sub-set of possible outcomes of a random experiment.

28.5.1 Continuous Uniform Distribution

After discussing the discrete uniform distribution in Section 28.4.1, we now elaborate on its continuous form[49].

In a uniform distribution, all possible outcomes in a range $[a, b], b > a$ have exactly the same probability. The characteristics of this distribution can be found in Table 28.5. Examples of its probability density function is illustrated in Figure 28.7 whereas the according cumulative density functions are outlined Figure 28.8.

parameter	definition	
parameters	$a, b \in \mathbb{R}, a \geq b$	(28.147)
PDF	$f_X(x) = \begin{cases} \frac{1}{b-a} & \text{if } x \in [a, b] \\ 0 & \text{otherwise} \end{cases}$	(28.148)
CDF	$P(X \leq x) = F_X(x) = \begin{cases} 0 & \text{if } x < a \\ \frac{x-a}{b-a} & \text{if } x \in [a, b] \\ 1 & \text{otherwise} \end{cases}$	(28.149)
mean	$EX = \frac{1}{2}(a + b)$	(28.150)
median	$\text{med} = \frac{1}{2}(a + b)$	(28.151)
mode	$\text{mode} = \emptyset$	(28.152)
variance	$D^2X = \frac{1}{12}(b - a)^2$	(28.153)
skewness	$\gamma_1 = 0$	(28.154)
kurtosis	$\gamma_2 = -\frac{6}{5}$	(28.155)
entropy	$\text{h}(X) = \ln(b - a)$	(28.156)
mgf	$M_X(t) = \frac{e^{tb} - e^{ta}}{t(b-a)}$	(28.157)
char. func.	$\varphi_X(t) = \frac{e^{itb} - e^{ita}}{it(b-a)}$	(28.158)

Table 28.5: Parameters of the continuous uniform distribution.

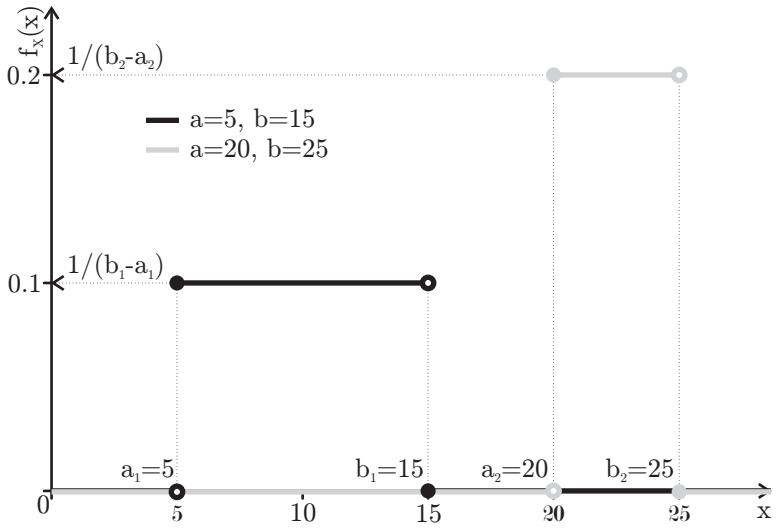


Figure 28.7: The PDFs of some continuous uniform distributions

[49] http://en.wikipedia.org/wiki/Uniform_distribution_%28continuous%29 [accessed 2007-07-03]

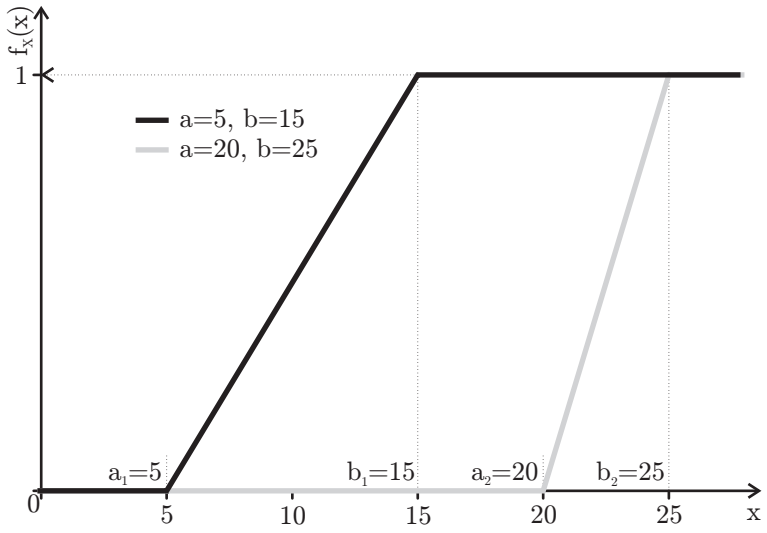


Figure 28.8: The CDFs of some continuous uniform distributions

28.5.2 Normal Distribution $\mathbf{N}(\mu, \sigma^2)$

Many phenomena in nature, like the size of chicken eggs, noise, errors in measurement, and such and such, can be considered as outcomes of random experiments with properties that can be approximated by the normal distribution[50] $N(\mu, \sigma^2)$ [2312]. Its probability density function, shown for some example values in Figure 28.9, is symmetric to the expected value μ and becomes flatter with rising standard deviation σ. The cumulative density function is outline for the same example values in Figure 28.10. Other characteristics of the normal distribution can be found in Table 28.6.

parameter	definition	
parameters	$\mu \in \mathbb{R}, \sigma \in \mathbb{R}^+$	(28.159)
PDF	$f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$	(28.160)
CDF	$P(X \leq x) = F_X(x) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{(z-\mu)^2}{2\sigma^2}} dz$	(28.161)
mean	$EX = \mu$	(28.162)
median	$\text{med} = \mu$	(28.163)
mode	$\text{mode} = \mu$	(28.164)
variance	$D^2X = \sigma^2$	(28.165)
skewness	$\gamma_1 = 0$	(28.166)
kurtosis	$\gamma_2 = 0$	(28.167)
entropy	$\text{h}(X) = \ln\left(\sigma\sqrt{2\pi e}\right)$	(28.168)
mgf	$M_X(t) = e^{\mu t + \frac{\sigma^2 t^2}{2}}$	(28.169)
char. func.	$\varphi_X(t) = e^{\mu i t + \frac{\sigma^2 t^2}{2}}$	(28.170)

Table 28.6: Parameters of the normal distribution.

Definition 28.48 (Standard Normal Distribution).

For the sake of simplicity, the standard normal distribution $N(0, 1)$ with the CDF $\Phi(x)$ is defined with $\mu = 0$ and $\sigma = 1$. Values of this function are listed in tables. You can compute

[50] http://en.wikipedia.org/wiki/Normal_distribution [accessed 2007-07-03]

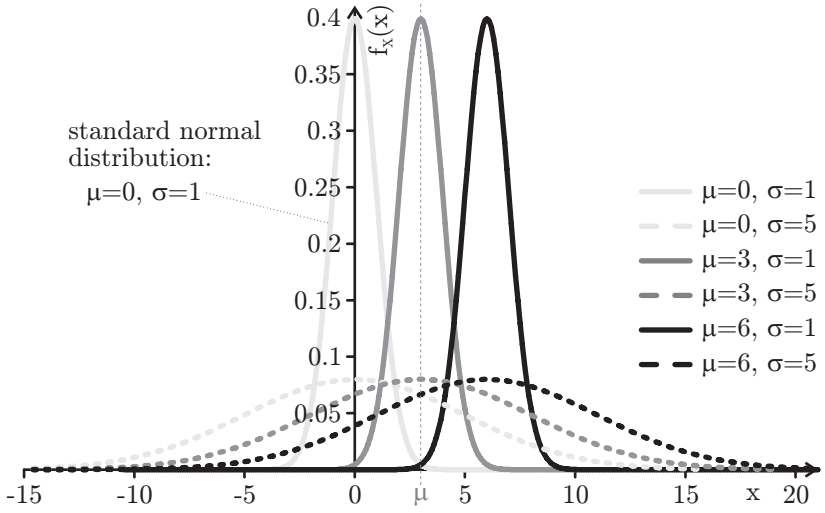


Figure 28.9: The PDFs of some normal distributions

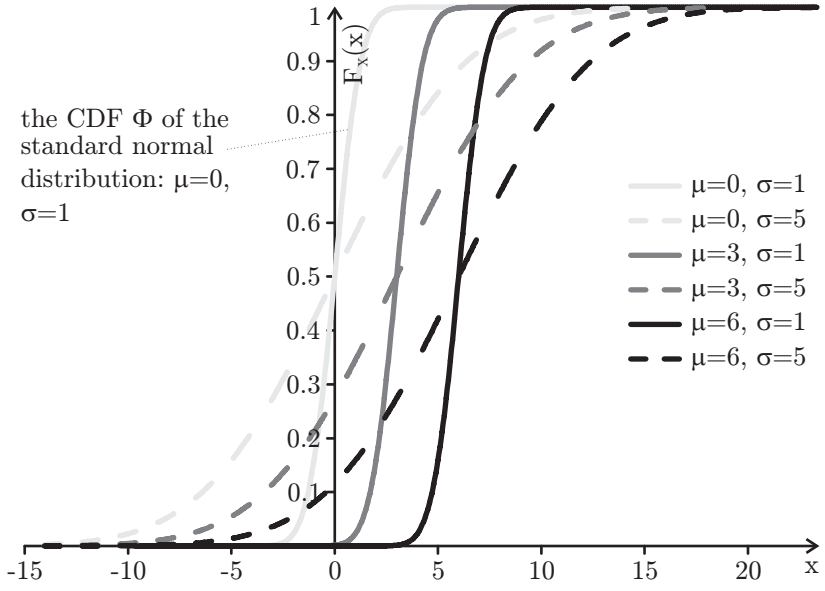


Figure 28.10: The CDFs of some normal distributions

the CDF of any normal distribution using the one of the standard normal distribution by applying Equation 28.171.

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{z^2}{2}} dz \tag{28.171}$$

$$P(X \leq x) = \Phi\left(\frac{x-\mu}{\sigma}\right) \tag{28.172}$$

Some values of $\Phi(x)$ are listed in Table 28.7. For the sake of saving space by using two dimensions, we compose the values of x as a sum of a row and column value. If you want to look up $\Phi(2.13)$ for example, you'd go to the row which starts with 2.1 and the column of 0.03, so you'd find $\Phi(2.13) \approx 0.9834$.

x	0.00	0.01	0.02	0.03	0.04	0.05	0.06	0.07	0.08	0.09
0.0	0.5000	0.5040	0.5080	0.5120	0.5160	0.5199	0.5239	0.5279	0.5319	0.5359
0.1	0.5398	0.5438	0.5478	0.5517	0.5557	0.5596	0.5636	0.5675	0.5714	0.5753
0.2	0.5793	0.5832	0.5871	0.5910	0.5948	0.5987	0.6026	0.6064	0.6103	0.6141
0.3	0.6179	0.6217	0.6255	0.6293	0.6331	0.6368	0.6406	0.6443	0.6480	0.6517
0.4	0.6554	0.6591	0.6628	0.6664	0.6700	0.6736	0.6772	0.6808	0.6844	0.6879
0.5	0.6915	0.6950	0.6985	0.7019	0.7054	0.7088	0.7123	0.7157	0.7190	0.7224
0.6	0.7257	0.7291	0.7324	0.7357	0.7389	0.7422	0.7454	0.7486	0.7517	0.7549
0.7	0.7580	0.7611	0.7642	0.7673	0.7704	0.7734	0.7764	0.7794	0.7823	0.7852
0.8	0.7881	0.7910	0.7939	0.7967	0.7995	0.8023	0.8051	0.8078	0.8106	0.8133
0.9	0.8159	0.8186	0.8212	0.8238	0.8264	0.8289	0.8315	0.8340	0.8365	0.8389
1.0	0.8413	0.8438	0.8461	0.8485	0.8508	0.8531	0.8554	0.8577	0.8599	0.8621
1.1	0.8643	0.8665	0.8686	0.8708	0.8729	0.8749	0.8770	0.8790	0.8810	0.8830
1.2	0.8849	0.8869	0.8888	0.8907	0.8925	0.8944	0.8962	0.8980	0.8997	0.9015
1.3	0.9032	0.9049	0.9066	0.9082	0.9099	0.9115	0.9131	0.9147	0.9162	0.9177
1.4	0.9192	0.9207	0.9222	0.9236	0.9251	0.9265	0.9279	0.9292	0.9306	0.9319
1.5	0.9332	0.9345	0.9357	0.9370	0.9382	0.9394	0.9406	0.9418	0.9429	0.9441
1.6	0.9452	0.9463	0.9474	0.9484	0.9495	0.9505	0.9515	0.9525	0.9535	0.9545
1.7	0.9554	0.9564	0.9573	0.9582	0.9591	0.9599	0.9608	0.9616	0.9625	0.9633
1.8	0.9641	0.9649	0.9656	0.9664	0.9671	0.9678	0.9686	0.9693	0.9699	0.9706
1.9	0.9713	0.9719	0.9726	0.9732	0.9738	0.9744	0.9750	0.9756	0.9761	0.9767
2.0	0.9772	0.9778	0.9783	0.9788	0.9793	0.9798	0.9803	0.9808	0.9812	0.9817
2.1	0.9821	0.9826	0.9830	0.9834	0.9838	0.9842	0.9846	0.9850	0.9854	0.9857
2.2	0.9861	0.9864	0.9868	0.9871	0.9875	0.9878	0.9881	0.9884	0.9887	0.9890
2.3	0.9893	0.9896	0.9898	0.9901	0.9904	0.9906	0.9909	0.9911	0.9913	0.9916
2.4	0.9918	0.9920	0.9922	0.9925	0.9927	0.9929	0.9931	0.9932	0.9934	0.9936
2.5	0.9938	0.9940	0.9941	0.9943	0.9945	0.9946	0.9948	0.9949	0.9951	0.9952
2.6	0.9953	0.9955	0.9956	0.9957	0.9959	0.9960	0.9961	0.9962	0.9963	0.9964
2.7	0.9965	0.9966	0.9967	0.9968	0.9969	0.9970	0.9971	0.9972	0.9973	0.9974
2.8	0.9974	0.9975	0.9976	0.9977	0.9977	0.9978	0.9979	0.9979	0.9980	0.9981
2.9	0.9981	0.9982	0.9982	0.9983	0.9984	0.9984	0.9985	0.9985	0.9986	0.9986
3.0	0.9987	0.9987	0.9987	0.9988	0.9988	0.9989	0.9989	0.9989	0.9990	0.9990

Table 28.7: Some values of the standardized normal distribution.

Definition 28.49 (probit**).** The inverse of the cumulative distribution function of the standard normal distribution is called the probit function. It is also often denoted as z-quantile of the standard normal distribution.

$$z(y) \equiv \text{probit}(y) \equiv \Phi^{-1}(y) \tag{28.173}$$

$$y = \Phi(x) \Rightarrow \Phi^{-1}(y) = z(y) = x \tag{28.174}$$

The values of the quantiles of the standard normal distribution can also be looked up in Table 28.7. Therefore, the previously discussed process is simply reversed. If we wanted to find the value $z(0.922)$, we locate the closest match in the table. In Table 28.7, we will find 0.9222 which leads us to $x = 1.4 + 0.02$. Hence, $z(0.922) \approx 1.42$.

The probability density function PDF of the multivariate normal distribution[51] [2005, 1899, 1772] is illustrated in Equation 28.175 and Equation 28.176 in the general case (where $\mathbf{\Sigma}$ is the covariance matrix) and in Equation 28.177 in the uncorrelated form. If the distributions, additionally to being uncorrelated, also have the same parameters σ and μ, the probability density function of the multivariate normal distribution can be expressed as it is done in Equation 28.178.

[51] http://en.wikipedia.org/wiki/Multivariate_normal_distribution [accessed 2007-07-03]

$$f_X(\mathbf{x}) = \frac{\sqrt{\mathbf{\Sigma}^{-1}}}{(2\pi)^{\frac{n}{2}}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})} \tag{28.175}$$

$$= \frac{1}{(2\pi)^{\frac{n}{2}} \mathbf{\Sigma}^{\frac{1}{2}}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})} \tag{28.176}$$

$$f_X(\mathbf{x}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_i} e^{-\frac{(x_i-\mu_i)^2}{2\sigma_i^2}} \tag{28.177}$$

$$f_X(\mathbf{x}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x_i-\mu_i)^2}{2\sigma^2}}$$

$$= \left(\frac{1}{2\pi\sigma^2}\right)^{\frac{n}{2}} e^{-\frac{\sum_{i=1}^{n}(x_i-\mu)^2}{2\sigma^2}} \tag{28.178}$$

Definition 28.50 (Central Limit Theorem). The *central limit theorem*[52] (CLT) states that the sum $S_n = \sum_{i=1}^{n} X_i$ of i identically distributed random variables X_i with finite expected values $E[X_i]$ and non-zero variances $D^2[X_i] > 0$ approaches a normal distribution for $n \to +\infty$. [675, 1084, 2041]

28.5.3 Exponential Distribution exp(λ)

The exponential distribution[53] $exp(\lambda)$ [556] is often used if the probabilities of lifetimes of apparatuses, half-life periods of radioactive elements, or the time between two events in the Poisson process (see Section 28.4.2 on page 482) has to be approximated. Its PDF is sketched in Figure 28.11 for some example values of λ the according cases of the CDF are illustrated Figure 28.12. The most important characteristics of the exponential distribution can be obtained from Table 28.8.

parameter	definition	
parameters	$\lambda \in \mathbb{R}^+$	(28.179)
PDF	$f_X(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ \lambda e^{-\lambda x} & \text{otherwise} \end{cases}$	(28.180)
CDF	$P(X \leq x) = F_X(x) = \begin{cases} 0 & \text{if } x \leq 0 \\ 1 - e^{-\lambda x} & \text{otherwise} \end{cases}$	(28.181)
mean	$EX = \frac{1}{\lambda}$	(28.182)
median	$\text{med} = \frac{\ln 2}{\lambda}$	(28.183)
mode	$\text{mode} = 0$	(28.184)
variance	$D^2X = \frac{1}{\lambda^2}$	(28.185)
skewness	$\gamma_1 = 2$	(28.186)
kurtosis	$\gamma_2 = 6$	(28.187)
entropy	$\text{h}(X) = 1 - \ln\lambda$	(28.188)
mgf	$M_X(t) = \left(1 - \frac{t}{\lambda}\right)^{-1}$	(28.189)
char. func.	$\varphi_X(t) = \left(1 - \frac{\text{i}t}{\lambda}\right)^{-1}$	(28.190)

Table 28.8: Parameters of the exponential distribution.

[52] http://en.wikipedia.org/wiki/Central_limit_theorem [accessed 2008-08-19]
[53] http://en.wikipedia.org/wiki/Exponential_distribution [accessed 2007-07-03]

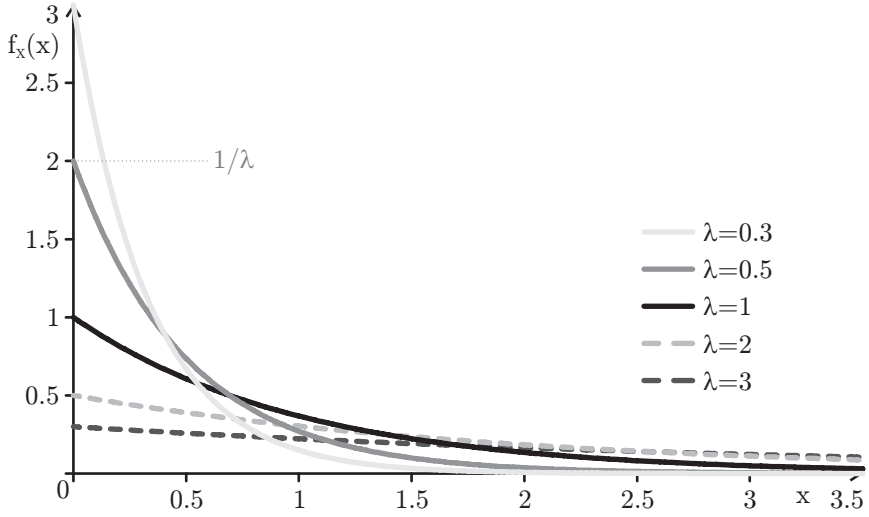


Figure 28.11: The PDFs of some exponential distributions

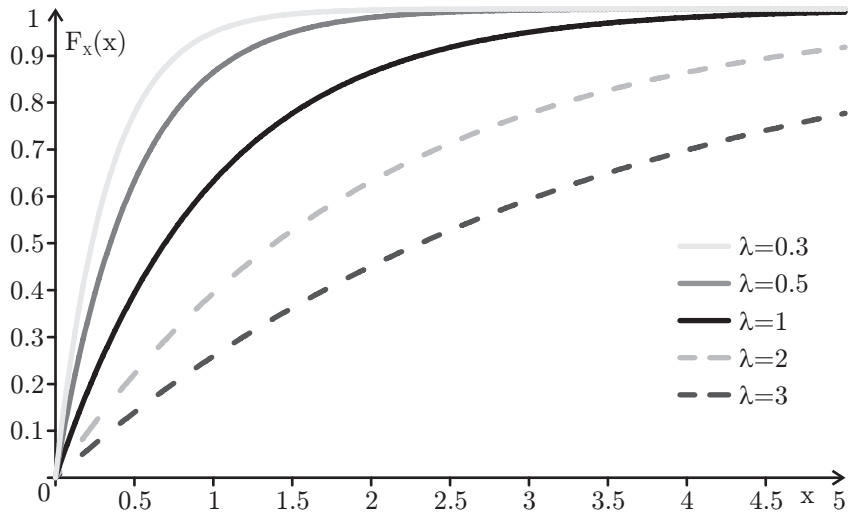


Figure 28.12: The CDFs of some exponential distributions

28.5.4 Chi-square Distribution

The chi-square (or χ^2) distribution[54] is a steady probability distribution on the set of positive real numbers. It is a so-called *sample distribution* which is used for the estimation of parameters like the variance of other distributions. We can also describe the sum of independent standardized normal distributions with it. Its sole parameter, n, denotes the degrees of freedom.

In Table 28.9[55], the characteristic parameters of the χ^2 distribution are outlined. A few examples for the PDF and CDF of the χ^2 distribution are illustrated in Figure 28.13 and Figure 28.14.

[54] http://en.wikipedia.org/wiki/Chi-square_distribution [accessed 2007-09-30]

[55] $\gamma(n, z)$ in Equation 28.193 is the lower incomplete Gamma function and $P_\gamma(n, z)$ is the regularized Gamma function.

Table 28.10 provides some selected values of the χ^2 distribution. The table's headline contains results of the cumulative distribution function $F_X(x)$ of a χ^2 distribution with n degrees of freedom (values in the first column). The cells now denote the x values that belong to these $(m, F_X(x))$ combinations.

parameter	**definition**	
parameters	$n \in \mathbb{R}^+,\ n > 0$	(28.191)
PDF	$f_X(x) = \begin{cases} 0 \text{ if } x \leq 0 \\ \frac{2^{-n/2}}{\Gamma(n/2)} x^{n/2-1} e^{-x/2} \text{ otherwise} \end{cases}$	(28.192)
CDF	$P(X \leq x) = F_X(x) = \frac{\gamma(n/2, x/2)}{\Gamma(n/2)} = P_\gamma(n/2, x/2)$	(28.193)
mean	$EX = n$	(28.194)
median	$\text{med} \approx n - \frac{2}{3}$	(28.195)
mode	$\text{mode} = n - 2$ *if* $n \geq 2$	(28.196)
variance	$D^2X = 2n$	(28.197)
skewness	$\gamma_1 = \sqrt{\frac{8}{n}}$	(28.198)
kurtosis	$\gamma_2 = \frac{12}{n}$	(28.199)
entropy	$\text{h}(X) = \frac{n}{2} + \ln(2\Gamma(n/2)) + (1 - n/2)\psi(n/2)$	(28.200)
mgf	$M_X(t) = (1 - 2t)^{-n/2}$ *for* $2t < 1$	(28.201)
char. func.	$\varphi_X(t) = (1 - 2\text{i}t)^{-n/2}$	(28.202)

Table 28.9: Parameters of the χ^2 distribution.

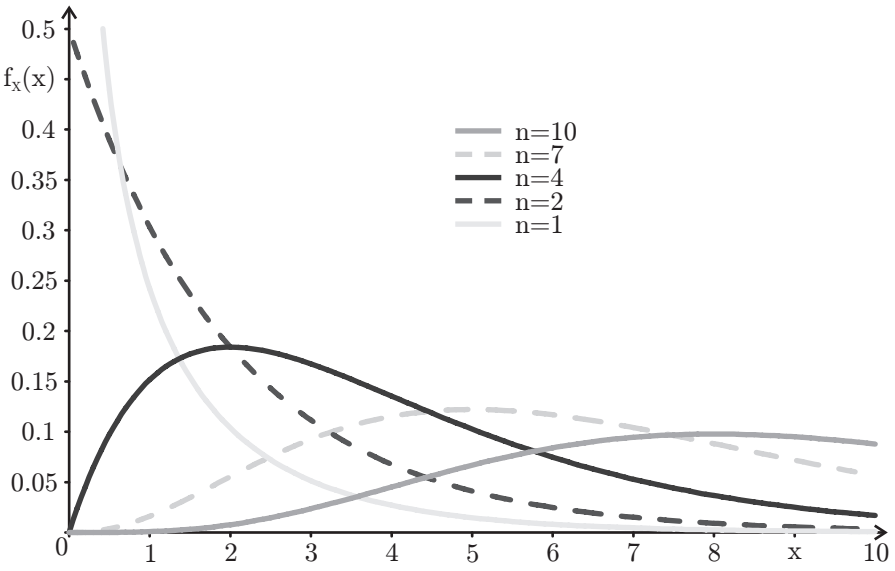


Figure 28.13: The PDFs of some χ^2 distributions

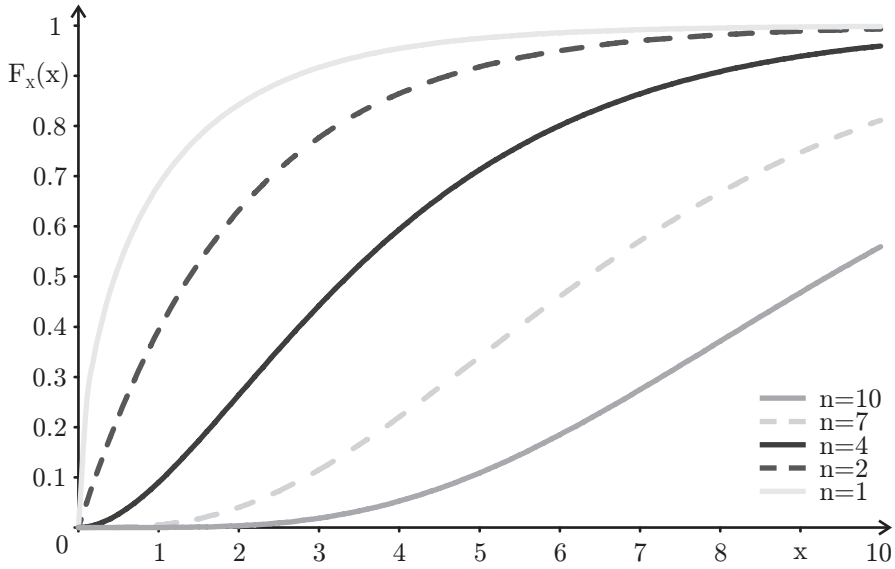


Figure 28.14: The CDFs of some χ^2 distributions

n	0.995	.99	.975	.95	.9	.1	.05	.025	.01	.005
1	–	–	0.001	0.004	0.016	2.706	3.841	5.024	6.635	7.879
2	0.010	0.020	0.051	0.103	0.211	4.605	5.991	7.378	9.210	10.597
3	0.072	0.115	0.216	0.352	0.584	6.251	7.815	9.348	11.345	12.838
4	0.207	0.297	0.484	0.711	1.064	7.779	9.488	11.143	13.277	14.860
5	0.412	0.554	0.831	1.145	1.610	9.236	11.070	12.833	15.086	16.750
6	0.676	0.872	1.237	1.635	2.204	10.645	12.592	14.449	16.812	18.548
7	0.989	1.239	1.690	2.167	2.833	12.017	14.067	16.013	18.475	20.278
8	1.344	1.646	2.180	2.733	3.490	13.362	15.507	17.535	20.090	21.955
9	1.735	2.088	2.700	3.325	4.168	14.684	16.919	19.023	21.666	23.589
10	2.156	2.558	3.247	3.940	4.865	15.987	18.307	20.483	23.209	25.188
11	2.603	3.053	3.816	4.575	5.578	17.275	19.675	21.920	24.725	26.757
12	3.074	3.571	4.404	5.226	6.304	18.549	21.026	23.337	26.217	28.300
13	3.565	4.107	5.009	5.892	7.042	19.812	22.362	24.736	27.688	29.819
14	4.075	4.660	5.629	6.571	7.790	21.064	23.685	26.119	29.141	31.319
15	4.601	5.229	6.262	7.261	8.547	22.307	24.996	27.488	30.578	32.801
16	5.142	5.812	6.908	7.962	9.312	23.542	26.296	28.845	32.000	34.267
17	5.697	6.408	7.564	8.672	10.085	24.769	27.587	30.191	33.409	35.718
18	6.265	7.015	8.231	9.390	10.865	25.989	28.869	31.526	34.805	37.156
19	6.844	7.633	8.907	10.117	11.651	27.204	30.144	32.852	36.191	38.582
20	7.434	8.260	9.591	10.851	12.443	28.412	31.410	34.170	37.566	39.997
21	8.034	8.897	10.283	11.591	13.240	29.615	32.671	35.479	38.932	41.401
22	8.643	9.542	10.982	12.338	14.041	30.813	33.924	36.781	40.289	42.796
23	9.260	10.196	11.689	13.091	14.848	32.007	35.172	38.076	41.638	44.181
24	9.886	10.856	12.401	13.848	15.659	33.196	36.415	39.364	42.980	45.559
25	10.520	11.524	13.120	14.611	16.473	34.382	37.652	40.646	44.314	46.928
26	11.160	12.198	13.844	15.379	17.292	35.563	38.885	41.923	45.642	48.290
27	11.808	12.879	14.573	16.151	18.114	36.741	40.113	43.195	46.963	49.645
28	12.461	13.565	15.308	16.928	18.939	37.916	41.337	44.461	48.278	50.993
29	13.121	14.256	16.047	17.708	19.768	39.087	42.557	45.722	49.588	52.336
30	13.787	14.953	16.791	18.493	20.599	40.256	43.773	46.979	50.892	53.672
40	20.707	22.164	24.433	26.509	29.051	51.805	55.758	59.342	63.691	66.766
50	27.991	29.707	32.357	34.764	37.689	63.167	67.505	71.420	76.154	79.490
60	35.534	37.485	40.482	43.188	46.459	74.397	79.082	83.298	88.379	91.952
70	43.275	45.442	48.758	51.739	55.329	85.527	90.531	95.023	100.425	104.215
80	51.172	53.540	57.153	60.391	64.278	96.578	101.879	106.629	112.329	116.321
90	59.196	61.754	65.647	69.126	73.291	107.565	113.145	118.136	124.116	128.299
100	67.328	70.065	74.222	77.929	82.358	118.498	124.342	129.561	135.807	140.169

Table 28.10: Some values of the χ^2 distribution.

28.5.5 Student's t-Distribution

The Student's t-distribution[56] is based on the insight that the mean of a normally distributed feature of a sample is no longer normally distributed if the variance is unknown and needs to be estimated from the data samples [840, 841, 679]. It has been design by Gosset [840] who published it under the pseudonym *Student*.

The parameter n of the distribution denotes the degrees of freedom of the distribution. If n approaches infinity, the t-distribution approaches the standard normal distribution.

The characteristic properties of Student's t-distribution are outlined in Table 28.11[57] and examples for its PDF and CDF are illustrated in Figure 28.15 and Figure 28.16.

Table 28.12 provides some selected values for the quantiles $t_{1-\alpha,n}$ of the t-distribution (one-sided confidence intervals, see Section 28.7.3 on page 503). The headline of the table contains results of the cumulative distribution function $F_X(x)$ of a Student's t-distribution with n degrees of freedom (values in the first column). The cells now denote the x values that belong to these $(n, F_X(x))$ combinations.

parameter	**definition**	
parameters	$n \in \mathbb{R}^+, n > 0$	(28.203)
PDF	$f_X(x) = \frac{\Gamma((n+1)/2)}{\sqrt{n\pi}\Gamma(n/2)} \left(1 + x^2/n\right)^{-(n+1)/2}$	(28.204)
CDF	$P(X \leq x) = F_X(x) = \frac{1}{2} + x\Gamma\left(\frac{n+1}{2}\right) \frac{{}_2F_1\left(\frac{1}{2}, \frac{n+1}{2}, \frac{3}{2}, -\frac{x^2}{n}\right)}{\sqrt{n\pi}\Gamma\left(\frac{n}{2}\right)}$	(28.205)
mean	$EX = 0$	(28.206)
median	$\text{med} = 0$	(28.207)
mode	$\text{mode} = 0$	(28.208)
variance	$D^2X = \frac{n}{n-2}$ *for* $n > 2$, *otherwise undefined*	(28.209)
skewness	$\gamma_1 = 0$ *for* $n > 3$	(28.210)
kurtosis	$\gamma_2 = \frac{6}{n-4}$ *for* $n > 4$	(28.211)
entropy	$\text{h}(X) = \frac{n}{2}\left[\psi\left(\frac{n+1}{2}\right) - \psi\left(\frac{n}{2}\right)\right] + \log\left[\sqrt{n}\text{B}\left(\frac{n}{2}, \frac{1}{2}\right)\right]$	(28.212)
mgf	*undefined*	(28.213)

Table 28.11: Parameters of the Student's t- distribution.

[56] http://en.wikipedia.org/wiki/Student%27s_t-distribution [accessed 2007-09-30]

[57] More information on the gamma function Γ used in Equation 28.204 and Equation 28.205 can be found in Section 28.10.1 on page 532. ${}_2F_1$ in Equation 28.205 stands for the hypergeometric function, ψ and B in Equation 28.212 are the digamma and the beta function.

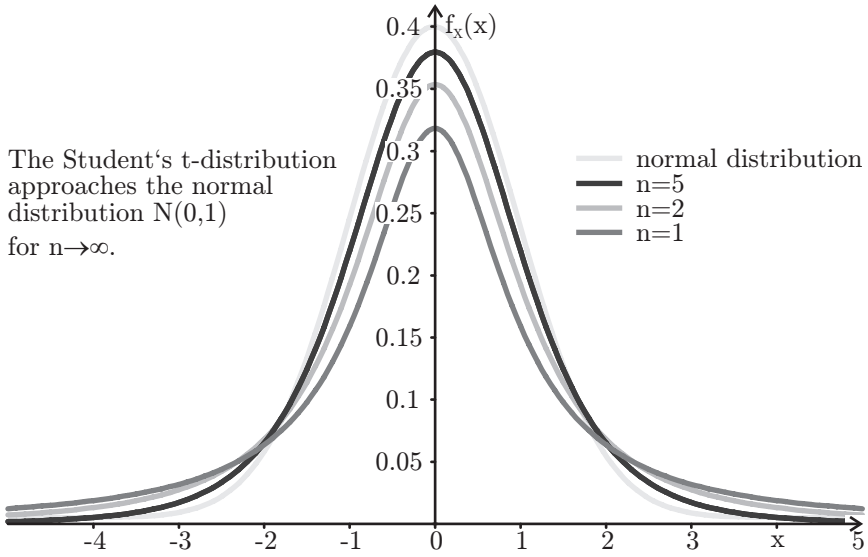


Figure 28.15: The PDFs of some Student's t-distributions

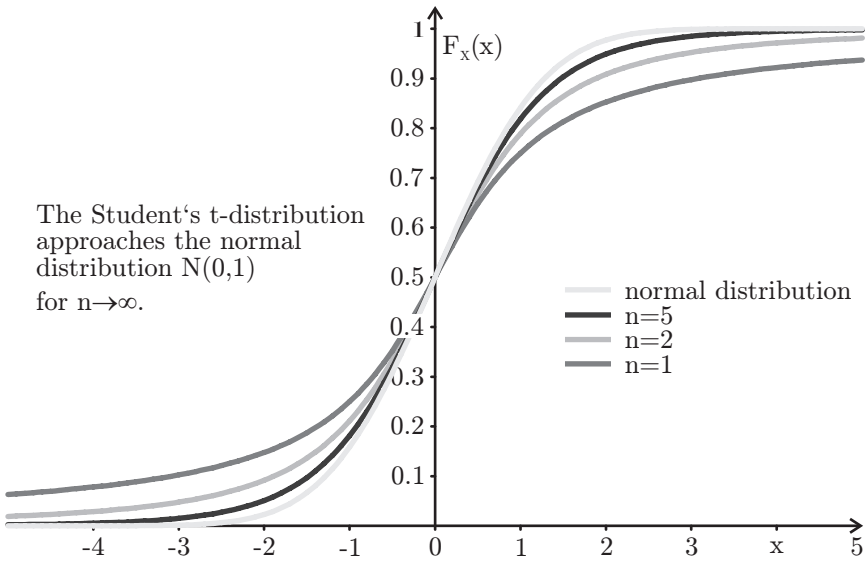


Figure 28.16: The CDFs of some Student's t-distributions

n	0.75	.8	.85	.875	.9	.95	.975	.99	.995	.9975	.999	.9995
1	1.000	1.376	1.963	2.414	3.078	6.314	12.71	31.82	63.66	127.3	318.3	636.6
2	0.816	1.061	1.386	1.605	1.886	2.920	4.303	6.965	9.925	14.09	22.33	31.60
3	0.765	0.978	1.250	1.423	1.638	2.353	3.182	4.541	5.841	7.453	10.21	12.92
4	0.741	0.941	1.190	1.344	1.533	2.132	2.776	3.747	4.604	5.598	7.173	8.610
5	0.727	0.920	1.156	1.301	1.476	2.015	2.571	3.365	4.032	4.773	5.893	6.869
6	0.718	0.906	1.134	1.273	1.440	1.943	2.447	3.143	3.707	4.317	5.208	5.959
7	0.711	0.896	1.119	1.254	1.415	1.895	2.365	2.998	3.499	4.029	4.785	5.408
8	0.706	0.889	1.108	1.240	1.397	1.860	2.306	2.896	3.355	3.833	4.501	5.041
9	0.703	0.883	1.100	1.230	1.383	1.833	2.262	2.821	3.250	3.690	4.297	4.781
10	0.700	0.879	1.093	1.221	1.372	1.812	2.228	2.764	3.169	3.581	4.144	4.587
11	0.697	0.876	1.088	1.214	1.363	1.796	2.201	2.718	3.106	3.497	4.025	4.437
12	0.695	0.873	1.083	1.209	1.356	1.782	2.179	2.681	3.055	3.428	3.930	4.318
13	0.694	0.870	1.079	1.204	1.350	1.771	2.160	2.650	3.012	3.372	3.852	4.221
14	0.692	0.868	1.076	1.200	1.345	1.761	2.145	2.624	2.977	3.326	3.787	4.140
15	0.691	0.866	1.074	1.197	1.341	1.753	2.131	2.602	2.947	3.286	3.733	4.073
16	0.690	0.865	1.071	1.194	1.337	1.746	2.120	2.583	2.921	3.252	3.686	4.015
17	0.689	0.863	1.069	1.191	1.333	1.740	2.110	2.567	2.898	3.222	3.646	3.965
18	0.688	0.862	1.067	1.189	1.330	1.734	2.101	2.552	2.878	3.197	3.610	3.922
19	0.688	0.861	1.066	1.187	1.328	1.729	2.093	2.539	2.861	3.174	3.579	3.883
20	0.687	0.860	1.064	1.185	1.325	1.725	2.086	2.528	2.845	3.153	3.552	3.850
21	0.686	0.859	1.063	1.183	1.323	1.721	2.080	2.518	2.831	3.135	3.527	3.819
22	0.686	0.858	1.061	1.182	1.321	1.717	2.074	2.508	2.819	3.119	3.505	3.792
23	0.685	0.858	1.060	1.180	1.319	1.714	2.069	2.500	2.807	3.104	3.485	3.767
24	0.685	0.857	1.059	1.179	1.318	1.711	2.064	2.492	2.797	3.091	3.467	3.745
25	0.684	0.856	1.058	1.178	1.316	1.708	2.060	2.485	2.787	3.078	3.450	3.725
26	0.684	0.856	1.058	1.177	1.315	1.706	2.056	2.479	2.779	3.067	3.435	3.707
27	0.684	0.855	1.057	1.176	1.314	1.703	2.052	2.473	2.771	3.057	3.421	3.690
28	0.683	0.855	1.056	1.175	1.313	1.701	2.048	2.467	2.763	3.047	3.408	3.674
29	0.683	0.854	1.055	1.174	1.311	1.699	2.045	2.462	2.756	3.038	3.396	3.659
30	0.683	0.854	1.055	1.173	1.310	1.697	2.042	2.457	2.750	3.030	3.385	3.646
40	0.681	0.851	1.050	1.167	1.303	1.684	2.021	2.423	2.704	2.971	3.307	3.551
50	0.679	0.849	1.047	1.164	1.299	1.676	2.009	2.403	2.678	2.937	3.261	3.496
60	0.679	0.848	1.045	1.162	1.296	1.671	2.000	2.390	2.660	2.915	3.232	3.460
80	0.678	0.846	1.043	1.159	1.292	1.664	1.990	2.374	2.639	2.887	3.195	3.416
100	0.677	0.845	1.042	1.158	1.290	1.660	1.984	2.364	2.626	2.871	3.174	3.390
120	0.677	0.845	1.041	1.157	1.289	1.658	1.980	2.358	2.617	2.860	3.160	3.373
∞	0.674	0.842	1.036	1.150	1.282	1.645	1.960	2.326	2.576	2.807	3.090	3.291

Table 28.12: Table of Student's t-distribution with right-tail probabilities.

28.6 Example – Throwing a Dice

Let us now discuss the different parameters of a random variable at the example of throwing a dice. On a dice, numbers from one to six are written and the result of throwing it is the number written on the side facing upwards. If a dice is perfect, the numbers one to six will show up with exactly the same probability, $\frac{1}{6}$. The set of all possible outcomes of throwing a dice Ω is thus

$$\Omega = \Big\{\boxed{1}, \boxed{2}, \boxed{3}, \boxed{4}, \boxed{5}, \boxed{6}\Big\} \tag{28.214}$$

We define a random variable $X : \Omega \mapsto \mathbb{R}$ that assigns real numbers to the possible outcomes of throwing the dice in a way that the value of X matches the number on the dice:

$$X : \Omega \mapsto \{1, 2, 3, 4, 5, 6\} \tag{28.215}$$

It is obviously a uniformly distributed discrete random variable (see Section 28.4.1 on page 479) that can take on six states. We can now define the probability mass function PMF and the according cumulative distribution function CDF as follows (see also Figure 28.17):

$$F_X(x) = P(X \leq x) = \begin{cases} 0 & \text{if } x < 1 \\ \frac{x}{6} & \text{if } 1 \leq x \leq 6 \\ 1 & \text{otherwise} \end{cases} \tag{28.216}$$

$$f_X(x) = P(X = x) = \begin{cases} 0 & \text{if } x < 1 \\ \frac{1}{6} & \text{if } 1 \leq x \leq 6 \\ 0 & \text{otherwise} \end{cases} \tag{28.217}$$

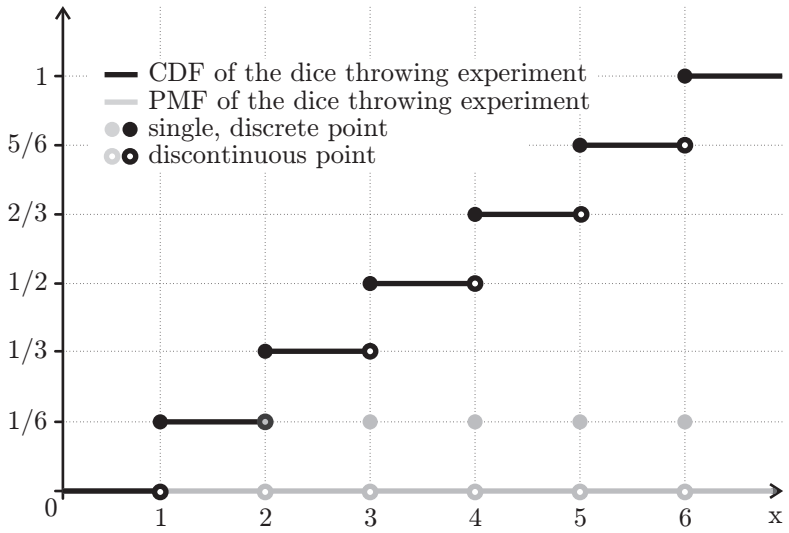


Figure 28.17: The PMF and CMF of the dice throw

We now can discuss the statistical parameters of this experiment. This is a good opportunity to compare the real parameters and their estimates. We therefore assume that the dice was thrown ten times ($n = 10$) in an experiment. The following numbers have been thrown as illustrated in Figure 28.18):

$$A = \{4, 5, 3, 2, 4, 6, 4, 2, 5, 3\} \tag{28.218}$$

Table 28.13 outlines how the parameters of the random variable are computed. The real

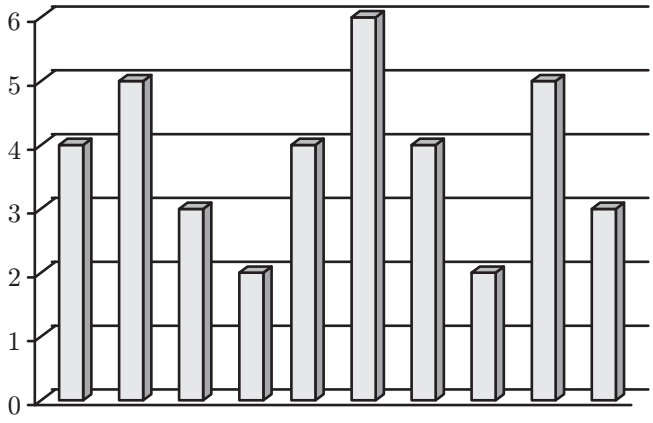


Figure 28.18: The numbers thrown in the dice example

parameter	true value	estimate	
count	non existent	$n = \text{len}(A) = 10$	(28.219)
minimum	$a = \min\{x : f_X(x) > 0\} = 1$	$\widetilde{a} = \min A = 2 \approx a$	(28.220)
maximum	$b = \max\{x : f_X(x) > 0\} = 6$	$\widetilde{b} = \max A = 6 \approx b$	(28.221)
range	$\text{range} = r = b - a + 1 = 6$	$\widetilde{r} = b - a + 1 = 6 \approx \text{range}$	(28.222)
mean	$EX = \frac{a+b}{2} = \frac{7}{2} = 3.5$	$\overline{a} = \frac{1}{n}\sum_{i=0}^{n-1} A_{[i]} = \frac{19}{5} = 3.8 \approx EX$	(28.223)
median	$\text{med} = \frac{a+b}{2} = \frac{7}{2} = 3.5$	$A_s = \text{sortList}_a(A, >)$ $\widetilde{\text{med}} = \frac{A_s\left[\frac{n}{2}\right] + A_s\left[\frac{n}{2}-1\right]}{2} = 4 \approx \text{med}$	(28.224)
mode	$\text{mode} = \emptyset$	$\widetilde{\text{mode}} = \{4\} \approx \text{mode}$	(28.225)
variance	$D^2X = \frac{r^2-1}{12} = \frac{35}{12} \approx 2.917$	$s^2 = \frac{1}{n-1}\sum_{i=0}^{n-1}(A_{[i]} - \overline{a})^2 = \frac{26}{15} \approx 1.73 \approx D^2X$	(28.226)
skewness	$\gamma_1 = 0$	$G_1 \approx 0.0876 \approx \gamma_1$	(28.227)
kurtosis	$\gamma_2 = -\frac{6\left(r^2+1\right)}{5\left(r^2-1\right)} = -\frac{222}{175} \approx -1.269$	$G_2 \approx -0.7512 \approx \gamma_2$	(28.228)

Table 28.13: Parameters of the dice throw experiment.

values of the parameters are defined using the PMF or CDF functions, while the estimations are based on the sample data obtained from our experiment solely.

As you can see, the estimations of the parameters sometimes differ significantly from their true values. More information about estimation can be found in the following section.

28.7 Estimation Theory

28.7.1 Introduction

Estimation theory is the science of approximating the values of parameters based on measurements or otherwise obtained sample data [1727, 1105, 1668, 2100, 1882]. The center of this branch of statistics is to find good estimators in order to approximate the real values the parameters as good as possible.

Definition 28.51 (Estimator). An estimator[58] $\widetilde{\theta}$ is a rule (most often a mathematical function) that takes a set of sample data A as input and returns an estimation of one parameter θ of the random distribution of the process sampled with this data set.

We have already discussed some estimators in Section 28.3 – the arithmetic mean of a sample data set (see Definition 28.29 on page 473) for example is an estimator for the expected value (see Definition 28.27 on page 473) and in Equation 28.67 on page 474 we have introduced an estimator for the sample variance.

Obviously, the estimator $\widetilde{\theta}$ is the better the closer its results (the estimates) come to the real values of the parameter θ.

Definition 28.52 (Point Estimator). We define a point estimator $\widetilde{\theta}$ to be an estimator which is a mathematical function $\widetilde{\theta} : \mathbb{R}^n \mapsto \mathbb{R}$. This function takes the data sample A (here considered as a real vector $A \in \mathbb{R}^n$) as input and returns the estimate in the form of a (real) scalar value.

Definition 28.53 (Error). The absolute (estimation) error ε[59] is the difference between the value returned by a point estimator $\widetilde{\theta}$ of a parameter θ for a certain input A and its real value. Notice that the error ε can be zero, positive, or negative.

$$\varepsilon_A\left(\widetilde{\theta}\right) = \widetilde{\theta}(A) - \theta \tag{28.229}$$

In the following, we will most often not explicitly refer to the data sample A as basis of the estimation $\widetilde{\theta}$ anymore. We assume that it is implicitly clear that estimations are usually based on such samples and that subscripts like the A in ε_A in Equation 28.229 are not needed.

Definition 28.54 (Bias). The bias $\text{Bias}\left(\widetilde{\theta}\right)$ of an estimator $\widetilde{\theta}$ is the expected value of the difference of the estimate and the real value. This mean error is null for all unbiased estimators.

$$\text{Bias}\left(\widetilde{\theta}\right) = E\left[\widetilde{\theta} - \theta\right] = E\left[\varepsilon\left(\widetilde{\theta}\right)\right] \tag{28.230}$$

Definition 28.55 (Unbiased Estimator). An unbiased estimator has a zero bias.

$$\text{Bias}\left(\widetilde{\theta}\right) = E\left[\widetilde{\theta} - \theta\right] = E\left[\varepsilon\left(\widetilde{\theta}\right)\right] = 0 \Leftrightarrow E\widetilde{\theta} = \theta \tag{28.231}$$

Definition 28.56 (Mean Square Error). The mean square error[60] $\text{MSE}\left(\widetilde{\theta}\right)$ of an estimator $\widetilde{\theta}$ is the expected value of the square of the estimation error ε. It is also the sum of the variance of the estimator and the square of its bias. The MSE is a measure for how much an estimator differs from the quantity to be estimated.

[58] http://en.wikipedia.org/wiki/Estimator [accessed 2007-07-03], http://mathworld.wolfram.com/Estimator.html [accessed 2007-07-03]

[59] http://en.wikipedia.org/wiki/Errors_and_residuals_in_statistics [accessed 2007-07-03]

[60] http://en.wikipedia.org/wiki/Mean_squared_error [accessed 2007-07-03]

$$\text{MSE}\left(\widetilde{\theta}\right) = E\left[\left(\widetilde{\theta} - \theta\right)^2\right] = E\left[\left(\varepsilon\left(\widetilde{\theta}\right)\right)^2\right] \tag{28.232}$$

$$\text{MSE}\left(\widetilde{\theta}\right) = D^2\widetilde{\theta} + \left(\text{Bias}\left(\widetilde{\theta}\right)\right)^2 \tag{28.233}$$

Notice that the MSE of unbiased estimators coincides with the variance $D^2\widetilde{\theta}$ of $\widetilde{\theta}$. For estimating the mean square error of an estimator $\widetilde{\theta}$, we use the sample mean:

$$\widetilde{\text{MSE}}\left(\widetilde{\theta}\right) = \frac{1}{n}\sum_{i=1}^{n}\left(\widetilde{\theta}_i - \widetilde{\theta}\right) \tag{28.234}$$

28.7.2 Likelihood and Maximum Likelihood Estimators

Definition 28.57 (Likelihood). Likelihood[61] is a mathematical expression complementary to probability. Whereas probability allows us to predict the outcome of a random experiment based on known parameters, likelihood allows us to predict unknown parameters based on the outcome of experiments.

Definition 28.58 (Likelihood Function). The likelihood function L returns a value that is proportional to the probability of a postulated underlying law or probability distribution φ according to an observed outcome (denoted as the vector $\mathbf{y}$). Notice that L not necessarily represents a probability density/mass function and its integral also does not necessarily equal to 1.

$$\text{L}[\varphi\,|\mathbf{y}\,] \propto P(\mathbf{y}|\,\varphi) \tag{28.235}$$

In many sources, L is defined in dependency of a parameter θ instead of the function φ. We preferred the latter notation since it is a more general superset of the first one.

Observation of an Unknown Process φ

Assume that we are given a finite set A of n sample data points.

$$A = \{(x_1, y_1), (x_2, y_2), .., (x_n, y_n)\},\ x_i, y_i \in \mathbb{R}\ \forall i \in [1, n] \tag{28.236}$$

The x_i are known inputs or parameters of an unknown process defined by the function $\varphi : \mathbb{R} \mapsto \mathbb{R}$. By observing the corresponding outputs of the process, we have obtained the y_i values. During our observations, we make the measurement errors[62] η_i.

$$y_i = \varphi(x_i) + \eta_i\ \forall i : 0 < i \leq n \tag{28.237}$$

About this measurement error η we make the following assumptions:

$$E\eta = 0 \tag{28.238}$$

$$\eta \sim N(0, \sigma^2)\ :\ 0 < \sigma < \infty \tag{28.239}$$

$$\text{cov}(\eta_i, \eta_j) = 0\ \forall i, j \in \mathbb{N} : i \neq j, 0 < i \leq n, 0 < j \leq n \tag{28.240}$$

1. The expected values of η in Equation 28.238 are all zero. Our measurement device thus gives us, in average, unbiased results. If the expected value of η was not zero, we could simple recalibrate our (imaginary) measurement equipment in order to subtract $E\eta$ from all measurements and would obtain unbiased observations.

[61] http://en.wikipedia.org/wiki/Likelihood [accessed 2007-07-03]

[62] http://en.wikipedia.org/wiki/Measurement_error [accessed 2007-07-03]

2. Furthermore, Equation 28.239 states that the η_i are normally distributed around the zero point with an unknown, nonzero variance σ^2. To suppose measurement errors to be normally distributed is quite common and correct in most cases. The white noise[63] in transmission of signals for example is often modeled with Gaussian distributed[64] amplitudes. This second assumption includes, of course, the first one: Being normally distributed with $N(\mu = 0, \sigma^2)$ implies a zero expected value of the error.
3. With Equation 28.240, we assume that the errors η_i of the single measurements are stochastically independent. If there existed a connection between them, it would be part of the underlying physical law φ and could be incorporated in our measurement device and again be subtracted.

Objective: Estimation

Assume that we can choose from a, possible infinite large, set of functions (estimators) $f \in F$.

$$f \in F \Rightarrow f : \mathbb{R} \mapsto \mathbb{R} \tag{28.241}$$

From this set we want to pick the function $f^\star \in F$ with that resembles φ the best (i. e., better than all other $f \in F : f \not\equiv f^\star$). φ is not necessarily an element of F, so we cannot always presume to find a $f^\star \equiv \varphi$.

Each estimator f deviates by the estimation error $\varepsilon(f)$ (see Definition 28.53 on page 499) from the y_i-values. The estimation error depends on f and may vary for different estimators.

$$y_i = f(x_i) + \varepsilon_i(f) \ \forall i : 0 < i \leq n \tag{28.242}$$

We consider all $f \in F$ to be valid estimators for φ and simple look for the one that "fits best". We now can combine Equation 28.242 with Equation 28.237:

$$f(x_i) + \varepsilon_i(f) = y_i = \varphi(x_i) + \eta_i \ \forall i : 0 < i \leq n \tag{28.243}$$

We do not know φ and thus, cannot determine the η_i. According to the likelihood method, we pick the function $f \in F$ that would have most probably produced the outcomes y_i. In other words, we have to maximize the likelihood of the occurrence of the $\varepsilon_i(f)$. The likelihood here is defined under the assumption that the true measurement errors η_i are normally distributed (see Equation 28.239). So what we can do is to determine the ε_i in a way that their occurrence is most probable according to the distribution of the random variable that created the η_i, $N(0, \sigma^2)$. In the best case, the $\varepsilon(f^\star) = \eta_i$ and thus, $f^\star$ is equivalent to $\varphi(x_i)$, at least in for the sample information A available to us.

Maximizing the Likelihood

Therefore, we can regard the $\varepsilon_i(f)$ as outcomes of independent random experiments, as uncorrelated random variables, and combine them to a multivariate normal distribution. For the ease of notation, we define the $\boldsymbol{\varepsilon}(f)$ to be the vector containing all the single $\varepsilon_i(f)$-values.

$$\boldsymbol{\varepsilon}(f) = \begin{pmatrix} \varepsilon_1(f) \\ \varepsilon_2(f) \\ \vdots \\ \varepsilon_n(f) \end{pmatrix} \tag{28.244}$$

[63] http://en.wikipedia.org/wiki/White_noise [accessed 2007-07-03]
[64] http://en.wikipedia.org/wiki/Gaussian_noise [accessed 2007-07-03]

The probability density function of a multivariate normal distribution with independent variables ε_i that have the same variance σ^2 looks like this (as defined in Equation 28.178 on page 489):

$$f_X(\varepsilon(f)) = \left(\frac{1}{2\pi\sigma^2}\right)^{\frac{n}{2}} e^{-\frac{\sum_{i=1}^{n}(\varepsilon_i(f)-\mu)^2}{2\sigma^2}} = \left(\frac{1}{2\pi\sigma^2}\right)^{\frac{n}{2}} e^{-\frac{\sum_{i=1}^{n}(\varepsilon_i(f))^2}{2\sigma^2}} \tag{28.245}$$

Amongst all possible vectors $\boldsymbol{\varepsilon}(f) : f \in F$ we need to find the most probable one $\boldsymbol{\varepsilon}^\star = \boldsymbol{\varepsilon}(f^\star)^\star$ according to Equation 28.245. The function $f^\star$ that produces it will then be the one that most probably matches to φ.

In order to express how likely the observation of some outcomes is under a certain set of parameters, we have defined the likelihood function L in Definition 28.58. Here we can use the probability density function f_X of the normal distribution, since the maximal values of f_X are those that are most probable to occur.

$$\mathrm{L}[\boldsymbol{\varepsilon}(f)|\, f] = f_X(\boldsymbol{\varepsilon}(f)) = \left(\frac{1}{2\pi\sigma^2}\right)^{\frac{n}{2}} e^{-\frac{\sum_{i=1}^{n}(\varepsilon_i(f))^2}{2\sigma^2}} \tag{28.246}$$

$$f^\star \in F : \mathrm{L}[\boldsymbol{\varepsilon}(f^\star)|\, f^\star] = \max_{\forall f \in F} \mathrm{L}[\boldsymbol{\varepsilon}(f)|\, f] \tag{28.247}$$

$$= \max_{\forall f \in F} \left(\frac{1}{2\pi\sigma^2}\right)^{\frac{n}{2}} e^{-\frac{\sum_{i=1}^{n}(\varepsilon_i(f))^2}{2\sigma^2}} \tag{28.248}$$

Finding a $f^\star$ that Maximizes the function f_X however is equal to find a $f^\star$ that minimizes the sum of the squares of the ε-values.

$$f^\star \in F : \sum_{i=1}^{n} (\varepsilon_i(f^\star))^2 = \min_{\forall f \in F} \sum_{i=1}^{n} (\varepsilon_i(f))^2 \tag{28.249}$$

According to Equation 28.242 we can now substitute the ε_i-values with the difference between the observed outcomes y_i and the estimates $f(x_i)$.

$$\sum_{i=1}^{n} (\varepsilon_i(f))^2 = \sum_{i=1}^{n} (y_i - f(x_i))^2 \tag{28.250}$$

Definition 28.59 (Maximum Likelihood Estimator). A maximum likelihood estimator[65] [37] $f^\star$ is an estimator which fits with maximum likelihood to a given set of sample data A. Under the particular assumption of uncorrelated error terms normally distributed around zero, a MLE minimizes Equation 28.251.

$$f^\star \in F : \sum_{i=1}^{n} (y_i - f^\star(x_i))^2 = \min_{\forall f \in F} \sum_{i=1}^{n} (y_i - f(x_i))^2 \tag{28.251}$$

Minimizing the sum of the difference between the observed y_i and the estimates $f(x_i)$ also minimizes their mean, so with this we have also shown that the estimator that minimizes mean square error MSE (see Definition 28.56) is the best estimator according to the likelihood of the produced outcomes.

$$f^\star \in F : \frac{1}{n} \sum_{i=1}^{n} (y_i - f^\star(x_i))^2 = \min_{\forall f \in F} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2 \tag{28.252}$$

$$f^\star \in F : \mathrm{MSE}(f^\star) = \min_{\forall f \in F} \mathrm{MSE}(f) \tag{28.253}$$

[65] http://en.wikipedia.org/wiki/Maximum_likelihood [accessed 2007-07-03]

The term $(y_i - f(x_i))^2$ is often justified by the statement that large deviations of f from the y-values are punished harder than smaller ones. The correct reason why we minimize the square error, however, is that we maximize the likelihood of the resulting estimator.

At this point, one should also notice that the x_i also could be replaced with vectors $\mathbf{x}_i \in \mathbb{R}^m$ without any further implications or modifications of the equations.

In most practical cases, the set F of possible functions is closely defined. It usually contains only one type of parameterized function, so we only have to determine the unknown parameters in order to find $f^\star$. Let us consider a set of linear functions as example. If we want to find estimators of the form $F = \{\forall\, f(x) = ax + b : a, b \in \mathbb{R}\}$, we will minimize Equation 28.254 by determining the best possible values for a and b.

$$\text{MSE}(\, f(x)|\, a, b) = \frac{1}{n}\sum_{i=1}^{n}(ax_i + b - y_i)^2 \tag{28.254}$$

If we now could find a perfect estimator $f_p^\star$ and our data would be free of any measurement error, all parts of the sum would become zero. For $n > 2$, this perfect estimator would be the solution of the over-determined system of linear equations illustrated in Equation 28.255.

$$\begin{aligned} 0 &= ax_1 + b - y_1 \\ 0 &= ax_2 + b - y_2 \\ \dots &\quad \dots \\ 0 &= ax_n + b - y_n \end{aligned} \tag{28.255}$$

Since it is normally not possible to obtain a perfect estimator because there are measurement errors or other uncertainties like unknown dependencies, the system in Equation 28.255 often cannot be solved but only minimized.

Best Linear Unbiased Estimators

The Gauss-Markov Theorem[66] defines BLUEs (best linear unbiased estimators) according to the facts just discussed:

Definition 28.60 (BLUE). In a linear model in which the measurement errors ε_i are uncorrelated and are all normally distributed with an expected value of zero and the same variance, the best linear unbiased estimators (BLUE) of the (unknown) coefficients are the least-square estimators [1649].

Hence, for the best linear unbiased estimator also the same three assumptions (Equation 28.238, Equation 28.239, and Equation 28.240 on page 500) as for the maximum likelihood estimator hold.

28.7.3 Confidence Intervals

There is a very simple principle in statistics that always holds: *All estimates may as well be wrong.* There is no guarantee whatsoever that we have estimated a parameter of an underlying distribution correct regardless how many samples we have analyzed. However, if we can assume or know the underlying distribution of the process which has been sampled, we can compute certain intervals which include the real value of the estimated parameter with a certain probability.

Definition 28.61 (Confidence Interval). Unlike point estimators, which approximate a parameter of a data sample with a single value, confidence intervals[67] (CIs) are estimations that give certain upper and lower boundaries in which the true value of the parameter will be located with a certain, predefined probability. [684, 351, 478]

[66] http://en.wikipedia.org/wiki/Gauss-Markov_theorem [accessed 2007-07-03], http://www.answers.com/topic/gauss-markov-theorem [accessed 2007-07-03]

[67] http://en.wikipedia.org/wiki/Confidence_interval [accessed 2007-10-01]

The advantage of confidence intervals is that we can directly derive the significance of the data samples from them – the larger the intervals are, the less reliable is the sample. The narrower confidence intervals get for high predefined probabilities, the more profound, i. e., significant, will the conclusions drawn from them be.

Example

Imagine we run a farm and own 25 chickens. Each chicken lays one egg a day. We collect all the eggs in the morning and weigh them in order to find the average weight of the eggs produced by our farm. Assume our sample contains the values (in g):

$$A = \begin{array}{l}\{\ 120, 121, 119, 116, 115, 122, 121, 123, 122, 120 \\ \quad 119, 122, 121, 120, 119, 121, 123, 117, 118, 121\ \}\end{array} \tag{28.256}$$

$$n = \text{len}(A) = 20 \tag{28.257}$$

From these measurements, we can determine the arithmetic mean $\overline{a}$ and the sample variance s^2 according to Equation 28.60 on page 473 and Equation 28.67 on page 474:

$$\overline{a} = \frac{1}{n}\sum_{i=0}^{n-1} A_{[i]} = \frac{2400}{20} = 120 \tag{28.258}$$

$$s^2 = \frac{1}{n-1}\sum_{i=0}^{n-1}(A_{[i]} - \overline{a})^2 = \frac{92}{19} \tag{28.259}$$

The question that arises now is if the mean of 120 is significant, i. e., whether it likely approximates the expected value of the egg weight, or if the data sample was too small to be representative. Furthermore, we would like to know in which interval the expected value of the egg weights will likely be located. Now confidence intervals come into play. First, we need to find out what the underlying distribution of the random variable producing A as sample output is. In case of chicken eggs, we safely can *assume*[68] that it is the normal distribution discussed in Section 28.5.2 on page 486. With that we can calculate an interval which includes the unknown parameter μ (i. e., the real expected value) with a confidence probability of γ. $\gamma = 1 - a$ is the so-called confidence coefficient and a is the probability that the real value of the estimated parameter lies not inside the confidence interval.

Let us compute the interval including the expected value μ of the chicken egg weights with a probability of $\gamma = 1 - a = 95\%$. Thus, $a = 0.05$. Therefore, we have to pick the right formula from Section 28.7.3 on the facing page (here it is Equation 28.272 on the next page) and substitute in the proper values:

$$\mu_\gamma \in \left[\overline{a} \pm t_{1-\frac{a}{2},n-1}\frac{s}{\sqrt{n}}\right] \tag{28.260}$$

$$\mu_{95\%} \in \left[120 \pm t_{0.975,19} * \frac{\sqrt{\frac{92}{19}}}{\sqrt{19}}\right] \tag{28.261}$$

$$\mu_{95\%} \in [120 \pm 2.093 * 0.5048] \tag{28.262}$$

$$\mu_{95\%} \in [118.94, 121.06] \tag{28.263}$$

The value of $t_{19,0.025}$ can easily be obtained from Table 28.12 on page 496 which contains the respective quantiles of Student's t-distribution discussed in Section 28.5.5 on page 494. Let us repeat the procedure in order to find the interval that will contain μ with probabilities $1 - \gamma = 99\% \Rightarrow a = 0.01$ and $1 - \gamma = 90\% \Rightarrow a = 0.1$:

[68] Notice that such an assumption is also a possible source of error!

$$\mu_{99\%} \in [120 \pm t_{0.995,19} * 0.5048] \tag{28.264}$$
$$\mu_{99\%} \in [120 \pm 2.861 * 0.5048] \tag{28.265}$$
$$\mu_{99\%} \in [118.56, 121.44] \tag{28.266}$$
$$\mu_{90\%} \in [120 \pm t_{0.95,19} * 0.5048] \tag{28.267}$$
$$\mu_{90\%} \in [120 \pm 1.729 * 0.5048] \tag{28.268}$$
$$\mu_{90\%} \in [119.13, 120.87] \tag{28.269}$$
$$\tag{28.270}$$

As you can see, the higher the confidence probabilities we specify the larger become the intervals in which the parameter is contained. We can be to 99% sure that the expected value of laid eggs is somewhere between 118.56 and 121.44. If we narrow the interval down to $[119.13, 120.87]$, we can only be 90% confident that the real expected value falls in it based on the data samples which we have gathered.

Some Hand-Picked Confidence Intervals

The following confidence intervals are two-sided, i. e., we determine a range $\widetilde{\theta}_\gamma \in \left[\widetilde{\theta}' - x, \widetilde{\theta}' + x\right]$ that contains the parameter θ with probability γ based on the estimate $\widetilde{\theta}$. If you need a one-sided confidence interval like $\widetilde{\theta}_\gamma \in \left(-\infty, \widetilde{\theta} + x\right]$ or $\widetilde{\theta}_\gamma \in \left[\widetilde{\theta}' - x, \infty\right)$, you just need to replace $1 - \frac{a}{2}$ with $1 - a$ in the equations.

Expected Value of a Normal Distribution $N(\mu, \sigma^2)$

With knowing the variance σ^2: If the exact variance σ^2 of the distribution underlying our data samples is known, and we have an estimate of the expected value μ by the arithmetic mean $\overline{a}$ according to Equation 28.60 on page 473, the two-sided confidence interval (of probability γ) for the expected value of the normal distribution is:

$$\mu_\gamma \in \left[\overline{a} \pm z\left(1 - \frac{a}{2}\right) \frac{\sigma}{\sqrt{n}}\right] \tag{28.271}$$

Where $z(y) \equiv \text{probit}(y) \equiv \Phi^{-1} y$ is the y-quantil of the standard normal distribution (see Definition 28.49 on page 488) which can for example be looked up in Table 28.7.

With estimated sample variance s^2: Often, the true variance σ^2 of an underlying distribution is not known and instead estimated with the sample variance s^2 according to Equation 28.67 on page 474. The two-sided confidence interval (of probability γ) for the expected value can then be computed using the arithmetic mean $\overline{a}$ and the estimate of the standard deviation $s = \sqrt{s^2}$ of the sample and the $t_{n-1,1-\frac{a}{2}}$ quantile of Student's t-distribution which can be looked up in Table 28.12 on page 496.

$$\mu_\gamma \in \left[\overline{a} \pm t_{1-\frac{a}{2},n-1} \frac{s^2}{\sqrt{n}}\right] \tag{28.272}$$

Variance of a Normal Distribution

The two-sided confidence interval (of probability γ) for the variance of a normal distribution can computed using sample variance s^2 and the $\chi^2(p,k)$-quantile of the χ^2 distribution which can be looked up in Table 28.10 on page 493.

$$\sigma_\gamma^2 \in \left[\frac{(n-1)s^2}{\chi^2\left(1 - \frac{a}{2}, n-1\right)}, \frac{(n-1)s^2}{\chi^2\left(\frac{\alpha}{2}, n-1\right)}\right] \tag{28.273}$$

Success Probability p of a $B(1,p)$ Binomial Distribution

The two-sided confidence interval (of probability γ) of the success probability p of a $B(1,p)$ binomial distribution can be computed as follows:

$$p_\gamma \in \left[\frac{n}{n+z^2_{1-\frac{a}{2}}}\left(\overline{a}+\frac{1}{2n}z^2_{1-\frac{a}{2}} \pm z^2_{1-\frac{a}{2}}\sqrt{\frac{\overline{a}(1-\overline{a})}{n}+\left(\frac{1}{2n}z^2_{1-\frac{a}{2}}\right)^2}\right)\right] \tag{28.274}$$

Expected Value of an Unknown Distribution with Sample Variance

The two-sided confidence interval (of probability γ) of the expected value EX of an unknown distribution with an unknown real variance D^2X can be determined using the arithmetic mean $\overline{a}$ and the sample variance s^2 if the sample data set contains more than $n = 50$ elements.

$$EX_\gamma \in \left[\overline{a} \pm z\left(1-\frac{a}{2}\right)\frac{s}{\sqrt{n}}\right] \tag{28.275}$$

Confidence Intervals from Tests

Many statistical tests (such as the Wilcoxon's signed rank test introduced in Section 28.8.1) can be inverted in order to obtain confidence intervals [205]. The topic of statistical tests are discussed in Section 28.8.

28.7.4 Density Estimation

In this section we discuss density estimation[69] techniques [185, 1845]. Density estimation is often used by global optimization algorithms in order to test whether a certain region of the search space has already been explored sufficiently and where to concentrate the further search efforts.

Definition 28.62 (Density Estimation). A density estimation $\rho(a)$ approximates an unobservable probability density function $f_X(a)$ (see Section 28.2.8 on page 472) using a set of sample data A.

$$\rho(a) \approx f_X(a) \tag{28.276}$$

$$\rho : A \to \mathbb{R}^+ \tag{28.277}$$

Histograms

TODO

The kth Nearest Neighbor Method

Definition 28.63 (kth Nearest Neighbor Distance). The kth nearest neighbor distance function $\text{dist}^\rho_{nn,k}$denotes the distance of one element a to its kth nearest neighbor in the set of all elements A. It relies on a distance measure (here called dist) to compute the element distances. See Section 29.1 on page 537 for more details on distance measures.

$$\text{dist}^\rho_{nn,k}(\text{dist}, a, A) = \text{dist}(a, a_k) \ : \ |\forall b \in A : \text{dist}(a,b) < \text{dist}(a,a_k)| = k-1 \tag{28.278}$$

[69] http://en.wikipedia.org/wiki/Density_estimation [accessed 2007-07-03]

Using the kth nearest neighbor method [1879], the probability density function of an element a is estimated by its distance to its kth nearest neighbor a_k in the test set A (with $k < |A|$). Most often, the kth nearest neighbor distance measure internally uses the Euclidian distance measure $\text{dist}_{eucl} \equiv \text{dist}_{n,2}$ (see Definition 29.8 on page 538), but theoretically any other one of the distance measures presented in Section 29.1 could also be applied. Normally, k is chosen to be $\sqrt{|A|}$

$$\rho_{nn,k}(a, A) = \frac{k}{2\,|A|\,\text{dist}^{\rho}_{nn,k}(\text{dist}, a, A)} \tag{28.279}$$

Crowding Distance

Crowding distance [542] treats every element $a \in A$ as n-dimensional vector (where each dimension will represent an objective subject to optimization in the context of this book). The crowding distance is not a distance measure as its name may suggest, but a base for a density estimate. When computing the crowding distance of an element a we consider every single dimension i of the element a separately. For each of its dimensions, we determine the nearest neighbor to the left a^l and the nearest neighbor to the right A^r. The crowding distance of the element a in the dimension i is then $a^r_i - a^l_i$, the distance of the (objective) values of the right and left neighbors of a in the dimension i. This distance is normalized so that the maximum crowding distance of all elements in A in any dimension is 1. If an element has no left or no right neighbor in this dimension, meaning that it is situated on either end of the spectrum represented by all elements in the sample A, its crowding distance in the dimension is also set to 1.

The original source [542] does not mention normalization explicitly and sets the crowding distance of edge elements to ∞, which is both problematic. If no normalization is performed, dimensions with large crowding distances will outweigh those with smaller values – they will play no role in the crowding density value finally computed. With normalization, each dimension has the same weight. If the crowding distance of edge elements is set to ∞, they will have a very outstanding position in A which could influence processes relying on the crowding distance in a very strong way.

The total crowding distance of an element a is the sum of its distance values corresponding to each dimension. Algorithm 28.1 on the following page computes a function $\text{dist}^{\rho}_{cr}(a, A)$ which relates each element a of the set A to its crowding distance. In this algorithm, we consider dist^{ρ}_{cr} to be some sort of lookup-table instead of a mathematical function. Therefore, we can build it iteratively by summing up the distance values dimension-wise. Since computing the crowding distance can be performed best by sorting the individuals according to their values in the single dimensions, we define the comparator function[70] cmp_iab as follows:

$$\text{cmp}_{cr,i}(a, b) = \begin{cases} -1 & \text{if } a_i < b_i \\ 0 & \text{if } a_i = b_i \\ 1 & \text{otherwise} \end{cases} \quad \forall a, b \in A,\ \forall i \in [0, |a|] \tag{28.280}$$

The crowding distance can now be used as density estimate whereas individuals with large crowding distance values are in a sparsely covered region while small values of dist^{ρ}_{cr} indicate dense portions of A. A density estimate derived from the crowding distance will therefore be inversely proportional to it. Hence, we define the density measure ρ_{cr} as the difference of 1 and $\text{dist}^{\rho}_{cr}(a, A)$ divided by the vector dimensions $n = |a|$, obtaining a value in $[0, 1]$ that is big if a is in crowded region and small if it is situated in a sparsely covered area of A. It should be noted that this density estimate is mathematically not fully sound since it only displays the crowding information.

$$\rho_{cr}(a, A) = 1 - \frac{\text{dist}^{\rho}_{cr}(a, A)}{n} \tag{28.281}$$

[70] Comparator functions where introduced in Definition 1.15 on page 38.

Algorithm 28.1: $\text{dist}^{\rho}_{cr}(\ldots, A) \longleftarrow \textit{computeCrowdingDistance}(a, A)$

Input: A: the set of sample data
Data: dd: a list used as store for the crowding distances of the single dimensions
Data: A_s: the list representation of A
Data: dim: the dimension counter
Data: j: the element counter
Data: max: the maximum crowding distance of the current dimension
Output: $\text{dist}^{\rho}_{cr}(\ldots, A)$: the crowding distance function

```
1  begin
2      dd ⟵ createList(len(A), 0)
3      dd[0] ⟵ 1
4      dd[len(A)−1] ⟵ 1
5      A_s ⟵ setToList(A)
6      dim ⟵ n
7      while dim > 0 do
8          A_s ⟵ sortList_a(A, cmp_dim)
9          max ⟵ 0
10         j ⟵ len(A) − 2
11         while j > 0 do
12             dd[j] ⟵ A_s[j+1]dim − A_s[j−1]dim
13             if dd[j] > max then max ⟵ dd[j]
14             j ⟵ j − 1
15         if max > 0 then
16             j ⟵ len(A) − 2
17             while j > 0 do
18                 dd[j] ⟵ dd[j] / max
19                 j ⟵ j − 1
20         j ⟵ len(A) − 1
21         while j ≥ 0 do
22             dist^ρ_cr(A_s[j], A) ⟵ dist^ρ_cr(A_s[j], A) + dd[j]
23             j ⟵ j − 1
24         dim ⟵ dim − 1
25     return dist^ρ_cr(..., A)
26 end
```

Parzen Window / Kernel Density Estimation

Another density estimation is the Parzen [1618] window method[71], also called kernel density estimation.

TODO

28.8 Statistical Tests

With statistical tests [874, 1866, 1878, 898, 1274, 478], it is possible to find out whether an alternative hypothesis H_1 about the distribution(s) from a set of measured data A is likely to be true. This is done by showing that the sampled data would very unlikely have

[71] http://en.wikipedia.org/wiki/Parzen_window [accessed 2007-07-03]

occurred if the opposite hypothesis, the null hypothesis H_0, holds. If we want to show, for instance, that two different settings for an evolutionary algorithm will probably lead to different solution qualities (H_1), we assume that the distributions of the objective values of the solution candidates returned by them are equal (H_0). Then, we run the two evolutionary algorithms multiple times and measure the outcome, i. e., obtain A. Based on A, we can estimate the probability α with which the two different sets of measurements (the samples) would have occurred if H_0 was true. In the case that this probability is very low, let's say $\alpha < 5\%$, H_0 can be rejected (with 5% probability of making a type 1 error) and H_1 is likely to hold. Otherwise, we would expect H_0 to hold and reject H_1.

Neyman and Pearson [1522, 1523] distinguish two classes of errors[72] that can be made when performing hypothesis tests:

Definition 28.64 (Type 1 Error). A *type 1 error* (α error, false positive) is the rejection of a correct null hypothesis H_0, i. e., the acceptance of a wrong alternative hypothesis H_1. Type 1 errors are made with probability α.

Definition 28.65 (Type 2 Error). A *type 2 error* (β error, false negative) is the acceptance of a wrong null hypothesis H_0, i. e., the rejection of a correct alternative hypothesis H_1. Type 2 errors are made with probability β.

Definition 28.66 (Power). The (statistical) *power*[73] of a statistical test is the probability of rejecting a false null hypothesis H_0. Therefore, the power equals $1 - \beta$.

A few basic principles for testing should be mentioned before going more into detail:

1. The more samples we have, the better the quality and significance of the conclusions that we can make by testing. An arithmetic mean of the runtime 7s is certainly more significant when being derived from 1000 runs of certain algorithm than from the sample set $A = \{9\text{s}, 5\text{s}\}\ldots$
2. The more assumptions that we can make about the sampled probability distribution, the powerful will the tests be that are available.
3. Wrong assumptions, falsely carried out measurements, or other misconduct will nullify all results and efforts put into testing.

In the following, we will discuss multiple methods for hypothesis testing. We can distinguish between tests based on paired samples and those for independent populations. In Table 28.14, we have illustrated an example for the former, where pairs of elements (a, b) are drawn from two different populations. Table 28.15 contains two independent samples a and b with a different number of elements ($n_a = 6 \neq n_b = 8$).

28.8.1 Non-Parametric Tests

All previously discussed estimation or testing methods have one thing in common: we have to know or to assume the type of distribution which drives the sampled process. When this distribution is known, everything is sweet. If we have to assume the distribution, we may make an error. The possibility of an error obviously renders the probabilities that we define for the tests or confidence intervals more or less useless. Additionally, there are cases where we either have no idea at all about the distribution in question or where it is questionable whether one of the distributions known to us (see, for instance, Section 28.4 and Section 28.5 for reference) does fit to the behavior of the observed process sufficiently good.

Non-parametric statistics[74] [1878, 1866, 252] is the branch of statistics focusing on the group of methods that make only extremely few assumptions about the distribution from

[72] http://en.wikipedia.org/wiki/Type_I_and_type_II_errors [accessed 2008-08-15]
[73] http://en.wikipedia.org/wiki/Statistical_power [accessed 2008-08-15]
[74] http://en.wikipedia.org/wiki/Non-parametric_statistics [accessed 2008-08-15]

Row	a	b	$d = b - a$	Sign	Rank $\lvert r\rvert$	Rank r
1.	2	10	+8	+	13	13
2.	3	4	+1	+	2	2
3.	6	10	+4	+	10	10
4.	4	6	+2	+	6	6
5.	6	11	+5	+	11	11
6.	5	6	+1	+	2	2
7.	4	11	+7	+	12	12
8.	9	6	−3	−	9	−9
9.	10	12	+2	+	6	6
10.	8	8	0	=	–	–
11.	6	8	+2	+	6	6
12.	7	6	−1	−	2	−2
13.	4	4	0	=	–	–
14.	4	6	+2	+	6	6
15.	9	7	−2	−	6	−6
	$\sum a_i = 87$	$\sum b_i = 115$	$D = \sum d_i = 28$		$R = \sum r_i = 57$	
$\text{med}(a) = 6$; $\overline{a} = 5.8$						
$\text{med}(b) = 7$; $\overline{b} = 7.67$						

Table 28.14: Example for paired samples (a, b).

Row	a	b	Ranks r_a	Ranks r_b
1.	2		1.5	
2.		2		1.5
3.	3		4.0	
4.	3		4.0	
5.	3		4.0	
6.	4		6.0	
7.		5		8.5
8.	5		8.5	
9.		5		8.5
10.		5		8.5
11.		6		11.5
12.		6		11.5
13.		7		13.5
14.		7		13.5
	$\text{med}(a) = 3$	$\text{med}(b) = 5.5$	$R_a = 28$	$R_b = 77$
	$n_a = 6$	$n_b = 8$		

Table 28.15: Example for unpaired samples.

which the data has been sampled. Many of the density estimation methods which we will discuss in Section 28.7.4 belong to this group, for instance. Here, we will concentrate on non-parametric tests which allow us to verify hypothesis on data samples with unknown underlying distribution.

Sign Test

The sign test[75] [874, 1878] is used for checking whether the differences in the medians of paired samples from *continuous* distributions are significant. This test is especially useful, for instance, when we have before-after or with-and-without-types of sample pairs (a, b) and

[75] http://en.wikipedia.org/wiki/Sign_test [accessed 2008-08-15]

can (only) measure the changes between them. The null hypothesis H_0 is that there is no difference between the medians of the distributions generating the elements a and b. The alternative hypothesis H_1 is that such a difference exists.

An example for this situation has been given in Table 28.14 on the facing page. The first step of applying the sign test is to reduce the measurement pairs (a, b) to + (if $a < b$), = (if $a = b$), and to - (if $a > b$), as done in the fifth column of Table 28.14. Then, the number of + and - in A' are counted.

$$n^+ = |\{a \in A : a = +\}| \tag{28.282}$$

$$n^- = |\{a \in A : a = -\}| \tag{28.283}$$

In Table 28.14, $n^+ = 10$ and $n^- = 3$. In the following, the samples with = are ignored, setting the total of "interesting" samples to $n = 13$. The motivation is that if the underlying distributions are continuous, the chance of drawing two similar elements $a_i = b_i$ (with difference $d_i = 0$) is also 0 and such measurements thus result from imprecision. On one hand, this makes complete sense, since these samples would have been either + or - with more precise measurement equipment and now we cannot determine to which group they belong anymore. On the other hand, by simply discarding these samples, we also discard information which supports the null hyporthesis H_0. This is a weakness of the sign test.

In the ideal case if H_0 holds, i. e., if the medians of the distributions of a and b are equal, the probability that one row in the Table 28.14 contains a + is exactly the same that it would contain a -, both are 0.5. Rarely one will encounter such an ideal situation in the data samples, so we need to find out how significant the 10 : 3 ratio in our sample is. With the binomial distribution (discussed in Section 28.4.3), we can determine the probability that n^+ (resp. n^-) or more extreme numbers of + (-) would occur under H_0.

$$\begin{aligned}\alpha = P\left(x \leq \min\left\{n^+, n^-\right\}\right) + P\left(x \geq \max\left\{n^+, n^-\right\}\right) = \\ 2P\left(x \leq \min\left\{n^+, n^-\right\}\right) = 2\sum_{i=0}^{\min\left\{n^+, n^-\right\}} \binom{n}{i} 0.5^n = \\ 2P\left(x \geq \max\left\{n^+, n^-\right\}\right) = 2\sum_{i=\max\{n^+, n^-\}}^{n} \binom{n}{i} 0.5^n\end{aligned} \tag{28.284}$$

Notice that this corresponds to computing a two-sided probability with the CFG of the binomial distribution (see Equation 28.137) with the parameters n and $p = 0.5$. In our example, we would compute:

$$\alpha = 2\sum_{i=0}^{3} \binom{13}{i} 0.5^{13} \approx 0.0923 \tag{28.285}$$

Data samples at least as extreme as our measurements could occur with a probability of approximately 9%. On a significance level of 5%, we cannot reject H_0. Hence, there is not enough information to believe in H_1 according to the sign test.

Randomization Test

Randomization tests[76] for equality of expected values have first been suggested by Fisher [683] in 1936 [252, 619, 1977]. They not only take into consideration the signs of the differences, but also the sum D of differences themselves. In our example from Table 28.14, the total difference D between the a and b is 28.

The null hypothesis H_0 is again that all samples have been drawn from the same population and thus, their expected values are the same. For each single pair (a_i, b_i) in Table 28.14,

[76] http://en.wikipedia.org/wiki/Randomization_test [accessed 2008-08-20]

we have computed the difference $d_i = b_i - a_i$ in the fourth column. If H_0 holds, the probability that we would measure d_i is exactly the same as for measuring $-d_i$, since drawing the pair (a_i, b_i) is as same as probable that drawing (b_i, a_i).

Leaving the zero differences ($d_i = 0$) out of consideration, we obtain the following combinatorial considerations from [252]: If our measured data consists of only one pair (a_1, b_1), under H_0, $2^1 = 2$ differences are possible: ($d_1 = b_1 - a_1$ or $d_1 = a_1 - b_1$) and both have the same probability $1/2$. For $n = 2$ pairs, the difference signs can occur in $2^2 = 4$ ways, $\{--, -+, +-, ++\}$, each having probability $1/4$. For $n = 3$, $2^3 = 8$ possible sign configurations with probability $1/8$ can emerge $\{---, --+, -+-, -++, +--, +-+, ++-, +++\}$. Generally, if we leave the pairs intact and exchange only their members, there are 2^n possible +/- arrangements for n pairs.

For all these arrangements, we compute the absolute value of the corresponding total difference D' and count the number n_e of differences that are more extreme than the absolute value of D. "Extreme" means either larger or smaller than D, depending on the sample distribution. The absolute values of the differences are used since the test of H_0 is basically a two-sided test. The probability α of the observed measurements under H_0 can then be estimated with

$$n_e = \min\left\{\left|\{D' : D' \geq |D|\}\right|, \left|\{D' : D' \leq |D|\}\right|\right\} \tag{28.286}$$

$$\alpha = \frac{n_e}{2^n} \tag{28.287}$$

In other words, the more often differences more extreme than the initial D occur, the more evidence is given for H_0. If, on the other hand, only very few configurations with differences as extreme as D exist, α becomes very small and we can reject H_0.

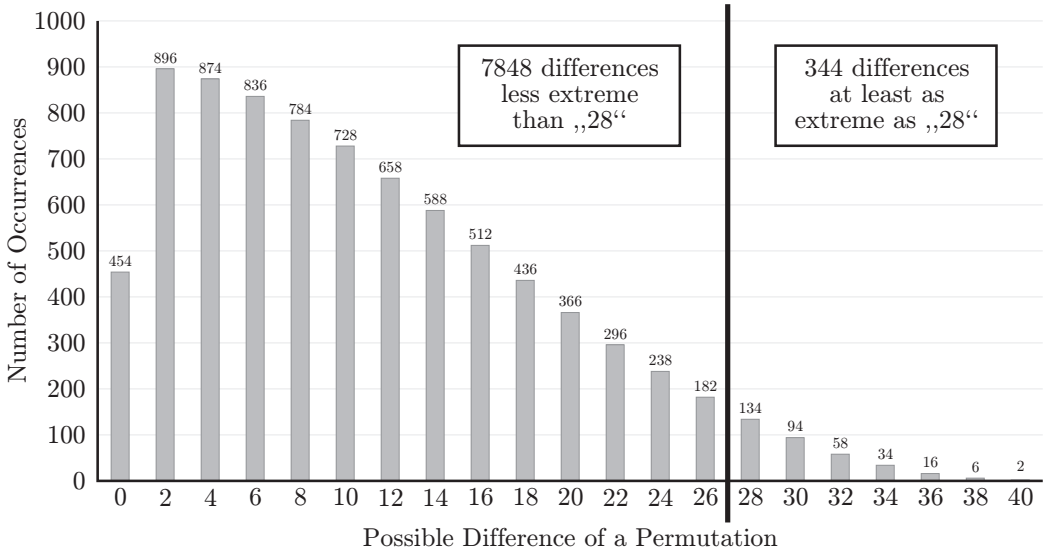


Figure 28.19: The randomization test applied to the example from Table 28.14.

Let us apply this procedure to the example given in Table 28.14. There are $n = 13$ pairs with non-zero difference in our samples, leading to a total of 8192 configurations. You can find these 8192 values illustrated in a histogram in Figure 28.19. From this histogram, we can see that there are 344 permutations with a difference at least as extreme as the sampled difference. Hence, $\alpha = 43/1024 \approx 0.042 = 4.2\%$ and, under a significance of 5%, the null hypothesis H_0 can be rejected.

The randomization test has a higher power than the sign test, but comes with the additional assumption that the measured sample represents the population(s) and the underlying distribution(s) sufficiently well and that the samples are pair-wise independent. The populations do not necessarily need to be homogeneous. Different than the sign test, the randomization test does not require the sampled distributions to be continuous and, thus, zero differences are possible. They would play no role in the computation since the same results with come out with and without them [252]. Leaving them away makes sense since it reduces the number of combinations which have to be tested. If we expect a high probability of outliers, the randomization test is maybe not the method of choice and we would prefer the sign test. Notice furthermore that computing all possible differences may become computationally intense for $n > 128$ data samples...

Signed Rank Test

Wilcoxon's signed rank test[77] [2220] basically works exactly the same as the randomization test except that it replaces the difference sums with difference ranks. The null hypothesis H_0 is that the average rank of the samples in the pairs are equal. The alternative hypothesis H_1 is that there is a difference. The ranks are computed as follows:

1. First, the differences d_i between the elements b_i and a_i of the sample pairs (a_i, b_i) have to be determined (the fourth column in Table 28.14).
2. The zero differences ($d_i = 0$) are discarded and only the remaining n samples are considered in the test.
3. The absolute values $|d_i|$ of these differences are sorted in ascending order.
4. Each absolute value d_i is assigned a rank $|r_i|$ corresponding to its position in this list. Rows $i, i+1, \ldots, i+m$ with equal absolute differences $|d_i| = |d_{i+1}| = \cdots = |d_{i+m}|$ share the same absolute rank $|r_i| = |r_{i+1}| = .. = |r_{i+m}| = \frac{i+(i+1)+\cdots+(i+m)}{m+1} = \frac{m}{2} + i$ which is determined by averaging, fractional ranks such as 3.5 are possible. The difference 1, for instance, occurs three times in the second to last (unsorted) column of Table 28.14: $|d_2| = |d_6| = |d_{12}| = 1$. All three rows received the same absolute rank $|r_2| = |r_6| = |r_{12}| = \frac{1+2+3}{2} = 2$.
5. The sign that has been stripped from the differences d_i is re-attached to the ranks, i. e., $r_i = |r_i| * \text{sign}(d_i)$. We have applied this ranking scheme to the example in Table 28.14 in the last table column.

The rank sum R of the initial sample is determined by adding up all the signed ranks r_i of the in-pair differences. Now the distribution of the absolute values of the 2^n possible rank sums R' are computed with the same method with which the distribution of possible difference sums is determined in the randomization test (see Section 28.8.1). Amongst these, we count the number n_e of rank sums at least as extreme as R and the probability α that the ranks samples would have been measured then is determined with Equation 28.289.

$$n_e = \min\left\{\left|\{R' : R' \geq |R|\}\right|, \left|\{R' : R' \leq |R|\}\right|\right\} \tag{28.288}$$

$$\alpha = \frac{n_e}{2^n} \tag{28.289}$$

When we apply this procedure to our example from Table 28.14 and illustrate the histogram the absolute rank sums in Figure 28.20 (analogously to the histogram of possible absolute difference sums in Figure 28.19). The rank sum of the original sample is 57, as noted in Table 28.14. In our example, we have $n = 13$ non-zero differences and there are a total of $2^n = 8192$ possible configurations. Amongst these, ranks as high as 57 and higher occur $n_e = 372$ times, leading to $\alpha = {}^{93}/_{2048} \approx 0.045$. Therefore, the set of ranks which we have measured would have a probability of 4.5% if H_0 would holds. In other words, at a significance level of 5%, we can reject H_0 and assume the alternative hypothesis H_1.

[77] http://en.wikipedia.org/wiki/Wilcoxon_signed-rank_test [accessed 2008-08-21]

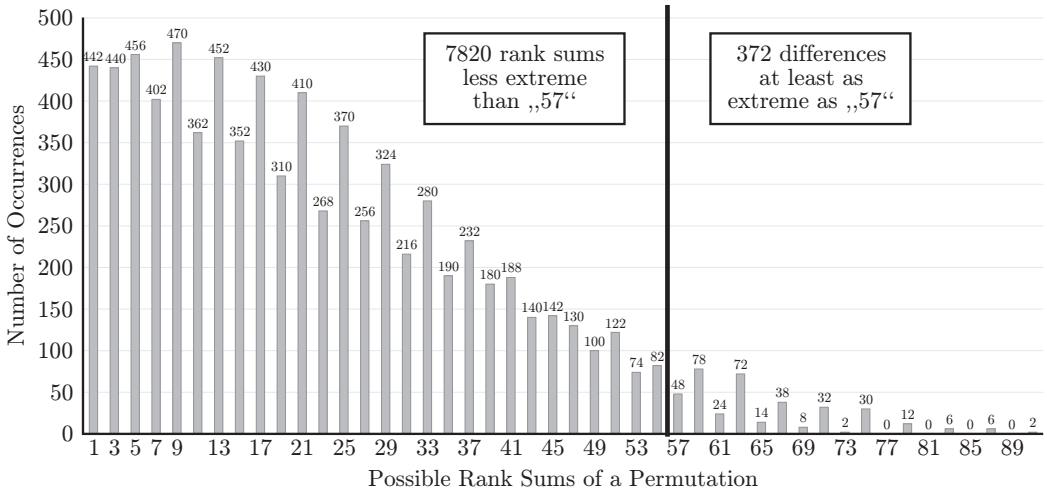


Figure 28.20: The histogram of possible absolute rang sums in Table 28.14.

By neither weighting every difference the same nor giving outliers too big of a chance to influence, the outcome of the test, the signed rank test lies somewhat between the sign and the randomization test [252]. When applying the signed rank test, we do not need to assume that the sampled data represent the true distributions with high precision. At first glance, this test has the same drawback as the randomization test: in order to get to the $\alpha = {}^{93}/_{2048}$, we need to compute all the $2^n = 8192$ possible rank combinations (or, at least half of them). For larger n, this will not work properly – $n = 32$, for instance 4 294 967 296 iterations, and for $n = 100$, we would have to check $1.3 * 10^{30}$ combinations.

Different from the randomization test though, the rank numbers are finite and we know, for instance, their maximum value $\hat{r} = \frac{n(n+1)}{2}$. For the signed rank approach, there exist tables (such as Table 28.16), where the corresponding α values are listed. In order to use them, we use another (equivalent) approach for computing the rank sums [1076]. First, we proceed exactly as before until we have computed all absolute ranks $|r_i|$ of the absolute differences $|d_i|$ (see the sixth column in Table 28.14). Similar to the sign test, we compute the sum of ranks r^+ belonging to the positive and the sum of ranks r^- belonging to the negative differences.

$$r^+ = \sum_{i=1}^{n} \begin{cases} |r_i| & \text{if } d_i > 0 \\ 0 & \text{otherwise} \end{cases} \tag{28.290}$$

$$r^- = \sum_{i=1}^{n} \begin{cases} |r_i| & \text{if } d_i < 0 \\ 0 & \text{otherwise} \end{cases} \tag{28.291}$$

$$\check{r} = \min\left\{r^+, r^-\right\} \tag{28.292}$$

In our example, $r^+ = 74$, $r^- = 17$ (notice that $r^+ - r^- = R = 57$ and $r^+ + r^- = 0.5 * n * (n+1) = 91$), and, thus, $\check{r} = 17$. In the following, three tables with the distribution of the Wilcoxon rank values for *two-sided* hypothesis can be found. The first two (Table 28.16 and Table 28.17) contain the critical values of $\check{r}$ for certain α whereas the third table (Table 28.18) lists the exact α values for various n and $\check{r}$. When we want to find the significance level with which we can reject H_0, we will look up the row with $n = 13$ Table 28.16 and search the first cell which is greater or equal than $\check{r}$. We find that $\check{r} \leq 17 \Rightarrow \alpha \leq 0.05$. (If $\check{r}$ was $16, 15, .., 13$, we would assume the same α and for $\check{r} = 12$, $\alpha \leq 0.02$ hold.) We had determined the precise α in our case to be 4.5%, so this fits to the table value. According to Table 28.16, we could reject H_0 with a significance of 5%. Notice that an $\emptyset$ in a cell of Table 28.16 or Table 28.17 means that not value of $\check{r}$ exists for which the given significance level is fulfilled.

Table 28.18 lists the precise values of α for $n \in 4..30$. We can again look up our example by first finding the section dealing with $n = 13$. There, we can find the α corresponding to $\check{r}$. In each row of this section, ten values of $\check{r}$ are listed. The first cell of row X stands for $\check{r} = X$, the second one for $\check{r} = X + 1$, and so on. Since $\check{r} = 17$, we go to the eight column to the second row (the row that starts with **10**) and find $\alpha = 0.047\,86$. This value is equal to $^{392}/_{8192}$, whereas the exact value that we have computed is smaller: $(^{372}/_{8192})$. This difference results from the fact that in our example, there are some sample differences which share the same rank $|r|$ (row 9 and 11 in Table 28.14, for example). The table is only precise for unique ranks. Basically, shared sums can lead to increases as well as decreases of α. For example, if *all* ranks were 7, there are only 184 combinations for $n = 13$ where the ranks are at least as extreme as 17 (α would be 2.2%) and if the 13 ranks were $(3, 3, 3, 3, 3, 7, 7, 7, 11, 11, 11, 11, 11)$, there would be 416 combinations, i. e., $\alpha = 0.0507$. Thus, the tables can only be used correctly if most rank numbers are indeed unique.

n\α	.2	.1	.05	.02	.01	.005	.002
1	∅	∅	∅	∅	∅	∅	∅
2	∅	∅	∅	∅	∅	∅	∅
3	∅	∅	∅	∅	∅	∅	∅
4	0	∅	∅	∅	∅	∅	∅
5	2	0	∅	∅	∅	∅	∅
6	3	2	0	∅	∅	∅	∅
7	5	3	2	0	∅	∅	∅
8	8	5	3	1	0	∅	∅
9	8	5	3	1	0	∅	∅
10	14	10	8	5	3	1	0
11	17	13	10	7	5	3	1
12	21	17	13	9	7	5	2
13	26	21	17	12	9	7	4
14	31	25	21	15	12	9	6
15	36	30	25	19	15	12	8
16	42	35	29	23	19	15	11
17	48	41	34	27	23	19	14
18	55	47	40	32	27	23	18
19	62	53	46	37	32	27	21
20	69	60	52	43	37	32	26
21	77	67	58	49	42	37	30
22	86	75	65	55	48	42	35
23	94	83	73	62	54	48	40
24	104	91	81	69	61	54	45
25	113	100	89	76	68	60	51
26	124	110	98	84	75	67	58
27	134	119	107	92	83	74	64
28	145	130	116	101	91	82	71
29	157	140	126	110	100	90	79
30	169	151	137	120	109	98	86
31	181	163	147	130	118	107	94
32	194	175	159	140	128	116	103
33	207	187	170	151	138	126	112
34	221	200	182	162	148	136	121
35	235	213	195	173	159	146	131
36	250	227	208	185	171	157	141
37	265	241	221	198	182	168	151
38	281	256	235	211	194	180	162
39	297	271	249	224	207	192	173
40	313	286	264	238	220	204	185
41	330	302	279	252	233	217	197
42	348	319	294	266	247	230	209
43	365	336	310	281	261	244	222
44	384	353	327	296	276	258	235
45	402	371	343	312	291	272	249
46	422	389	361	328	307	287	263
47	441	407	378	345	322	302	277
48	462	426	396	362	339	318	292
49	482	446	415	379	355	334	307
50	503	466	434	397	373	350	323

n\α	.2	.1	.05	.02	.01	.005	.002
51	525	486	453	416	390	367	339
52	547	507	473	434	408	384	355
53	569	529	494	454	427	402	372
54	592	550	514	473	445	420	389
55	615	573	536	493	465	439	407
56	639	595	557	514	484	457	425
57	664	618	579	535	504	477	443
58	688	642	602	556	525	497	462
59	714	666	625	578	546	517	482
60	739	690	648	600	567	537	501
61	765	715	672	623	589	558	521
62	792	741	697	646	611	580	542
63	819	767	721	669	634	602	563
64	847	793	747	693	657	624	584
65	875	820	772	718	681	647	606
66	903	847	798	742	705	670	628
67	932	875	825	768	729	694	651
68	962	903	852	793	754	718	674
69	992	931	879	819	779	742	697
70	1022	960	907	846	805	767	721
71	1053	990	936	873	831	792	745
72	1084	1020	964	901	858	818	770
73	1116	1050	994	928	884	844	795
74	1148	1081	1023	957	912	871	821
75	1181	1112	1053	986	940	898	847
76	1214	1144	1084	1015	968	925	873
77	1247	1176	1115	1044	997	953	900
78	1282	1209	1147	1075	1026	981	927
79	1316	1242	1179	1105	1056	1010	955
80	1351	1276	1211	1136	1086	1039	983
81	1387	1310	1244	1168	1116	1069	1011
82	1423	1345	1277	1200	1147	1099	1040
83	1459	1380	1311	1232	1178	1129	1070
84	1496	1415	1345	1265	1210	1160	1099
85	1533	1451	1380	1298	1242	1191	1130
86	1571	1487	1415	1332	1275	1223	1160
87	1609	1524	1451	1366	1308	1255	1191
88	1648	1561	1487	1400	1342	1288	1223
89	1688	1599	1523	1435	1376	1321	1255
90	1727	1638	1560	1471	1410	1355	1287
91	1767	1676	1597	1507	1445	1389	1320
92	1808	1715	1635	1543	1480	1423	1353
93	1849	1755	1674	1580	1516	1458	1387
94	1891	1795	1712	1617	1552	1493	1421
95	1933	1836	1752	1655	1589	1529	1455
96	1976	1877	1791	1693	1626	1565	1490
97	2019	1918	1832	1731	1664	1601	1525
98	2062	1960	1872	1770	1702	1638	1561
99	2106	2003	1913	1810	1740	1676	1598
100	2151	2045	1955	1850	1779	1714	1634

Table 28.16: Wilcoxon's two-sided signed-rank distribution $2W(\alpha, n)$ for $n \in 1..100$ (from Junge [1076]).

n\α	.2	.1	.05	.02	.01	.005	.002
101	2195	2089	1997	1890	1818	1752	1671
102	2241	2133	2039	1931	1858	1791	1709
103	2287	2177	2082	1972	1898	1830	1747
104	2333	2222	2125	2014	1939	1870	1785
105	2380	2267	2169	2056	1980	1910	1824
106	2427	2312	2213	2099	2022	1950	1863
107	2475	2359	2258	2142	2063	1991	1903
108	2523	2405	2303	2186	2106	2032	1943
109	2572	2452	2349	2230	2149	2074	1984
110	2621	2500	2395	2274	2192	2117	2025
111	2671	2548	2442	2319	2236	2159	2066
112	2721	2596	2489	2364	2280	2202	2108
113	2771	2645	2536	2410	2325	2246	2150
114	2822	2695	2584	2456	2370	2290	2193
115	2874	2744	2632	2503	2415	2335	2236
116	2926	2795	2681	2550	2461	2380	2280
117	2978	2846	2731	2598	2508	2425	2324
118	3031	2897	2780	2646	2555	2471	2368
119	3085	2948	2831	2694	2602	2517	2413
120	3139	3001	2881	2743	2650	2564	2459
121	3193	3053	2933	2793	2698	2611	2505
122	3248	3106	2984	2843	2747	2658	2551
123	3303	3160	3036	2893	2796	2707	2598
124	3359	3214	3089	2944	2846	2755	2645
125	3416	3269	3142	2995	2896	2804	2692
126	3472	3324	3195	3047	2946	2853	2740
127	3530	3379	3249	3099	2997	2903	2789
128	3587	3435	3304	3152	3049	2953	2838
129	3645	3492	3359	3205	3100	3004	2887
130	3704	3548	3414	3258	3153	3055	2937
131	3763	3606	3470	3312	3205	3107	2987
132	3823	3664	3526	3367	3259	3159	3038
133	3883	3722	3583	3421	3312	3212	3089
134	3944	3781	3640	3477	3366	3264	3140
135	4005	3840	3697	3533	3421	3318	3192
136	4066	3900	3756	3589	3476	3372	3245
137	4128	3960	3814	3646	3531	3426	3298
138	4191	4020	3873	3703	3587	3481	3351
139	4254	4081	3933	3760	3644	3536	3405
140	4317	4143	3993	3819	3701	3592	3459
141	4381	4205	4053	3877	3758	3648	3514
142	4445	4268	4114	3936	3816	3704	3569
143	4510	4331	4175	3996	3874	3761	3624
144	4575	4394	4237	4055	3932	3819	3680
145	4641	4458	4299	4116	3991	3877	3737
146	4708	4522	4362	4177	4051	3935	3793
147	4774	4587	4425	4238	4111	3994	3851
148	4842	4652	4489	4300	4171	4053	3908
149	4909	4718	4553	4362	4232	4113	3967
150	4978	4785	4618	4425	4294	4173	4025

n\α	.2	.1	.05	.02	.01	.005	.002
151	5046	4851	4683	4488	4355	4233	4084
152	5115	4919	4748	4551	4418	4294	4144
153	5185	4986	4814	4615	4480	4356	4204
154	5255	5054	4881	4680	4544	4418	4264
155	5326	5123	4948	4745	4607	4480	4325
156	5397	5192	5015	4810	4671	4543	4387
157	5468	5262	5083	4876	4736	4606	4448
158	5540	5332	5151	4943	4801	4670	4511
159	5613	5402	5220	5009	4866	4734	4573
160	5686	5473	5289	5077	4932	4799	4636
161	5759	5545	5359	5144	4999	4864	4700
162	5833	5617	5429	5213	5065	4930	4764
163	5908	5689	5500	5281	5133	4996	4828
164	5982	5762	5571	5350	5200	5062	4893
165	6058	5835	5643	5420	5269	5129	4959
166	6134	5909	5715	5490	5337	5196	5024
167	6210	5983	5787	5560	5406	5264	5091
168	6287	6058	5860	5631	5476	5332	5157
169	6364	6133	5934	5703	5546	5401	5224
170	6442	6209	6008	5775	5616	5470	5292
171	6520	6285	6082	5847	5687	5540	5360
172	6599	6362	6157	5920	5759	5610	5429
173	6678	6439	6232	5993	5831	5681	5497
174	6758	6517	6308	6067	5903	5752	5567
175	6838	6595	6385	6141	5976	5823	5637
176	6919	6673	6461	6216	6049	5895	5707
177	7000	6752	6538	6291	6123	5967	5778
178	7081	6832	6616	6366	6197	6040	5849
179	7163	6912	6694	6442	6271	6113	5921
180	7246	6992	6773	6519	6346	6187	5993
181	7329	7073	6852	6596	6422	6261	6065
182	7412	7155	6932	6673	6498	6336	6138
183	7496	7236	7012	6751	6574	6411	6212
184	7581	7319	7092	6829	6651	6486	6285
185	7666	7402	7173	6908	6728	6562	6360
186	7751	7485	7254	6987	6806	6639	6435
187	7837	7569	7336	7067	6884	6716	6510
188	7924	7653	7419	7147	6963	6793	6585
189	8010	7738	7502	7228	7042	6871	6662
190	8098	7823	7585	7309	7122	6949	6738
191	8186	7908	7669	7391	7202	7028	6815
192	8274	7995	7753	7473	7283	7107	6893
193	8363	8081	7838	7555	7364	7187	6971
194	8452	8168	7923	7638	7445	7267	7049
195	8542	8256	8008	7722	7527	7347	7128
196	8632	8344	8095	7806	7610	7428	7207
197	8723	8432	8181	7890	7692	7510	7287
198	8814	8521	8268	7975	7776	7592	7367
199	8905	8611	8356	8060	7859	7674	7448
200	8998	8701	8444	8146	7944	7757	7529

Table 28.17: Wilcoxon's two-sided signed-rank distribution $2W(\alpha, n)$ for $n \in 101..200$ (from Junge [1076]).

$\check{r}$	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9
Precise α values for $n = 4$										
0	0.125									
Precise α values for $n = 5$										
0	0.0625	0.125	0.1875							
Precise α values for $n = 6$										
0	0.03126	0.0625	0.09376	0.15626						
Precise α values for $n = 7$										
0	0.015626	0.03126	0.04688	0.07812	0.10938	0.15626				
Precise α values for $n = 8$										
0	0.007812	0.015626	0.02344	0.03906	0.05468	0.07812	0.10938	0.14844	0.19532	
Precise α values for $n = 9$										
0	0.003906	0.007812	0.011718	0.019532	0.02734	0.03906	0.05468	0.07422	0.09766	0.1289
10	0.16406									
Precise α values for $n = 10$										
0	0.0019532	0.003906	0.00586	0.009766	0.013672	0.019532	0.02734	0.0371	0.04882	0.06446
10	0.08398	0.10546	0.13086	0.16016	0.19336					
Precise α values for $n = 11$										
0	9.766E-4	0.0019532	0.00293	0.004882	0.006836	0.009766	0.013672	0.018554	0.02442	0.03222
10	0.042	0.05372	0.06738	0.083	0.10156	0.12304	0.14746	0.1748		
Precise α values for $n = 12$										
0	4.882E-4	9.766E-4	0.0014648	0.002442	0.003418	0.004882	0.006836	0.009278	0.012208	0.016114
10	0.021	0.02686	0.03418	0.04248	0.05224	0.06396	0.07714	0.09228	0.10986	0.1294
20	0.15136	0.17626								
Precise α values for $n = 13$										
0	2.442E-4	4.882E-4	7.324E-4	0.0012208	0.001709	0.002442	0.003418	0.004638	0.006104	0.008056
10	0.010498	0.013428	0.01709	0.02148	0.02662	0.03272	0.0398	0.04786	0.05738	0.06812
20	0.08032	0.09424	0.10986	0.1272	0.14648	0.16772	0.19092			
Precise α values for $n = 14$										
0	1.2208E-4	2.442E-4	3.662E-4	6.104E-4	8.544E-4	0.0012208	0.001709	0.00232	0.003052	0.004028
10	0.00525	0.006714	0.008544	0.010742	0.013428	0.016602	0.02026	0.02454	0.02954	0.03528
20	0.04188	0.04944	0.05798	0.06762	0.0785	0.09058	0.104	0.1189	0.13526	0.15308
30	0.1726	0.19372								
Precise α values for $n = 15$										
0	6.104E-5	1.2208E-4	1.831E-4	3.052E-4	4.272E-4	6.104E-4	8.544E-4	0.0011596	0.0015258	0.002014
10	0.002624	0.003356	0.004272	0.005372	0.006714	0.008362	0.010254	0.012452	0.015076	0.018066
20	0.02154	0.02558	0.03016	0.03534	0.04126	0.04792	0.05536	0.06372	0.073	0.08326
30	0.0946	0.107	0.12054	0.13538	0.15142	0.16882	0.18762			
Precise α values for $n = 16$										
0	3.052E-5	6.104E-5	9.156E-5	1.5258E-4	2.136E-4	3.052E-4	4.272E-4	5.798E-4	7.63E-4	0.001007
10	0.0013122	0.0016784	0.002136	0.002686	0.003356	0.00418	0.005158	0.006286	0.00763	0.009186
20	0.010986	0.013092	0.015502	0.01825	0.0214	0.02496	0.029	0.03354	0.03864	0.04432
30	0.05066	0.05768	0.0654	0.07392	0.08326	0.09344	0.10458	0.11666	0.12974	0.14386
40	0.15906	0.17536	0.19282							
Precise α values for $n = 17$										
0	1.5258E-5	3.052E-5	4.578E-5	7.63E-5	1.0682E-4	1.5258E-4	2.136E-4	2.9E-4	3.814E-4	5.036E-4

10	6.562E-4	8.392E-4	0.0010682	0.0013428	0.0016784	0.00209	0.002578	0.003158	0.003846	0.004638
20	0.00557	0.006652	0.007904	0.009338	0.010986	0.012864	0.015	0.017426	0.02016	0.02322
30	0.02668	0.03052	0.0348	0.03954	0.04476	0.05054	0.05688	0.06382	0.07142	0.07968
40	0.08866	0.09838	0.10888	0.1202	0.13236	0.14544	0.15938	0.17426	0.1901	
Precise α values for $n = 18$										
0	7.63E-6	1.5258E-5	2.288E-5	3.814E-5	5.34E-5	7.63E-5	1.0682E-4	1.4496E-4	1.9074E-4	2.518E-4
10	3.28E-4	4.196E-4	5.34E-4	6.714E-4	8.392E-4	0.0010452	0.0012894	0.0015792	0.0019302	0.002334
20	0.002808	0.003364	0.004006	0.004746	0.0056	0.006576	0.00769	0.008964	0.010406	0.012032
30	0.01387	0.01593	0.018234	0.02082	0.02368	0.02684	0.03036	0.03424	0.0385	0.04316
40	0.04828	0.05386	0.05994	0.06654	0.07368	0.08142	0.08976	0.09874	0.10838	0.1187
50	0.12974	0.14152	0.15404	0.16736	0.18146	0.19638				
Precise α values for $n = 19$										
0	3.814E-6	7.63E-6	1.1444E-5	1.9074E-5	2.67E-5	3.814E-5	5.34E-5	7.248E-5	9.536E-5	1.2588E-4
10	1.6404E-4	2.098E-4	2.67E-4	3.356E-4	4.196E-4	5.226E-4	6.446E-4	7.896E-4	9.652E-4	0.0011712
20	0.0014114	0.0016938	0.002022	0.0024	0.002838	0.003342	0.003918	0.004578	0.00533	0.00618
30	0.007144	0.008232	0.009452	0.010826	0.01236	0.014068	0.015972	0.018082	0.02042	0.02298
40	0.02582	0.02894	0.03234	0.03606	0.04014	0.04456	0.04936	0.05458	0.0602	0.06628
50	0.07284	0.07988	0.08742	0.09552	0.10416	0.11338	0.12318	0.13362	0.14468	0.1564
60	0.1688	0.18186	0.19564							
Precise α values for $n = 20$										
0	1.9074E-6	3.814E-6	5.722E-6	9.536E-6	1.3352E-5	1.9074E-5	2.67E-5	3.624E-5	4.768E-5	6.294E-5
10	8.202E-5	1.049E-4	1.3352E-4	1.6784E-4	2.098E-4	2.614E-4	3.224E-4	3.948E-4	4.826E-4	5.856E-4
20	7.076E-4	8.506E-4	0.0010166	0.0012092	0.0014324	0.00169	0.0019856	0.002326	0.002712	0.003152
30	0.003654	0.00422	0.00486	0.00558	0.00639	0.007296	0.008308	0.009436	0.010688	0.01208
40	0.013616	0.015312	0.017182	0.019234	0.02148	0.02396	0.02664	0.02958	0.03276	0.03624
50	0.03998	0.04406	0.04844	0.05316	0.05826	0.06372	0.06958	0.07586	0.08256	0.0897
60	0.0973	0.1054	0.11398	0.1231	0.13272	0.1429	0.15364	0.16496	0.17686	0.18934
Precise α values for $n = 21$										
0	9.536E-7	1.9074E-6	2.862E-6	4.768E-6	6.676E-6	9.536E-6	1.3352E-5	1.812E-5	2.384E-5	3.148E-5
10	4.1E-5	5.246E-5	6.676E-5	8.392E-5	1.049E-4	1.3066E-4	1.6118E-4	1.9742E-4	2.412E-4	2.928E-4
20	3.538E-4	4.262E-4	5.102E-4	6.074E-4	7.21E-4	8.516E-4	0.0010024	0.0011758	0.0013742	0.0016002
30	0.0018588	0.002152	0.002482	0.002858	0.003278	0.003752	0.004284	0.004878	0.005542	0.00628
40	0.007102	0.00801	0.009016	0.010126	0.011346	0.012692	0.014166	0.01578	0.017546	0.019474
50	0.02158	0.02386	0.02634	0.02902	0.03192	0.03506	0.03844	0.04208	0.046	0.0502
60	0.0547	0.0595	0.06464	0.07014	0.07598	0.0822	0.0888	0.0958	0.10322	0.11106
70	0.11934	0.12808	0.13728	0.14696	0.15714	0.1678	0.17898	0.19068		
Precise α values for $n = 22$										
0	4.768E-7	9.536E-7	1.4306E-6	2.384E-6	3.338E-6	4.768E-6	6.676E-6	9.06E-6	1.192E-5	1.5736E-5
10	2.05E-5	2.622E-5	3.338E-5	4.196E-5	5.246E-5	6.532E-5	8.058E-5	9.87E-5	1.2064E-4	1.4638E-4
20	1.769E-4	2.132E-4	2.556E-4	3.046E-4	3.62E-4	4.282E-4	5.044E-4	5.928E-4	6.938E-4	8.092E-4
30	9.412E-4	0.0010914	0.0012618	0.0014548	0.0016728	0.0019184	0.002194	0.002504	0.002852	0.00324
40	0.003672	0.004152	0.004684	0.005276	0.005928	0.00665	0.007444	0.008316	0.009274	0.010324
50	0.011472	0.012728	0.014094	0.015584	0.0172	0.018956	0.02086	0.02292	0.02514	0.02754
60	0.03012	0.0329	0.03588	0.03908	0.0425	0.04616	0.05008	0.05424	0.0587	0.06342
70	0.06844	0.07378	0.07942	0.0854	0.09174	0.09842	0.10546	0.11288	0.12068	0.12888
80	0.13748	0.14652	0.15598	0.16586	0.1762	0.18698	0.19822			
Precise α values for $n = 23$										

0	2.384E-7	4.768E-7	7.152E-7	1.192E-6	1.669E-6	2.384E-6	3.338E-6	4.53E-6	5.96E-6	7.868E-6
10	1.0252E-5	1.3114E-5	1.669E-5	2.098E-5	2.622E-5	3.266E-5	4.03E-5	4.936E-5	6.032E-5	7.32E-5
20	8.846E-5	1.0658E-4	1.278E-4	1.5258E-4	1.8144E-4	2.148E-4	2.534E-4	2.98E-4	3.492E-4	4.08E-4
30	4.752E-4	5.518E-4	6.388E-4	7.376E-4	8.494E-4	9.758E-4	0.0011184	0.0012786	0.0014584	0.0016598
40	0.001885	0.002136	0.002416	0.002726	0.00307	0.003452	0.003874	0.004338	0.004852	0.005414
50	0.006032	0.00671	0.007452	0.008262	0.009146	0.01011	0.011156	0.012294	0.013528	0.014866
60	0.016312	0.017872	0.019558	0.02138	0.02332	0.02542	0.02768	0.03008	0.03266	0.03544
70	0.03838	0.04152	0.04488	0.04844	0.05222	0.05626	0.06052	0.06504	0.06982	0.07486
80	0.0802	0.08582	0.09176	0.09798	0.10454	0.11142	0.11864	0.12622	0.13414	0.14242
90	0.15108	0.1601	0.16952	0.17934	0.18954					
Precise α values for $n = 24$										
0	1.192E-7	2.384E-7	3.576E-7	5.96E-7	8.344E-7	1.192E-6	1.669E-6	2.264E-6	2.98E-6	3.934E-6
10	5.126E-6	6.556E-6	8.344E-6	1.049E-5	1.3114E-5	1.6332E-5	2.014E-5	2.468E-5	3.016E-5	3.66E-5
20	4.422E-5	5.328E-5	6.39E-5	7.63E-5	9.084E-5	1.0764E-4	1.2708E-4	1.496E-4	1.7548E-4	2.052E-4
30	2.392E-4	2.782E-4	3.224E-4	3.728E-4	4.298E-4	4.944E-4	5.676E-4	6.498E-4	7.424E-4	8.462E-4
40	9.626E-4	0.0010926	0.001238	0.0013998	0.0015796	0.0017796	0.0020	0.002246	0.002516	0.002814
50	0.003144	0.003504	0.0039	0.004336	0.00481	0.00533	0.005898	0.006516	0.00719	0.00792
60	0.008714	0.009576	0.010508	0.011516	0.012604	0.01378	0.015044	0.016406	0.01787	0.019442
70	0.02112	0.02294	0.02486	0.02692	0.02914	0.03148	0.03398	0.03664	0.03948	0.04248
80	0.04568	0.04906	0.05264	0.05642	0.06042	0.06464	0.0691	0.0738	0.07872	0.08392
90	0.08938	0.0951	0.1011	0.10738	0.11396	0.12084	0.12802	0.13552	0.14336	0.1515
100	0.15998	0.1688	0.17798	0.1875	0.19738					
Precise α values for $n = 25$										
0	5.96E-8	1.192E-7	1.7882E-7	2.98E-7	4.172E-7	5.96E-7	8.344E-7	1.1324E-6	1.4902E-6	1.967E-6
10	2.562E-6	3.278E-6	4.172E-6	5.246E-6	6.556E-6	8.166E-6	1.0074E-5	1.2338E-5	1.508E-5	1.8298E-5
20	2.212E-5	2.664E-5	3.194E-5	3.814E-5	4.542E-5	5.388E-5	6.366E-5	7.498E-5	8.804E-5	1.03E-4
30	1.2022E-4	1.399E-4	1.623E-4	1.8788E-4	2.17E-4	2.498E-4	2.87E-4	3.29E-4	3.764E-4	4.296E-4
40	4.894E-4	5.564E-4	6.314E-4	7.15E-4	8.082E-4	9.118E-4	0.0010272	0.0011548	0.0012964	0.0014528
50	0.0016254	0.0018156	0.002026	0.002256	0.002508	0.002784	0.003088	0.00342	0.00378	0.004176
60	0.004604	0.005072	0.005578	0.00613	0.006726	0.00737	0.008068	0.008822	0.009636	0.01051
70	0.011454	0.012466	0.013554	0.014722	0.015972	0.017312	0.018744	0.02028	0.0219	0.02364
80	0.0255	0.02748	0.02958	0.0318	0.03418	0.03668	0.03934	0.04216	0.04512	0.04826
90	0.05158	0.05508	0.05876	0.06262	0.0667	0.07098	0.07548	0.0802	0.08514	0.09032
100	0.09574	0.1014	0.10732	0.1135	0.11994	0.12664	0.13364	0.14092	0.14848	0.15634
110	0.1645	0.17296	0.18172	0.19082						
Precise α values for $n = 26$										
0	2.98E-8	5.96E-8	8.94E-8	1.4902E-7	2.086E-7	2.98E-7	4.172E-7	5.662E-7	7.45E-7	9.834E-7
10	1.2814E-6	1.6392E-6	2.086E-6	2.622E-6	3.278E-6	4.082E-6	5.036E-6	6.17E-6	7.54E-6	9.15E-6
20	1.1056E-5	1.3322E-5	1.5974E-5	1.9074E-5	2.27E-5	2.694E-5	3.186E-5	3.756E-5	4.41E-5	5.164E-5
30	6.032E-5	7.024E-5	8.156E-5	9.45E-5	1.092E-4	1.2588E-4	1.448E-4	1.6618E-4	1.9028E-4	2.174E-4
40	2.48E-4	2.822E-4	3.208E-4	3.636E-4	4.116E-4	4.65E-4	5.246E-4	5.908E-4	6.642E-4	7.454E-4
50	8.354E-4	9.348E-4	0.0010444	0.0011652	0.001298	0.001444	0.001604	0.0017796	0.0019716	0.002182
60	0.00241	0.00266	0.002932	0.00323	0.00355	0.0039	0.00428	0.00469	0.005134	0.005612
70	0.00613	0.00669	0.00729	0.007938	0.008634	0.009382	0.010186	0.011046	0.011966	0.012952
80	0.014006	0.015132	0.016334	0.017614	0.018978	0.02042	0.02198	0.02362	0.02536	0.0272
90	0.02916	0.03122	0.03342	0.03572	0.03816	0.04074	0.04346	0.04634	0.04934	0.05252
100	0.05586	0.05936	0.06302	0.06688	0.07092	0.07514	0.07958	0.0842	0.08902	0.09408

110	0.09934	0.10482	0.11054	0.11648	0.12266	0.1291	0.13578	0.14272	0.1499	0.15736
120	0.16508	0.17308	0.18136	0.1899	0.19874					
Precise α values for $n = 27$										
0	1.4902E-8	2.98E-8	4.47E-8	7.45E-8	1.043E-7	1.4902E-7	2.086E-7	2.832E-7	3.726E-7	4.918E-7
10	6.408E-7	8.196E-7	1.043E-6	1.3114E-6	1.6392E-6	2.042E-6	2.518E-6	3.084E-6	3.77E-6	4.574E-6
20	5.528E-6	6.66E-6	7.988E-6	9.536E-6	1.1354E-5	1.347E-5	1.593E-5	1.879E-5	2.208E-5	2.586E-5
30	3.024E-5	3.522E-5	4.094E-5	4.746E-5	5.488E-5	6.332E-5	7.29E-5	8.372E-5	9.596E-5	1.0978E-4
40	1.2532E-4	1.4278E-4	1.624E-4	1.8434E-4	2.088E-4	2.364E-4	2.668E-4	3.008E-4	3.388E-4	3.808E-4
50	4.272E-4	4.788E-4	5.356E-4	5.984E-4	6.678E-4	7.44E-4	8.278E-4	9.2E-4	0.001021	0.0011316
60	0.0012526	0.001385	0.0015294	0.001687	0.0018586	0.002046	0.002248	0.002468	0.002708	0.002966
70	0.003248	0.00355	0.003878	0.004232	0.004612	0.005024	0.005466	0.00594	0.00645	0.006998
80	0.007586	0.008216	0.008888	0.009608	0.010378	0.0112	0.012076	0.01301	0.014006	0.015064
90	0.01619	0.017386	0.018656	0.02	0.02142	0.02294	0.02454	0.02624	0.02802	0.0299
100	0.0319	0.034	0.0362	0.03854	0.04098	0.04356	0.04626	0.0491	0.05208	0.0552
110	0.05848	0.0619	0.06548	0.06922	0.07314	0.07722	0.08148	0.08594	0.09056	0.09538
120	0.1004	0.10562	0.11106	0.11668	0.12254	0.1286	0.13488	0.1414	0.14816	0.15514
130	0.16236	0.16982	0.17752	0.18548	0.19368					
Precise α values for $n = 28$										
0	7.45E-9	1.4902E-8	2.236E-8	3.726E-8	5.216E-8	7.45E-8	1.043E-7	1.4156E-7	1.8626E-7	2.458E-7
10	3.204E-7	4.098E-7	5.216E-7	6.556E-7	8.196E-7	1.0208E-6	1.2592E-6	1.5422E-6	1.885E-6	2.288E-6
20	2.764E-6	3.33E-6	3.994E-6	4.768E-6	5.678E-6	6.736E-6	7.964E-6	9.396E-6	1.105E-5	1.295E-5
30	1.514E-5	1.765E-5	2.052E-5	2.38E-5	2.754E-5	3.18E-5	3.664E-5	4.212E-5	4.83E-5	5.53E-5
40	6.318E-5	7.204E-5	8.202E-5	9.32E-5	1.057E-4	1.197E-4	1.3532E-4	1.5274E-4	1.7214E-4	1.9368E-4
50	2.176E-4	2.442E-4	2.736E-4	3.06E-4	3.418E-4	3.814E-4	4.25E-4	4.73E-4	5.256E-4	5.834E-4
60	6.468E-4	7.162E-4	7.922E-4	8.752E-4	9.658E-4	0.0010646	0.0011722	0.0012892	0.0014166	0.0015548
70	0.0017048	0.0018674	0.002044	0.002234	0.00244	0.002662	0.002902	0.00316	0.003438	0.003738
80	0.00406	0.004406	0.004778	0.005176	0.005604	0.00606	0.006548	0.007072	0.00763	0.008224
90	0.00886	0.009536	0.010256	0.011024	0.011838	0.012704	0.013624	0.014598	0.015632	0.016728
100	0.017886	0.019114	0.0204	0.02178	0.02322	0.02474	0.02636	0.02806	0.02984	0.0317
110	0.03368	0.03576	0.03792	0.04022	0.0426	0.04512	0.04774	0.0505	0.05338	0.05638
120	0.05954	0.06282	0.06624	0.06982	0.07354	0.07742	0.08146	0.08566	0.09002	0.09456
130	0.09928	0.10418	0.10926	0.11452	0.11998	0.12562	0.13146	0.13752	0.14378	0.15024
140	0.1569	0.1638	0.1709	0.17824	0.18578	0.19356				
Precise α values for $n = 29$										
0	3.726E-9	7.45E-9	1.1176E-8	1.8626E-8	2.608E-8	3.726E-8	5.216E-8	7.078E-8	9.314E-8	1.2294E-7
10	1.6018E-7	2.048E-7	2.608E-7	3.278E-7	4.098E-7	5.104E-7	6.296E-7	7.712E-7	9.424E-7	1.1436E-6
20	1.382E-6	1.6652E-6	1.9968E-6	2.384E-6	2.838E-6	3.368E-6	3.982E-6	4.698E-6	5.524E-6	6.478E-6
30	7.578E-6	8.836E-6	1.0278E-5	1.1928E-5	1.381E-5	1.5952E-5	1.8388E-5	2.114E-5	2.428E-5	2.78E-5
40	3.18E-5	3.628E-5	4.134E-5	4.7E-5	5.336E-5	6.048E-5	6.844E-5	7.732E-5	8.72E-5	9.822E-5
50	1.1046E-4	1.2406E-4	1.3914E-4	1.5582E-4	1.743E-4	1.9468E-4	2.172E-4	2.42E-4	2.692E-4	2.992E-4
60	3.322E-4	3.684E-4	4.08E-4	4.514E-4	4.988E-4	5.506E-4	6.072E-4	6.688E-4	7.36E-4	8.09E-4
70	8.884E-4	9.746E-4	0.0010684	0.0011698	0.0012798	0.0013988	0.0015274	0.0016664	0.0018164	0.0019782
80	0.002152	0.00234	0.002542	0.00276	0.002992	0.003242	0.00351	0.003798	0.004106	0.004436
90	0.004788	0.005164	0.005566	0.005994	0.006452	0.006938	0.007456	0.008008	0.008594	0.009216
100	0.009878	0.010578	0.011322	0.01211	0.012944	0.013826	0.014758	0.015744	0.016786	0.017884
110	0.019044	0.02026	0.02156	0.0229	0.02434	0.02584	0.0274	0.02906	0.0308	0.03262
120	0.03454	0.03654	0.03864	0.04082	0.04312	0.04552	0.04802	0.05064	0.05338	0.05622

130	0.0592	0.0623	0.06552	0.06888	0.07236	0.07598	0.07976	0.08368	0.08774	0.09196
140	0.09634	0.10086	0.10556	0.11042	0.11546	0.12066	0.12604	0.1316	0.13732	0.14326
150	0.14936	0.15566	0.16216	0.16886	0.17574	0.18284	0.19012	0.19762		
Precise α values for $n = 30$										
0	1.8626E-9	3.726E-9	5.588E-9	9.314E-9	1.3038E-8	1.8626E-8	2.608E-8	3.54E-8	4.656E-8	6.146E-8
10	8.01E-8	1.0244E-7	1.3038E-7	1.6392E-7	2.048E-7	2.552E-7	3.148E-7	3.856E-7	4.712E-7	5.718E-7
20	6.91E-7	8.326E-7	9.984E-7	1.192E-6	1.4194E-6	1.6838E-6	1.9912E-6	2.348E-6	2.762E-6	3.24E-6
30	3.79E-6	4.422E-6	5.144E-6	5.974E-6	6.918E-6	7.994E-6	9.22E-6	1.061E-5	1.2184E-5	1.3966E-5
40	1.5978E-5	1.8244E-5	2.08E-5	2.366E-5	2.688E-5	3.05E-5	3.454E-5	3.904E-5	4.408E-5	4.968E-5
50	5.592E-5	6.286E-5	7.056E-5	7.91E-5	8.856E-5	9.902E-5	1.1058E-4	1.2334E-4	1.374E-4	1.5288E-4
60	1.699E-4	1.886E-4	2.092E-4	2.316E-4	2.562E-4	2.832E-4	3.128E-4	3.45E-4	3.8E-4	4.184E-4
70	4.602E-4	5.054E-4	5.548E-4	6.084E-4	6.666E-4	7.296E-4	7.978E-4	8.718E-4	9.518E-4	0.0010382
80	0.0011314	0.0012322	0.0013406	0.0014574	0.0015832	0.0017186	0.001864	0.00202	0.002188	0.002368
90	0.00256	0.002766	0.002988	0.003222	0.003476	0.003744	0.004032	0.004338	0.004664	0.005012
100	0.005382	0.005776	0.006194	0.00664	0.007112	0.007612	0.008142	0.008706	0.009302	0.009932
110	0.010598	0.011304	0.012048	0.012834	0.013664	0.014538	0.01546	0.016432	0.017454	0.01853
120	0.01966	0.02084	0.0221	0.02342	0.02478	0.02622	0.02774	0.02932	0.03098	0.03272
130	0.03454	0.03644	0.03842	0.04048	0.04266	0.0449	0.04726	0.04972	0.05226	0.05492
140	0.05768	0.06056	0.06356	0.06666	0.0699	0.07324	0.07672	0.08032	0.08406	0.08794
150	0.09194	0.0961	0.1004	0.10484	0.10944	0.11418	0.11908	0.12414	0.12936	0.13474
160	0.14028	0.146	0.15188	0.15794	0.16418	0.1706	0.1772	0.18396	0.19092	0.19808

Table 28.18: Table with precise α-values for $n \in 4..30$ (from Darlington [484]).

Further information and tables for Wilcoxon rank distributions can be found in [1290, 1436, 2221] and [2222]. [1076]

Mann-Whitney U Test

Wilcoxon's signed rank test [2220] has the drawback that the data samples must be arranged in pairs (a_i, b_i) and thus, the number of the a_i has to be the same as the b_i. In many practical experiments, this is not the case and Wilcoxon's test is not applicable. Assume we have run two experimental series with different EAs applied to the same problem for some time (where each run in a series has the same configuration). Then, there is no group relation between the measurements a_i and b_i and creating tuples as needed for the signed rank test is, basically, nonsense. Furthermore, it could be possible that the runs in the first series of tests usually finished faster than those in the second series. Then, we would have more samples a_i than b_i. In order to apply the signed rank test properly, we need as same as many samples from both configuration and therefore would have to discard some samples which may lead to a loss of important information.

The *U test*[78] developed by Mann and Whitney [1356] circumvents these problems [1878, 573, 1520]. It assesses whether two samples are from the same distribution (H_0) or not (H_1). Different from the sign rank test, it does not require the samples to be paired nor to contain the same number of elements.

Basically, the U test is carried out almost exactly like the signed rank test. We will illustrate this using the example for unpaired samples given in Table 28.15 where the set a with median 3 has $n_a = 6$ samples and b consists of $n_b = 8$ elements with a median of 5.5.

1. The elements a_i and b_i are mixed together and sorted.
2. Each element now receives a rank corresponding to its position in the list. Like in the sign test, elements which have the same value receive the same rank (see point 4 in Section 28.8.1). The elements of the first two rows in Table 28.15, for instance, both receive rank $0.5(1+2) = 1.5$ whereas those in row 7 to 10 have the rank $0.5(7+10) = 8.5$.

[78] http://en.wikipedia.org/wiki/Mann-Whitney_U [accessed 2008-10-24]

3. The rank sums $R_a = \sum_{r_a} = 28$ and $R_b = \sum_{r_b} = 77$ are determined, where $R_a + R_b = 105 = \frac{n(n+1)}{2} = \frac{14*15)}{2}$ (with $n = n_a + n_b$) always holds.
4. The sample statistics are then given as $U_a = R_a - \frac{n_a(n_a+1)}{2} = 28 - 21 = 7$ and $U_b = R_b - \frac{n_b(n_b+1)}{2} = 77 - 36 = 41$ (where $U_a + U_b = n_a n_b$ always holds).
5. The smaller of the two values $U = \min\{U_a, U_b\} = 7$ is used.
6. For the significance level α the critical U values can be computed for the two-sided test as
$$U_\alpha = \frac{n_a n_b}{2} - z\Big(1 - \frac{\alpha}{2}\Big)\sqrt{\frac{n_a n_b\,(n_a + n_b + 1)}{12}} \tag{28.293}$$
where z is probit function, the inverse cumulative distribution function of the standard normal distribution (see Definition 28.49). The values of z can be looked up in Table 28.7. For $\alpha = 0.05$ we get $z\big(1 - \frac{\alpha}{2}\big) = z(0.975) \approx 1.96$ and for $\alpha = 0.01$, we find $z\big(1 - \frac{\alpha}{2}\big) = z(0.995) \approx 2.575$. Hence, $U_{0.05} \approx 24 - 1.96\sqrt{60} \approx 8.82$ and $U_{0.01} \approx 24 - 2.575\sqrt{60} \approx 4.05$.
7. We compare U with U_α and can discard the null hypothesis H_0 if and only if U is smaller.

In the example, $U < U_{0.05}$ holds whereas $U < U_{0.01}$ does not. In other words, with 5% chance of error, we can declare the two samples a and b to be different and can assume that the median $\text{med}(a) = 3$ is significantly smaller than $\text{med}(b) = 5.5$. If we wish for no more than 1% chance of error, however, the differences between a and b are not significant (enough). For $\alpha = 0.05$, the critical values of $U_{0.05}$, i. e., the highest allowed U values for which the null hypothesis H_0 can be rejected, are listed in Table 28.19. For $n_a = 6$ and $n_b = 8$, we find $U_{0.05} \approx 8$ which fits to the value used in point 6 of the example application of the test. The U test can be computed with the nice online utility provided by Lowry [1312].

$n_b \backslash n_a$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	∅																			
2	∅	∅																		
3	∅	∅	∅																	
4	∅	∅	∅	0																
5	∅	∅	0	1	2															
6	∅	∅	1	2	3	5														
7	∅	∅	1	3	5	6	8													
8	∅	0	2	4	6	8	10	13												
9	∅	0	2	4	7	10	12	15	17											
10	∅	0	3	5	8	11	14	17	20	23										
11	∅	0	3	6	9	13	16	19	23	26	30									
12	∅	1	4	7	11	14	18	22	26	29	33	37								
13	∅	1	4	8	12	16	20	24	28	33	37	41	45							
14	∅	1	5	9	13	17	22	26	31	36	40	45	50	55						
15	∅	1	5	10	14	19	24	29	34	39	44	49	54	59	64					
16	∅	1	6	11	15	21	26	31	37	42	47	53	59	64	70	75				
17	∅	2	6	11	17	22	28	34	39	45	51	57	63	69	75	81	87			
18	∅	2	7	12	18	24	30	36	42	48	55	61	67	74	80	86	93	99		
19	∅	2	7	13	19	25	32	38	45	52	58	65	72	78	85	92	99	106	113	
20	∅	2	8	14	20	27	34	41	48	55	62	69	76	83	90	98	105	112	119	127
21	∅	3	8	15	22	29	36	43	50	58	65	73	80	88	96	103	111	119	126	134
22	∅	3	9	16	23	30	38	45	53	61	69	77	85	93	101	109	117	125	133	141
23	∅	3	9	17	24	32	40	48	56	64	73	81	89	98	106	115	123	132	140	149
24	∅	3	10	17	25	33	42	50	59	67	76	85	94	102	111	120	129	138	147	156
25	∅	3	10	18	27	35	44	53	62	71	80	89	98	107	117	126	135	145	154	163
26	∅	4	11	19	28	37	46	55	64	74	83	93	102	112	122	132	141	151	161	171
27	∅	4	11	20	29	38	48	57	67	77	87	97	107	117	127	137	147	158	168	178
28	∅	4	12	21	30	40	50	60	70	80	90	101	111	122	132	143	154	164	175	186
29	∅	4	13	22	32	42	52	62	73	83	94	105	116	127	138	149	160	171	182	193
30	∅	5	13	23	33	43	54	65	76	87	98	109	120	131	143	154	166	177	189	200
31	∅	5	14	24	34	45	56	67	78	90	101	113	125	136	148	160	172	184	196	208
32	∅	5	14	24	35	46	58	69	81	93	105	117	129	141	153	166	178	190	203	215
33	∅	5	15	25	37	48	60	72	84	96	108	121	133	146	159	171	184	197	210	222
34	∅	5	15	26	38	50	62	74	87	99	112	125	138	151	164	177	190	203	217	230
35	∅	6	16	27	39	51	64	77	89	103	116	129	142	156	169	183	196	210	224	237
36	∅	6	16	28	40	53	66	79	92	106	119	133	147	161	174	188	202	216	231	245
37	∅	6	17	29	41	55	68	81	95	109	123	137	151	165	180	194	209	223	238	252
38	∅	6	17	30	43	56	70	84	98	112	127	141	156	170	185	200	215	230	245	259
39	0	7	18	31	44	58	72	86	101	115	130	145	160	175	190	206	221	236	252	267
40	0	7	18	31	45	59	74	89	103	119	134	149	165	180	196	211	227	243	258	274

Table 28.19: The critical values $U_{0.05}$ for the two-sided Mann-Whitney U test [2219].

Fisher's Exact Test

Fisher's exact test[79] [678, 680, 15, 252, 1310] tests whether two samples with binary data are independent or not. The null hypothesis H_0 is that the values in both samples follow the same distribution. If H_0 does not hold, the two samples differ in the probabilities with which the two possible binary values occur in them and hence, then distribution of these values depends on the samples.

For illustration purposes let us go back to the (unpaired) example data sets given in Table 28.15 on page 510. Assume that the columns a and b would represent different configurations of an optimizer applied to a single-objective optimization problem. Then, in

[79] http://en.wikipedia.org/wiki/Fisher%27s_exact_test [accessed 2008-12-08]

each cell, the objective value (subject to minimization) of the best solution candidate found in the corresponding run is noted. Assume we can consider an experiment as successful if this value is below 4 and that success (s) or failure ($\overline{s}$) was the binary criterion which we want to investigate.

Series a has four successful runs ($a_s = 4$) and two failed ones ($a_{\overline{s}} = 2$) whereas series b succeeded only once ($b_s = 1$) while seven runs could not create a solution candidate with an objective value below 4 ($b_{\overline{s}} = 7$). We have sketched this scenario (let us call it C_5) in Table 28.20. Series a seems to be more successful than series b in this example. The question is if this is indeed statistical significant or whether it might have been a fluke as well (and the null hypothesis H_0 is more likely to hold).

C_5	a	b	$\sum$
s	$a_s = 4$	$b_s = 1$	5
$\overline{s}$	$a_{\overline{s}} = 2$	$b_{\overline{s}} = 7$	9
$\sum$	6	8	$N = 14$

Table 28.20: An 2×2 contingency table based on Table 28.15.

Assume that the distribution of s and $\overline{s}$ was the same in the stochastic processes which have been sampled (as a and b), i.e., that the null hypothesis H_0 holds. Then, the probability that any configuration $C = (a_s, a_{\overline{s}}, b_s, b_{\overline{s}})$ would have resulted is given in Equation 28.294. In Equation 28.295, we apply this equation to the scenario C_5 shown in Table 28.20 and obtain a probability of roughly 6% for it.

$$P(C) = \frac{\binom{a_s+b_s}{a_s}\binom{a_{\overline{s}}+b_{\overline{s}}}{a_{\overline{s}}}}{\binom{N}{a_s+a_{\overline{s}}}} = \frac{(a_s+b_s)!\ (a_{\overline{s}}+b_{\overline{s}})!\ (a_s+a_{\overline{s}})!\ (b_s+b_{\overline{s}})\ !}{N!\ a_s!\ a_{\overline{s}}!\ b_s!\ b_{\overline{s}}!} \tag{28.294}$$

$$P(C_5) = \frac{5!\ 9!\ 6!\ 8!}{14!\ 4!\ 2!\ 1!\ 7!} = \frac{1\,264\,146\,186\,240\,000}{21\,090\,172\,207\,104\,000} = \frac{60}{1001} \approx 0.059\,940\,1 \tag{28.295}$$

In order to test whether we can reject the null hypothesis, we need to compute the total

C_1	a	b	$\sum$	C_2	a	b	$\sum$	C_3	a	b	$\sum$	C_4	a	b	$\sum$	C_5	a	b	$\sum$	C_6	a	b	$\sum$
s	0	5	5	s	1	4	5	s	2	3	5	s	3	2	5	s	4	1	5	s	5	0	5
$\overline{s}$	6	3	9	$\overline{s}$	5	4	9	$\overline{s}$	4	5	9	$\overline{s}$	3	6	9	$\overline{s}$	2	7	9	$\overline{s}$	1	8	9
$\sum$	6	8	14	$\sum$	6	8	14	$\sum$	6	8	14	$\sum$	6	8	14	$\sum$	6	8	14	$\sum$	6	8	14
$P(C_1) \approx 0.0280$				$P(C_2) \approx 0.2098$				$P(C_3) \approx 0.4196$				$P(C_4) \approx 0.2797$				$P(C_5) \approx 0.0599$				$P(C_6) \approx 0.0030$			
$D(C_1) = 0.\overline{5}$				$D(C_2) = 0.3\overline{5}$				$D(C_3) = 0.0\overline{4}$				$D(C_4) = 0.2\overline{6}$				$D(C_5) = 0.5\overline{7}$				$D(C_6) = 0.\overline{8}$			

Table 28.21: All configurations with the same total sums in a/b and $s/\overline{s}$ than in Table 28.20.

probability that a configuration *at least as extreme* as C_5 with the same a/b and $s/\overline{s}$ division (the $\sum$ row and column) can occur. In Table 28.21 we list the scenarios with the same marginal distributions and their corresponding probabilities (which sum up to one) in the second-to-last row. What remains to do is to find out which of these scenarios are *at least as extreme* as C_5. For each scenario C, we compute the disproportion $D(C)$ – the degree of ostensible dependency between the two samples [1310] – according to Equation 28.296. For C_5, we have computed this value in Equation 28.297.

$$D(C) = \left| \frac{a_s}{a_s + b_s} - \frac{a_{\overline{s}}}{a_{\overline{s}} + b_{\overline{s}}} \right| \tag{28.296}$$

$$D(C_5) = \left| \frac{4}{4+1} - \frac{2}{2+7} \right| = \frac{26}{45} = 0.5\overline{7} \approx 0.578 \tag{28.297}$$

In the bottom row of Table 28.21, we have listed the disproportion values D for all six possible scenarios. Amongst these, only scenario C_6 and C_1 (and C_5) have a D-value at least as big as C_5, so we can compute the probability p with which an experimental outcome at least as extreme as the observed C_5 would occur under H_0 as $p = P(C_1) + P(C_5) + P(C_6) = 0.\overline{09} \approx 0.09$. Hence, under a significance level of $\alpha = 10\%$, we could reject the null hypothesis H_0 and assume that there is a difference between the samples a and b. If we want a significance level of $\alpha = 5\%$, we cannot reject H_0 and must consider the experimental outcome as fluke.

Computing Fisher's exact test by hand may become time consuming. The online utilities provided by Lowry [1311] and Langsrud [1246] (which sometimes produce slightly different results) provide a handy alternative.

28.9 Generating Random Numbers

Definition 28.67 (Random Number). Random numbers are the values taken on by a random variable. A random number generator[80] produces a sequence $r = (r_1, r_2, \dots)$ of random numbers r_i as result of independent repetitions of the same random experiment.

Since the numbers r_i are all produced by the same random experiment, they approximate a certain random distribution according to the law of large numbers (see Section 28.3.8 on page 478).

For true random number generators, there exists no function or algorithm $f(i) = r_i$ or $f(r_{i-n+1}, r_{i-n+2}, .., r_i) = r_{i+1}$ that can produce this sequence in a deterministic manner with or without knowledge of the random numbers previously returned from the generator. Such behavior can be achieved by obtaining the numbers r_i from measurements of a physical process, for instance. Today, there exist many such so-called hardware random number generators [81] [603, 2126, 676, 1981].

Of course, most computers are not equipped with special hardware for random number production, although some standard devices can be utilized for that purpose. One could, for example, measure the white noise of soundcards or the delays between the user's keystrokes. Such methods have the drawback that they require the presence of and access to such components. Furthermore, the speed of them is limited since you cannot produce random numbers faster than the recording speed of the soundcard or faster than the user is typing.

28.9.1 Generating Pseudorandom Numbers

In security-sensitive areas like cryptography, we need true random numbers [2125, 1981, 1373]. For normal PC applications and most scientific purposes, pseudorandom number generators[82] are sufficient.

The principle of pseudorandom number generators is to produce a sequence of numbers $r = (r_1, r_2, .., r_i), r_j \in R \; \forall j \in \mathbb{N}, R \subseteq \mathbb{R}$ which are not *obviously* interdependent, i. e., if knowing a number r_i there is not simple way to find out the value of r_{i+1}.

[80] http://en.wikipedia.org/wiki/Random_number_generator [accessed 2007-07-03], http://en.wikipedia.org/wiki/Random_number_generation [accessed 2007-07-03]

[81] http://en.wikipedia.org/wiki/Hardware_random_number_generator [accessed 2007-07-03]

[82] http://en.wikipedia.org/wiki/Pseudorandom_number_generator [accessed 2007-07-03], http://en.wikipedia.org/wiki/Pseudorandomness [accessed 2007-07-03]

Of course, since the values r_i are no real random numbers, there is an algorithm or function $f : V \to R \times V$ where R is the set of possible numbers and V is the space of some internal variables. These internal variables are referred to as seed and normally change whenever a new number is produced. Often, the seed is initialized with either a true random number or the current system time. In the first case, it is also practicable to re-initialize the seed from time to time with new true random values.

Pseudorandom numbers are attractive to all not security-critical applications where we need some sort of unpredictable behavior. They are often used in games or simulations, since they usually can be generated much quicker than true random numbers. On the other hand, especially in scientific applications the "degree" of randomness is very important. There are many incidents, for example in physical simulation, where the inappropriate use of pseudorandom number generators of poor quality lead to wrong conclusions [2112, 1852, 662]. It should be noted that there also exist cryptographically secure pseudorandom number generators[83] which create pseudo-random number that are of especially high quality.

There exists a variety of algorithms that generate pseudorandom numbers [2008, 1611, 234, 348] and many implementations for different programming languages and architectures [918, 1594, 421]. It is even possible to evolve pseudorandom number generators using Genetic Programming, as shown, for instance, by Koza [1192].

Linear Congruential Generator (LCG)

The linear congruential generator[84] (LCG) was first proposed by Lehmer [1272] and is one of the most frequently used and simplest pseudo random number generators [1271]. It updates an internal integer number $v \in V = (0 \ldots (m-1)), m \in \mathbb{N}$ in each step according to Equation 28.298. The modulus m is a natural number which defines the maximum number of values v can take on. a and b are both constants. Therefore, v will periodically take on the same values – at most after m steps. The pseudorandom numbers r_i are approximately uniformly distributed in the interval $[0, m)$ (see Section 28.4.1) and can be computed as proposed in Equation 28.299.

$$v_i = (av_{i-1} + b) \bmod m \tag{28.298}$$

$$r_i = \frac{v_i}{m} \tag{28.299}$$

If the full period can really be reached depends a lot on the values of the parameters a, b, and m. There are many constellations known where only a small fraction of the period m is utilized [634]. In order to produce the full period, the following requirements should be met according to Wikipedia [2219].

1. b and m are relatively prime
2. $a - 1$ is divisible by all prime factors of m
3. $a - 1$ is a multiple of 4 if m is a multiple of 4
4. $m > \max\{a, b, v_0\}$
5. $a > 0, b > 0$

Good standard values for the constants are $a = 1\,664\,525$, $b = 1\,013\,904\,223$, and $m = 2^{32}$. One of the widest spread realizations of LCGs has been outlined by Knuth [1161]. In Java, the class `java.util.Random` uses this approach with the settings $a = 25\,214\,903\,917$, $b = 11$, and $m = 2^{48}$.

[83] http://en.wikipedia.org/wiki/Cryptographically_secure_pseudorandom_number_generator [accessed 2007-07-03]

[84] http://en.wikipedia.org/wiki/Linear_congruential_generator [accessed 2007-07-03]

28.9.2 Random Functions

Definition 28.68 (Random Function). In the context of this book, we define a random function random as a construct that eases the utilization of random numbers and random variables. It represents access to a random process, an infinite sequence of random variables X_i all distributed according to the same distribution function. Starting with X_1, each time a random function is evaluated, it returns the value of the next random variable in the sequence $i = 1, 2, 3, \ldots$.

Definition 28.69 (Uniformly Distributed Random Number Generator). We define the function $\text{random}_u(\check{r}, \hat{r})$ to draw uniformly distributed (see Section 28.5.1 on page 485) random numbers from the interval with the boundaries $\check{r}$ (inclusively) and $\hat{r}$ (exclusively). The parameter-less function $\text{random}_u()$ will return a uniformly distributed number from the interval spanning from 0 inclusively to 1 exclusively.

$$\text{random}_u(\check{r}, \hat{r}) \in [\check{r}, \hat{r}) \subseteq \mathbb{R}, \check{r}, \hat{r} \in \mathbb{R}, \check{r} < \hat{r} \tag{28.300}$$

$$\text{random}_u() \equiv \text{random}_u(0, 1) \tag{28.301}$$

The $\text{random}_u()$-function can be realized with the linear congruential pseudorandom number generators that we have just discussed in Section 28.9.1, for example.

Definition 28.70 (Normally Distributed Random Number Generator). We define the function $\text{random}_n\left(\mu, \sigma^2\right)$ to generate normally distributed (see Section 28.5.2 on page 486) random numbers with the expected value μ and the variance σ^2. The parameter-less function $\text{random}_n()$ will return a standard normally distributed number (with $\mu = 0$ and $\sigma^2 = 1$).

$$\text{random}_n\left(\mu, \sigma^2\right) \sim N\left(\mu, \sigma^2\right) \tag{28.302}$$

$$\text{random}_n() \equiv \text{random}_n(0, 1) \tag{28.303}$$

Cut-off Random Functions

We often use random processes and random functions to model or simulate a certain features of a real system. If we, for example, simulate a chicken farm, we might be interested in the size of the eggs laid by the hens. We can assume this weight to be normally distributed[85] around some mean μ with a variance $\sigma^2 \neq 0$. In the simulation, a series of egg weights is created simply be drawing subsequent such random numbers by calling $\text{random}_n\left(\mu, \sigma^2\right)$ repeatedly. Although the normal distribution is a good model for egg weights, it has a serious drawback: no matter how we chose μ or σ, there is still a positive probability of drawing zero, negative, or extremely large (> 10t) weights. In reality however, such things could have not yet been documented.

What we need here is a cut-off mechanism for our random function $\text{random}_n\left(\mu, \sigma^2\right)$ that still preserves as many of its properties as possible. Given any random function random the function $\text{random}_l(\text{random}, low, high)$, defined as Algorithm 28.2, ensures that $low \leq \text{random}_l(\text{random}, low, high) < high$.

28.9.3 Converting Random Numbers to other Distributions

There are occasions where random numbers of a different distribution than available are needed. We could, for example, have a linear congruential generator for uniformly distributed random numbers like elaborated in Section 28.9.1 but may need normally distributed values.

[85] see Section 28.5.2 on page 486

Algorithm 28.2: $r \longleftarrow \text{random}_l(\text{random}, low, high)$

Input: random: a random function (maybe with further implicit parameters)
Input: $low \in \mathbb{R}$: the inclusive, lower bound of the random result
Input: $high \in \mathbb{R}, low < high$: the exclusive, upper bound of the random result
Data: r: the intermediate random value
Output: r: a value returned by *random* with $low \leq r < high$

```
1 begin
2 |  repeat
3 |  |  r ⟵ random()
4 |  until (r≥low) ∧ (r < high)
5 |  return r
6 end
```

Uniform Distribution → Uniform Distribution

If we have random numbers r_i distributed uniformly in the interval $[a_1, b_1)$ and need random numbers s_i uniformly distributed in the interval $[a_2, b_2)$, they can be converted really simple according to

$$s_i = a_2 + (b_2 - a_2)\frac{r_i - a_1}{b_1 - a_1} \tag{28.304}$$

Uniform Distribution → Normal Distribution

In order to transform random numbers which are uniformly distributed in the interval $[0, 1)$ to standard-normally distributed random numbers ($\mu = 0$, $\sigma^2 = 1$), we can apply the Box-Muller[86] transformation [262]. This approach creates two standard-normally distributed random numbers n_1, n_2 from two random numbers r_1, r_2 which are uniformly distributed in $[0, 1)$ at once according to Equation 28.305. In both formulas, the terms $\sqrt{-2\ln r_1}$ and $2\pi r_2$ are used. The performance can be increased if both terms are computed only once and reused.

$$\begin{aligned} n_1 &= \sqrt{-2\ln r_1}\cos(2\pi r_2) \\ n_2 &= \sqrt{-2\ln r_1}\sin(2\pi r_2) \end{aligned} \tag{28.305}$$

The polar form of this method, illustrated as Algorithm 28.3, is not only faster, but also numerically more robust [556]. It creates two independent random numbers uniformly distributed in $[-1, 1)$ and computes their product w. This is repeated until $w \in (0, 1)$. With this value, we now can compute two independent, standard-normally distributed random numbers. Effectively, we have traded a trigonometric operation and a multiplication against a division compared to the original method in Equation 28.305. The implementation of this algorithm is discussed in [1161] which is the foundation of the method `nextGaussian` of the Java-class `java.util.Random`.

Normal Distribution → Normal Distribution

With Equation 28.306, a normally distributed random number $n_1 \sim N(\mu_1, \sigma_1^2)$ can be transformed to another normally distributed random number $n_2 \sim N(\mu_2, \sigma_2^2)$.

$$n_2 = \mu_2 + \sigma_2 * \frac{n_1 - \mu_1}{\sigma_1} \tag{28.306}$$

[86] http://en.wikipedia.org/wiki/Box_muller [accessed 2007-07-03]

Algorithm 28.3: $(n_1, n_2) \longleftarrow \text{random}_{n,p^2}()$

Data: n_1, n_2: the intermediate and result variables
Data: w: the polar radius
Output: (n_1, n_2): a tuple of two independent values $n_1 \sim N(0,1), n_2 \sim N(0,1)$

```
1 begin
2     repeat
3         n1 ⟵ random_u(−1, 1)
4         n2 ⟵ random_u(−1, 1)
5         w ⟵ (n1 ∗ n1) + (n2 ∗ n2)
6     until (w > 0) ∧ (w < 1)
7     w ⟵ sqrt((−2 ln w)/w)
8     return (n1 ∗ w, n2 ∗ w)
9 end
```

Uniform Distribution → Exponential Distribution

With Equation 28.307, a random number r uniformly distributed in the interval $(0, 1)$ (0 is excluded) can be transformed into a exponentially distributed random number $s \sim exp(\lambda)$.

$$s = \frac{-\ln r}{\lambda} \tag{28.307}$$

Exponential Distribution → Exponential Distribution

With Equation 28.308, an exponentially distributed random number $r_1 \sim exp(\lambda_1)$ can be transformed to an exponentially distributed number $r_2 \sim exp(\lambda_2)$.

$$r_2 = \frac{\lambda_1}{\lambda_2} r_1 \tag{28.308}$$

Uniform Distribution → Bell-shaped Distribution

The bases of many numerical optimization algorithms is the modification of a value x by adding some random number to it. If the probability density function of the underlying distribution producing number is symmetrically bell-shaped, the result of the additive modification will be smaller or larger than x with the same probability. Results which are close to x will be more likely than such that are very distant. One example for such a distribution is the normal distribution. Another example is the bell-shaped random number generator used by Worakul et al. [2255, 2256], defined here as Algorithm 28.4. It is algorithmically close to the polar form of the Box-Muller transform for the normal distribution (see Algorithm 28.3) but differs in the way the internal variable w is created. The function $random_{bs}(\mu, \sigma)$ creates a new random number according to this distribution, with an expected value μ and the standard deviation σ.

You may have wondered about the factor 0.5513 in the algorithm. This number "normalizes" the standard deviation of the bell-shaped distribution, since $D^2\left[r(y) = \ln\left(\frac{1-y}{y}\right)\right] \neq 1$. We can show this by first determining the cumulative distribution function $F_X(x)$ for $r(y)$ in Equation 28.311 and then differentiating in order to obtain the probability density function f_Xx in Equation 28.313.

Algorithm 28.4: $y \longleftarrow \text{random}_{bs}(\mu, \sigma)$

Input: μ: the mean value of the bell-shaped distribution
Input: σ: the approximate standard deviation of the bell-shaped distribution
Data: w: a uniformly distributed random number $w \in (0, 1)$
Output: y: a bell-shaped distributed random number

```
1 begin
2     repeat
3         w ⟵ random_u()
4     until (w > 0) ∧ (w < 1)
5     y ⟵ μ + σ * 0.5513 * ln((1−r)/r)
6     return r
7 end
```

$$F_X(x) \equiv r^{-1}(0,1) \tag{28.309}$$

$$x = r(y) = \ln\left(\frac{y}{1-y}\right) \tag{28.310}$$

$$F_X(x) = y = \frac{e^x}{1+e^x} \tag{28.311}$$

$$f_X(x) = F_X(x)\frac{dx}{dy} \tag{28.312}$$

$$\left(\frac{e^x}{1+e^x}\right)\frac{dx}{dy} = \frac{e^x\left(1+e^x\right) - e^x\left(e^x\right)}{\left(1+e^x\right)^2}$$

$$f_X(x) = \frac{e^x}{\left(1+e^x\right)^2} \tag{28.313}$$

Unfortunately, here it stops. We can neither apply Equation 28.56 on page 473 or Equation 28.63 on page 474 in order to determine the expected value or the variance, since both will result in integrals that the author[87] cannot compute. However, it is easy to see that $EX = 0$, since $r(y)$ is point symmetric around 0.5. The value $D^2X \approx 3.28984$ I can only determine numerically with the small Java program Listing 28.1 which bases on the idea that we can assume the uniform random numbers to be uniformly distributed in $(0, 1)$ (of course). Hence we can simulate a "complete sample" by iterating over codeili = 1 to T-1 and take `i/T` as input for $r(y)$. Since we step over all `i` from `1` to `T-1`, this resembles a uniform distribution and also leaves away the special cases $y = 0$ ($\sim$`i=0`) and $y = 1$ ($\sim$`i=T`). Furthermore, we can skip half of the steps since our distribution is symmetric. Well, $EX = 0$ if $\mu = 0$ and therefore we can simplify $D^2X = E[X] - (EX)^2$ (see Equation 28.61 on page 474) to $D^2X = E[X]$.

This method is, of course, very crude and subject to numerical errors in the floating point computations. However, with $D^2X \approx 3.28984$ and $DX = \sqrt{D^2X} \approx 1.8138$ we know that we have to scale $r(y)$ by $\frac{1}{DX} \approx 0.5513$ (see Equation 28.65 on page 474) so the standard deviation the bell-shaped distribution $\text{random}_{bs}(\mu, \sigma)$ will become $D[\text{random}_{bs}(\mu, \sigma)] \approx \sigma$.

[87] Yes. I suck in maths.

```
long i, max;
double sum2, v;

max   = 10000000;
sum2  = 0;
v     = 0;

// distribution is symmetric -> iterate one wing
for (i = (max>>1); i < max; i++) {
  v = Math.log(((double) (max - i)) / ((double) i));
  sum2 += (v * v); //sum up the squares of the single terms
  }

System.out.print(sum2 / (max - (max>>1)));
```

Listing 28.1: Approximating D^2X of $r(y)$.

28.10 List of Functions

28.10.1 Gamma Function

Definition 28.71 (Gamma Function). The Gamma function[88] $\Gamma : \mathbb{C} \mapsto \mathbb{R}$ is the extension of the factorial (see Definition 28.8 on page 467) to the real and complex numbers. For complex numbers $z \in \mathbb{C}$ with a positive real part $\text{Re}(z) > 0$ it is defined as:

$$\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt \tag{28.314}$$

Furthermore, the following equations hold for the gamma function.

$$\Gamma(z+1) = z\Gamma(z) \tag{28.315}$$

$$\Gamma(1) = 1 \tag{28.316}$$

$$\Gamma(z) = (z-1)! \; \forall \, z \in \mathbb{N} \tag{28.317}$$

$$\Gamma(z) = \lim_{n \to \infty} \frac{n! n^z}{z(z+1)..(z+n)} \tag{28.318}$$

$$\Gamma(z) = \frac{e^{\gamma z}}{z} \prod_{n=1}^{\infty} \left(1 + \frac{z}{n}\right)^{-1} e^{\frac{z}{n}} \tag{28.319}$$

γ in Equation 28.319 denotes the Euler-Mascheroni constant[89].

$$\gamma = \lim_{n \to \infty} \left[\left(\sum_{k=1}^{n} \frac{1}{k} \right) - \log n \right] = \int_0^\infty \left(\frac{1}{\lfloor x \rfloor} - \frac{1}{x} \right) dx \tag{28.320}$$

$$\approx 0.57721566490153286060651209008240243\ldots \tag{28.321}$$

28.10.2 Riemann Zeta Function

Definition 28.72 (Riemann Zeta Function). The Riemann zeta function[90] $\zeta(s)$ [1733] is the function of the complex variable s defined as

[88] http://en.wikipedia.org/wiki/Gamma_function [accessed 2007-09-30]
[89] http://en.wikipedia.org/wiki/Euler-Mascheroni_constant [accessed 2007-09-30]
[90] http://en.wikipedia.org/wiki/Riemann_zeta_function [accessed 2008-08-24]

$$\zeta(s) = \sum_{i=1}^{+\infty} \frac{1}{i^s} = \prod_{\forall primes\ p} \frac{1}{1 - p^{-s}} \tag{28.322}$$

Some values of the zeta function are listed in Table 28.22.

s	$\zeta(n)$
0	$\zeta(0) = -1/2$
1/2	$\zeta(1/2) \approx -1.460\,354\,508\,809\,586\,8$
1	$\zeta(1) \Rightarrow \infty$
3/2	$\zeta(3/2) \approx 2.612$
2	$\zeta(2) \approx 1.645$
5/2	$\zeta(5/2) \approx 1.341$
3	$\zeta(3) \approx 1.202$
7/2	$\zeta(7/2) \approx 1.127$
6	$\zeta(6) \approx 1.0173$

Table 28.22: Some values of the Riemann zeta function.

29

Clustering

Clustering algorithms[1] divide a dataset into several disjoint subsets. All elements in such a subset share common features like, for example, spatial proximity. Clustering has many different applications like:

1. Data Mining and Data Analysis [133, 610, 1430, 183],
2. Information Processing and Information Management [187, 1151, 2265, 1770],
3. Pattern Recognition [203, 2156, 1172],
4. Image Processing [1102, 1817], and
5. Medicine [1868, 477, 2316].

Definition 29.1 (Clustering). Clustering is the unsupervised classification of patterns (observations, data items, or feature vectors) into groups (clusters) [1029]. With clustering, one dataset is partitioned into subsets (clusters), so that the data in each subset (ideally) share some common trait - often proximity according to some defined distance measure. Figure 29.1 illustrates a possible result C of the application of a clustering algorithm to a set A of elements with two features.

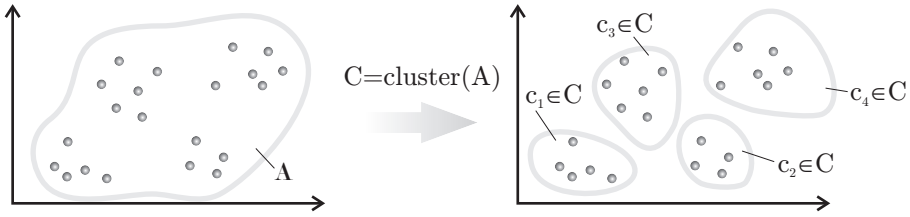


Figure 29.1: A clustering algorithm applied to a two-dimensional dataset A.

In the field of global optimization there is another application for clustering algorithms. For many problems the set of optimal solutions $X^\star$ is very large or even infinite. An optimization algorithm then cannot be able to store or return it on the whole. Therefore, clustering techniques are often used in order to reduce the optimal set while not losing its characteristics – the diversity of the individuals included in the "current optimal set" is preserved, just their number is reduced. Especially in elitist evolutionary algorithms (see Definition 2.4 on page 103) which maintain an archive of the best individuals currently known.

Data clustering algorithms are either hierarchical or partitional. A hierarchical algorithm uses previously established clusters to successively find new clusters. The result of such an algorithm is a hierarchy of clusters. Partitional algorithms, on the other hand, determine

[1] http://en.wikipedia.org/wiki/Data_clustering [accessed 2007-07-03]

all clusters at once. In the context of this book, we do only need the division of a set into clusters – a hierarchy of this division is unnecessary.

There also exist so-called fuzzy clustering[2] [1106, 1216] methods that do not create clear divisions but assign a vector of probabilities to each element. This vector contains a component for each cluster which denotes the probability of the element to belong to it. Again, in the context of this book, we only regard clustering algorithms that group each data element to exactly one single cluster. Therefore, we define a clustering algorithm as follows:

Definition 29.2 (Clustering Algorithm). A clustering algorithm $C = \text{cluster}(A)$ constructs a set C consisting of elements which are disjoint subsets of a set A and, if united, cover A completely (see also Figure 29.1).

$$\begin{aligned} C = \text{cluster}(A) \Rightarrow\ & \forall c \in C,\ \forall a \in c \Rightarrow a \in A \wedge \\ & \forall c_1 \neq c_2 \wedge c_1, c_2 \in C \Rightarrow c_1 \cap c_2 = \emptyset \wedge \\ & \forall a \in A\ \exists c \in C : a \in c \qquad (29.1) \\ & \textit{deduced:} \bigcup_{\forall c \in C} = A \qquad (29.2) \\ & \textit{deduced:} C \subset \mathcal{P}(A) \qquad (29.3) \end{aligned}$$

For the last deduced formula see the definition of the power set $\mathcal{P}$, Definition 27.9 on page 458.

There is however one important fact that must not be left unsaid here: Although we define clustering algorithms in terms of sets for simplicity, they are actually applied to lists. A set can contain the *same* element only once, hence $\{1, 2, 1\} = \{1, 2\}$. A clustering algorithm however may receive an input A that contains multiple *equal* elements. This is our little dirty backdoor here, we consider $A = \{a_1, a_2, .., a_n\}$ as the input set and allow its elements to have *equal values*, such as $a_1 = 1$, $a_2 = 2$, and $a_3 = 1$. When performing the clustering, we only consider the symbols $a_1 \ldots a_n$. This allows us to use straightforward and elegant set-based definitions as done in Definition 29.2 without loss of generality.

Definition 29.3 (Partitions in Clustering). We define the set $\mathfrak{C}$ of all possible partitions of A into clusters C. Furthermore, the subset $\mathfrak{C}_k \subseteq \mathfrak{C}$ is the set of all partitions of A into k clusters. The number of possible configurations $\mathfrak{C}_k$ for any given k equals the Sterling number $S(|A|, k)$ [221].

$$\begin{aligned} \forall C \in \mathfrak{C} \Leftrightarrow\ & \forall c \in C,\ \forall a \in c \Rightarrow a \in A \wedge \\ & \forall c_1, c_2 \in C \Rightarrow c_1 \cap c_2 = \emptyset \wedge \\ & \forall a \in A\ \exists c \in C\ :\ a \in c \qquad (29.4) \\ C \in \mathfrak{C}_k \Leftrightarrow\ & C \in \mathfrak{C} \wedge |C| = k \qquad (29.5) \\ |\mathfrak{C}_k| =\ & S(|A|, k) = \frac{1}{k!} \sum_{i=1}^{k} (-1)^{k-i} \binom{k}{i} i^n \qquad (29.6) \\ |\mathfrak{C}| =\ & \sum_{k=1}^{n} |\mathfrak{C}_k| = \sum_{k=1}^{n} S(|A|, k) \qquad (29.7) \end{aligned}$$

On the elements a of the set A which are subject to clustering, we impose an simple restriction: Although we allow any sort of elements a in A, we assume that to each such element a there is assigned exactly one single $\alpha(a) \in \mathbb{R}^n$. In other words, there exists a function $\alpha : A \mapsto \mathbb{R}^n$ which relates the features of each element a of A to a vector of real numbers. This allows us to apply distance metrics and such and such.

[2] http://en.wikipedia.org/wiki/Fuzzy_clustering [accessed 2007-07-03]

In the context of global optimization, the elements a would for example be the solution candidates like evolved programs in the population A and the function $\alpha(a)$ then would correspond to the values of their objective functions $f \in F$.

From now on, we will be able treat the elements a like vectors of real numbers (if needed) without loss of generality. Note that even though we assume that there exists a binary relation which assigns a real vector to each element of A, this is not necessarily the case for the opposite direction. Picking up the previous example it is most probably not likely to have one program for each possible combination of objective values.

Definition 29.4 (Centroid). The centroid[3] [4] of a cluster is its center, the arithmetic mean of all its points to put it plain and simple.

$$\text{centroid}(c) = \frac{1}{|c|} \sum_{\forall a \in c} a \tag{29.8}$$

29.1 Distance Measures

Each clustering algorithm needs some form of distance measuring, be it between two elements or between two clusters. Therefore we define the prototype of a distance measurement function as follows:

Definition 29.5 (Distance Measure). A distance measurement function dist rates the distance between two elements of the same type (set) as positive real number which is the bigger the bigger the distance between the two elements is.

$$\text{dist}(a, b) \in \mathbb{R}^+,\ a, b \in A \tag{29.9}$$

29.1.1 Distance Measures for Strings of Equal Length

Definition 29.6 (Hamming Distance). For two tuples a and b of the same length, the Hamming [882] distance[4] $\text{dist}_{Ham}(a, b)$ is defined as the number of locations in which a and b differ.

$$\text{dist}_{Ham}(a, b) = |\{i : a_{[i]} \neq b_{[i]},\ \forall 0 \leq i < |a|\}|\ \forall a, b : \text{len}(a) = \text{len}(b) \tag{29.10}$$

The Hamming distance is used in many error-correction schemes, since it also equals to the number of single substitutes required to change one string into another one. The Hamming distance of `100101` and `101001` is 2 whereas the Hamming distance of `Hello World` and `Hello Earth` is 5.

29.1.2 Distance Measures for Real-Valued Vectors

As already mentioned in Chapter 29, we assume that there is a real-values vector in $\mathbb{R}^n$ assigned to each element $a \in A$ by an implicit $\alpha : A \mapsto \mathbb{R}^n$-function. Therefore, the distance measures introduced here can be used for all A subject to clustering.

Definition 29.7 (Manhattan Distance). The Manhattan distance[5] $\text{dist}_{Man}(\mathbf{a}, \mathbf{b})$ denotes the sum of the absolute distances of the coordinates of the two vectors $\mathbf{a}$ and $\mathbf{b}$.

$$\text{dist}_{Man}(\mathbf{a}, \mathbf{b}) = \sum_{i=1}^{n} |\mathbf{a}_{[i]} - \mathbf{b}_{[i]}|\ \forall \mathbf{a}, \mathbf{b} \in \mathbb{R}^n \tag{29.11}$$

[3] `http://en.wikipedia.org/wiki/Centroid` [accessed 2007-07-03]
[4] `http://en.wikipedia.org/wiki/Hamming_distance` [accessed 2007-07-03]
[5] `http://en.wikipedia.org/wiki/Manhattan_distance` [accessed 2007-07-03]

Thus, the Manhattan distance of $(1,2,3)^T$ and $(3,2,1)^T$ is 4.

Definition 29.8 (Euclidian Distance). The Euclidian distance[6] $\text{dist}_{eucl}(\mathbf{a},\mathbf{b})$ is the "ordinary" distance of two points (denoted by the two vectors $\mathbf{a}$ and $\mathbf{b}$) in Euclidian space. This value is obtained by applying of the Pythagorean theorem[7].

$$\text{dist}_{eucl}(\mathbf{a},\mathbf{b}) = \sqrt{\sum_{i=1}^{n} (\mathbf{a}_{[i]} - \mathbf{b}_{[i]})^2} \; \forall \mathbf{a},\mathbf{b} \in \mathbb{R}^n \tag{29.12}$$

Therefore, the Euclidian distance of $(1,2,3)^T$ and $(3,2,1)^T$ is $\sqrt{8}$.

Definition 29.9 (Norm). A vector norm[8], denoted by $||\mathbf{a}||$ is a function which assigns a positive length or size to all vectors $\mathbf{a}$ in a vector space (or set) $A \subseteq \mathbb{R}^n$, other than the zero vector.

Some common norms of the element $\mathbf{a} \in A \subseteq \mathbb{R}^n$ are:

1. The Manhattan norm[9]:
$$||\mathbf{a}||_1 = \sum_{i=1}^{n} |\mathbf{a}_{[i]}| \tag{29.13}$$
2. The Euclidian norm:
$$||\mathbf{a}||_2 = \sqrt{\sum_{i=1}^{n} (\mathbf{a}_{[i]})^2} \tag{29.14}$$
3. The p-norm is a generalization of the two examples above:
$$||a||_p = \left(\sum_{i=1}^{n} |\mathbf{a}_{[i]}|^p \right)^{\frac{1}{p}} \tag{29.15}$$
4. The infinity norm[10] is the special case of the p-norm for $p \to \infty$:
$$||\mathbf{a}||_\infty = \max\left\{ |\mathbf{a}_{[1]}|, |\mathbf{a}_{[2]}|, .., |\mathbf{a}_{[n]}| \right\} \tag{29.16}$$

Such norms can be used as distance measures, and we hence define a new distance measurement function as:

$$\text{dist}_{n,p}(\mathbf{a},\mathbf{b}) = ||\mathbf{a}-\mathbf{b}||_p \; \forall \mathbf{a},\mathbf{b} \in A \subseteq \mathbb{R}^n \tag{29.17}$$
$$\text{dist}_{Man} \equiv \text{dist}_{n,1} \tag{29.18}$$
$$\text{dist}_{eucl} \equiv \text{dist}_{n,2} \tag{29.19}$$

If the places of the vectors $\mathbf{a}$ have different ranges, for example $\mathbf{a}_{[1]} \in [0...1]$ and $\mathbf{a}_{[2]} \in [0..100\,000]$, a norm of the difference of two such vectors may not represent their true "semantic" distance. Here, the contribution of the first elements of two vectors $\mathbf{a}$ and $\mathbf{b}$ to their distance will most likely be negligible. However, the two vectors $(0,0)^T$ and $(1,100)^T$ may be considered "more different" than $(0,0)^T$ and $(0.1,110)^T$, since they differ the whole range in their first elements. Therefore, an additional distance measure, the $\text{dist}^n_{n,p}$distance is defined which normalizes the vector places before finally computing the norm.

$$\text{dist}^n_{n,p}(\mathbf{a},\mathbf{b}) = \left(\sum_{i=1}^{n} \left| \begin{cases} \frac{\mathbf{a}_{[i]}-\mathbf{b}_{[i]}}{\mathbf{range(A)}_{[i]}} & \text{if } \mathbf{range(A)}_{[i]} > 0 \\ \mathbf{a}_{[i]} - \mathbf{b}_{[i]} & \text{otherwise} \end{cases} \right|^p \right)^{\frac{1}{p}} \tag{29.20}$$

[6] http://en.wikipedia.org/wiki/Euclidean_distance [accessed 2007-07-03]
[7] http://en.wikipedia.org/wiki/Pythagorean_theorem [accessed 2007-07-03]
[8] http://en.wikipedia.org/wiki/Vector_norm [accessed 2007-07-03]
[9] http://en.wikipedia.org/wiki/Taxicab_geometry [accessed 2007-07-03]
[10] http://en.wikipedia.org/wiki/Maximum_norm [accessed 2007-07-03]

29.1.3 Elements Representing a Cluster

We already stated that there is not necessarily an $a \in A$ assigned to each real vector in $\mathbb{R}^n$. Thus, there also does not necessarily exist an element a in the center centroid$c$ of a cluster c. For our purposes in this book, we are however interested in elements representing clusters. Since I have not found any other in literature, we will call such elements nuclei. We can define different functions nucleus($c \in C$) to compute such nuclei which, in turn, depend on a distance measure. We will again abbreviate this distance function by dist. dist is an implicit parameter which can be replaced by any of the functions introduced before. Also again, the default setting is $\text{dist} = \text{dist}_{eucl} \equiv \text{dist}_{n,2}$.

The first possible nucleus method, nucleus_c, would be to take the element which is closest to the centeroid centroid$c$ of the cluster c:

$$n \in c = \text{nucleus}(c) \Leftrightarrow \text{dist}(a, \text{centroid}(c)) \geq \text{dist}(n, \text{centroid}(c)) \ \forall a \in c \tag{29.21}$$

29.1.4 Distance Measures Between Clusters

In order to determine the distance between two clusters, another set of distance measures can be applied. Such distance measures usually will compute the distance between two clusters as a function of the distances between their elements which is, in turn, defined using a secondary distance function. We will abbreviate this secondary distance function by dist which can be replaced by any of the functions named in the above subsections. We assume it to be an implicit parameter with the default value $\text{dist} = \text{dist}_{eucl} \equiv \text{dist}_{n,2}$. Let c_1 and c_2 be two clusters in C, then we can define the following distance measures between them:

1. The maximum distance between the elements of the two clusters (also called complete linkage):

$$\text{dist}_{max}(c_1, c_2) = \max\left\{\text{dist}(a, b)\,,\ \forall a \in c_1, b \in c_2\right\};\ \forall c_1, c_2 \in C \tag{29.22}$$

2. The minimum distance between the elements of the two clusters (also called single linkage):

$$\text{dist}_{min}(c_1, c_2) = \min\left\{\text{dist}(a, b)\,,\ \forall a \in c_1, b \in c_2\right\};\ \forall c_1, c_2 \in C \tag{29.23}$$

3. The mean distance between the elements of the two clusters (also called average linkage):

$$\text{dist}_{avg}(c_1, c_2) = \frac{1}{|c_1| * |c_2|} \sum_{\forall a \in c1} \sum_{\forall b \in c2} \text{dist}(a, b)\,;\ \forall c_1, c_2 \in C \tag{29.24}$$

4. The increase in variance $\text{dist}_{var}(a, b)$ if the clusters were merged.
5. The distance of their centers:

$$\text{dist}_{cent}(c_1, c_2) = \text{dist}(\text{centroid}(c_1)\,, \text{centroid}(c_2))\,;\ \forall c_1, c_2 \in C \tag{29.25}$$

6. The distance of their nuclei computed by the nucleus function nucleus (see the definition of nucleus in Section 29.1.3):

$$\text{dist}_{nucl}(c_1, c_2) = \text{dist}(\text{nucleus}(c_1)\,, \text{nucleus}(c_2))\,;\ \forall c_1, c_2 \in C \tag{29.26}$$

29.2 Clustering Algorithms

29.2.1 Cluster Error

The most commonly used partitional clustering strategies are based on the square error criterion. The general aim is to obtain a partition which minimizes the square error for a given number k [1028] which we generalize to fit any given distance measure dist:

Definition 29.10 (Clustering Error). We define the error error_c inside a cluster as the sum of the distances of its elements from its center basing on a distance measure function. The total error of a partition error_p is then the sum of all the errors of the clusters included. Normally, we will use $\text{dist}_{eucl} \equiv \text{dist}_{n,2}$ as distance measure.

$$\text{error}_c(c) = \sum_{\forall a \in c} \text{dist}(a, \text{centroid}(c)) \tag{29.27}$$

$$\text{error}_p(C) = \sum_{\forall c \in C} \text{error}_c(c) = \sum_{\forall c \in C} \sum_{\forall a \in c} \text{dist}(a, \text{centroid}(c)) \tag{29.28}$$

Normally, this error is minimized under the premise of a fixed number of clusters $k = |C|$. Then, an optimum configuration $C^\star$ is searched within the set $\mathfrak{C}_k$ of all such partitions of A into k clusters C. This optimum $C^\star$ is defined by $\text{error}_p(C^\star) = \min\{\text{error}_p(C)\ \forall C \in \mathfrak{C}_k\}$. Since testing all possible configurations C is too expensive (see Equation 29.7), finding the optimum $C^\star$ is an optimization tasks itself. Doing so inside an optimization process itself hence only rarely is applicable. Here we will introduce some algorithms which approximate good C.

29.2.2 k-means Clustering

k-means clustering[11] [1343, 210, 2111] partitions the data points $a \in A$ into k disjoint subsets $c \subseteq A,\ c \in C \subseteq \mathfrak{C}_k$. It tries to minimize the sum of all distance of the data points and the centers of the clusters they belong to. In general, the algorithm does not achieve a global minimum of over the assignments. Despite this limitation, k-means clustering is used frequently as a result of its ease of implementation. [2191]

k-means clustering works approximately as follows [1028]:

1. Select an initial partition of k clusters.
2. Create a new partition by assigning each $a \in A$ to the cluster with the closest center. Repeat this until the partition does not change anymore.
3. Modify the cluster set by merging, dividing, deleting or creating cluster. If the clustering error of the new partition is smaller than the error of the previous one then go back to step 2.

In order to perform the modification of the cluster set, we introduce a function called kMeansModify obeying the following conditions.

$$\begin{aligned} C_{new} = \text{kMeansModify}_k(C) \Rightarrow \forall a \in c_1 \in C\ \exists c_2 \in C_{new}\ &: a \in c_2 \wedge \\ \forall a \in c_2 \in C_{new}\ \exists c_1 \in C\ &: a \in c_1 \end{aligned} \tag{29.29}$$

In other words, kMeansModify translates one set of clusters C to another one C_{new} by redeeming Definition 29.2 on page 536. One (crude) example for an implementation of kMeansModify is specified as Algorithm 29.1.

We demonstrate how k-means clustering works in Algorithm 29.2. As distance measure dist (lines 23 and 25) usually the Euclidian distance between the centroids of the clusters c_1 and c_2, $\text{dist}_{cent}(c_1, c_2)$, see page 539, is used.

[11] http://en.wikipedia.org/wiki/K-nearest-neighbor_estimator [accessed 2007-07-03]

Algorithm 29.1: $C_{new} \longleftarrow \text{kMeansModify}_k(C)$

Input: [implicit] k: the number of clusters wanted, $k \leq |A|$
Input: [implicit] dist: the distance measure between clusters to be used
Input: [implicit] dist_2: the distance measure between elements to be used
Input: C: the list of clusters c to be modified
Data: m: the index of the cluster $C_{[m]}$ with the lowest error
Data: n: the index of the cluster $C_{[n]}$ nearest to $C_{[m]}$
Data: s: index of the cluster $C_{[s]}$ with the highest error
Output: C_{new}: the modified list of closters

```
1  begin
2  |  m ⟵ m : error_c(C_[m]) = min{C_[i] ∀i ∈ [0, k−1]}
3  |  n ⟵ n : dist(C_[m], C_[n]) = min{dist(C_[m], C_[i]) ∀i ∈ [0, k−1] \ {m}}
4  |  s ⟵ s : error_c(C_[s]) = max{error_c(C_[i]) ∀i ∈ [0, k−1] \ {m, n}}
5  |  C_[m] ⟵ C_[m] ∪ C_[n]
6  |  a ⟵ a ∈ C_[s] : dist_2(a, centroid(C_[s])) ≥ dist_2(b, centroid(C_[s])) ∀b ∈ C_[s]
7  |  C_[n] ⟵ {a}
8  |  C_[s] ⟵ C_[s] \ {a}
9  |  return B
10 end
```

29.2.3 n^{th} Nearest Neighbor Clustering

The n^{th} nearest neighbor clustering algorithm is defined in the context of this book only. It creates at most k clusters where the first $k-1$ clusters contain exactly one element. The remaining elements are all together included in the last cluster. The elements of the single-element clusters are those which have the longest distance to their n^{th}-nearest neighbor. This clustering algorithm is suitable for reducing a large set to a smaller one which contains still the most interesting elements (those in the single-element clusters). It has relatively low complexity and thus runs fast, but on the other hand has the setback that dense aggregations of $\geq n$ elements will be put into the "rest elements"-cluster. For n, normally a value of $n = \sqrt{k}$ is used.

n^{th} nearest neighbor clustering uses the k^{th} nearest neighbor distance function $\text{dist}^{\rho}_{nn,k}$ introduced in Definition 28.63 on page 506 with its parameter k set to n. Do not mix this parameter up with the parameter k of this clustering method – although they have the same name, they are not the same. I know, I know, this is not pretty.

Notice that Algorithm 29.3 should only be applied if all the elements $a \in A$ are unique, i.e., there exists no two equal elements in A) which is, per definition, true for all sets. In a real implementation, a preprocessing step should remove are duplicates from A before clustering is performed. Especially our home-made nearest neighbor clustering variant is unsuitable to process lists containing the same elements multiple times. Since all equal elements have the same distance to their n^{th} neighbor, it is likely that the result of the clustering is very unsatisfying since one element may occur multiple times whereas a variety of different other elements is ignored. Therefore, the aforementioned preprocessing should be applied, which may have the drawback that we could possible obtain a set C with less than k clusters. In the Sigoa system's implementation of the n^{th} nearest neighbor clustering, only one instance of each group of equal elements in A is permitted to become a single-node cluster per run and multiple runs are performed until k clusters have been created.

29.2.4 Linkage Clustering

The linkage method [1466, 2329] is used to create a set C containing at most k clusters. This algorithm initially creates a cluster of each single element in the set A. This set C of cluster c is reduced melting together the two closest clusters iteratively. Again, the distance

Algorithm 29.2: $C \longleftarrow \text{kMeansCluster}_k(A)$

Input: A: the set of elements a to be clustered
Input: [implicit] k: the number of clusters wanted, $0 < k \leq |A|$
Input: [implicit] dist: the distance measures between clusters to be used
Input: [implicit] dist_2: the distance measures between elements to be used
Input: [implicit] kMeansModify: a function that modifies the cluster set
Data: C: the tuple of clusters c computed, $|C| = k$
Data: A_{cpy}: a temporary copy of A used for initialization
Data: C_{old}: the cluster set of the previous inner iteration
Data: C_{new}: the cluster set of the current inner iteration
Data: i: a counter variable for the loops
Data: d: the distance between the cluster $\{a\}$ and the current cluster in C_{old}
Data: d_{min}: the minimum distance between $\{a\}$ and any cluster in C_{old}
Data: i_{min}: the index of that cluster with the minimum distance in C_{old}
Output: c: the set of clusters – all the items of the tuple B represented as set

```
1  begin
2      A_cpy ⟵ A
3      k ⟵ min{k, |A|}
4      C_new ⟵ createList(k, ∅)
5      i ⟵ len(C_new) − 1
6      while i > 0 do
7          C_new[i] ⟵ {a ∈ A_cpy}
8          A_cpy ⟵ A_cpy \ C_[i]
9          i ⟵ i − 1
10     C_new[0] ⟵ A_cpy
11     repeat
12         C ⟵ C_new
13         C_new ⟵ kMeansModify_k(C_new)
14         repeat
15             C_old ⟵ C_new
16             i ⟵ len(C_new) − 1
17             while i > 0 do
18                 C_new[i] ⟵ ∅
19                 i ⟵ i − 1
20             foreach a ∈ A do
21                 i ⟵ len(C_old) − 1
22                 i_min ⟵ 0
23                 d_min ⟵ dist({a}, C_old[0])
24                 while i > 0 do
25                     d ⟵ dist({a}, C_old[i])
26                     if d < d_min then
27                         d_min ⟵ d
28                         i_min ⟵ i
29                     i ⟵ i − 1
30                 C_new[i_min] ⟵ C_new[i_min] ∪ {a}
31         until C_old = C_new
32     until error_p(C) ≤ error_p(C_new)
33     return listToSet(C)
34 end
```

Algorithm 29.3: $C \longleftarrow \text{nNearestNeighborCluster}_k(n)\, A$

Input: A: the set of elements a to be clustered
Input: [implicit] k: the number of clusters wanted ($0 < k \leq |A|$)
Input: [implicit] n: index for the nearest neighbors
Input: [implicit] dist: the distance measure to be used
Data: L: the sorted list of elements
Data: i: the counter variable
Output: C: the set of clusters c computed, $|C| = k$

```
1  begin
2      L ⟵ sortList_d(setToList(A), dist^ρ_{nn,k}dist)
3      i ⟵ min{k, |L|} − 2
4      C ⟵ ∅
5      while i ≥ 0 do
6          C ⟵ C ∪ {L_[i]}
7          A ⟵ A \ L_[i]
8          i ⟵ i − 1
9      return C ∪ {A}
10 end
```

measure function dist (see line 11 of Algorithm 29.4) used can be any of distance measures already introduced.

According to the cluster distance measure dist chosen, linkageCluster realizes different types of linkage clustering algorithms[12] (see Section 29.1.4 on page 539):

1. If $\text{dist}(c_1, c_1) = \text{dist}_{max}(c_1, c_1)$ denotes the maximum distance of the elements in two clusters, complete linkage clustering is performed.
2. If $\text{dist}(c_1, c_1) = \text{dist}_{avg}(c_1, c_1)$ denotes the mean distance of the elements in two clusters, average linkage clustering is performed.
3. If $\text{dist}(c_1, c_1) = \text{dist}_{min}(c_1, c_1)$ denotes the minimum distance of the elements in two clusters, single linkage clustering is performed.

29.2.5 Leader Clustering

The leader clustering algorithm is a very simple one-pass method to create clusters. Basically, we begin with an empty leader list and an empty set of clusters. Step by step the elements a are extracted from the set A subject to clustering. a is then compared to the elements in the leader list in order to find one leader l with $\text{dist}(a, l)$ smaller than a specified maximum distance D. If such a leader exists, a is added to its cluster, otherwise a becomes leader of a new cluster containing only itself. The leader clustering can either be performed by using the first best leader l found with $\text{dist}(a, l) < D$ and assign a to its cluster ([255], Algorithm 29.5) or by comparing a to all possible leaders and thus finding the leader closest to a $\text{dist}(a, l) < \text{dist}(a, l2)\ \forall l2 \in \textit{leaders}$ ([84], Algorithm 29.6).

[12] http://en.wikipedia.org/wiki/Data_clustering#Agglomerative_hierarchical_clustering [accessed 2007-07-03]

Algorithm 29.4: $C \longleftarrow linkageClusteribkA$

Input: A: the set of elements a to be clustered
Input: [implicit] k: the number of clusters wanted ($0 < k \leq |A|$)
Input: [implicit] dist: the distance measure to be used
Input: [implicit] dist_2: the distance measure between elements a to be used by dist
Data: c_1: the first cluster to investigate
Data: c_2: the second cluster to investigate
Data: d: the distance between the clusters r_1 and r_2 currently investigated
Data: d_{min}: the minimum distance between two clusters c_{r1}, c_{r2} found in the current iteration
Data: c_{r1}: the first cluster of the nearest cluster pair
Data: c_{r2}: the second cluster of the nearest cluster pair
Output: C: the set of clusters c computed, $|C| = k$

```
1  begin
2      C ⟵ ∅
3      foreach a ∈ A do C ⟵ C ∪ {a}
4      while |C| > k do
5          d_min ⟵ ∞
6          c_r1 ⟵ ∅
7          c_r2 ⟵ ∅
8          foreach c_1 ∈ C do
9              foreach c_2 ∈ C do
10                 if c_1 ≠ c_2 then
11                     d ⟵ dist(c_1, c_2)
12                     if d ≤ d_min then
13                         d_min ⟵ d
14                         b_r1 ⟵ b_1
15                         b_r2 ⟵ b_2
16         C ⟵ C \ c_r1
17         C ⟵ C \ c_r2
18         C ⟵ C ∪ {c_r1 ∪ c_r2}
19     return C
20 end
```

Algorithm 29.5: $C \longleftarrow \text{leaderCluster}_D^f(A)$

Input: A: the set of elements a to be clustered
Input: [implicit] D: the maximum distance between an element an a cluster's leader
Input: [implicit] dist: the distance measure to be used
Data: a: an element in A
Data: i: a counter variable
Data: L: the list of cluster leaders
Output: C: the list of clusters c computed

```
1  begin
2      L ⟵ ()
3      B ⟵ ()
4      foreach a ∈ A do
5          i ⟵ len(L) − 1
6          while i ≥ 0 do
7              if dist(L[i], a) ≤ D then
8                  C[i] ⟵ C[i] ∪ {a}
9                  i ⟵ −2
10             i ⟵ i − 1
11         if i ≥ −1 then
12             L ⟵ addListItem(L, a)
13             C ⟵ addListItem(C, {a})
14     return listToSet(C)
15 end
```

Algorithm 29.6: $C \longleftarrow \text{leaderCluster}_D^a(A)$

Input: A: the set of elements a to be clustered
Input: [implicit] D: the maximum distance between an element an a cluster's leader
Input: [implicit] dist: the distance measure to be used
Data: a: an element in A
Data: i: a counter variable
Data: L: the list of cluster leaders
Output: C: the list of clusters c computed

```
1  begin
2      L ⟵ ()
3      B ⟵ ()
4      foreach a ∈ A do
5          i ⟵ len(L) − 1
6          j ⟵ 0
7          while i > 0 do
8              if dist(L[i], a) < dist(L[j], a) then j ⟵ i
9          if dist(L[j], a) ≤ D then
10             C[j] ⟵ C[j] ∪ {a}
11         else
12             L ⟵ addListItem(L, a)
13             B ⟵ addListItem(C, {a})
14     return listToSet(C)
15 end
```

30

Theoretical Computer Science

30.1 Introduction

Theoretical computer science[1] is the branch of computer science[2] that deals with the rather mathematical, logical, and abstract aspects of computing. It subsumes areas like algorithmic theory, complexity, the structure programming languages, and the solvability of problems.

30.1.1 Algorithms and Programs

In this and the following sections, we want to gain insight into the topic of algorithms, both in local and distributed systems. This seems to be appropriate, since any global optimization technique which we will discuss in this book is an algorithm. Often even a rather complicated one. Sometimes we even want to use several computers to solve an optimization problem cooperatively. Thus, we should know about the properties and theory of algorithms as well as of distributed systems.

The second reason is that many example applications discussed in this book will concern the automated syntheses of distributed algorithms. To understand these, knowledge of the features of distributed algorithms is valuable.

What are Algorithms?

The term *algorithm* comprises essentially all forms of "directives what to do to reach a certain goal". A culinary receipt is an algorithm, for example, since it tells how much of what is to be added to a meal in which sequence and how everything should be heated. The commands inside the algorithms can be very concise or very imprecise, depending on the area of application. How accurate can we, for instance, carry out the instruction "Add a tablespoon of sugar."? Hence, algorithms are a very wide field that there exist numerous different, rather fuzzy definitions for the word algorithm [19, 86, 90, 446, 1213]:

Definition 30.1 (algorithm). According to Whatis.com[3], an algorithm is a procedure or formula for solving a problem. The word derives from the name of the mathematician, Mohammed ibn-Musa al-Khwarizmi, who was part of the royal court in Baghdad and who lived from about 780 to 850. Al-Khwarizmi's work is the likely source for the word algebra as well.

Definition 30.2 (algorithm). Wikipedia[4] says that in mathematics, computing, linguistics, and related disciplines, an algorithm is a procedure (a finite set of well-defined instructions) for accomplishing some task which, given an initial state, will terminate in a defined

[1] http://en.wikipedia.org/wiki/Theoretical_computer_science [accessed 2007-07-03]
[2] http://en.wikipedia.org/wiki/Computer_science [accessed 2007-07-03]
[3] http://searchvb.techtarget.com/sDefinition/0,,sid8_gci211545,00.html [accessed 2007-07-03]
[4] http://en.wikipedia.org/wiki/Algorithm [accessed 2007-07-03]

end-state. The computational complexity and efficient implementation of the algorithm are important in computing, and this depends on suitable data structures.

Definition 30.3 (algorithm). An algorithm is a computable set of steps to achieve a desired result according to the National Institute of Standards and Technology[5].

Definition 30.4 (algorithm). Wolfram MathWorld[6] defines algorithm as a specific set of instructions for carrying out a procedure or solving a problem, usually with the requirement that the procedure terminate at some point. Specific algorithms sometimes also go by the name method, procedure, or technique. The word "algorithm" is a distortion of al-Khwarizmi, a Persian mathematician who wrote an influential treatise about algebraic methods. The process of applying an algorithm to an input to obtain an output is called a computation.

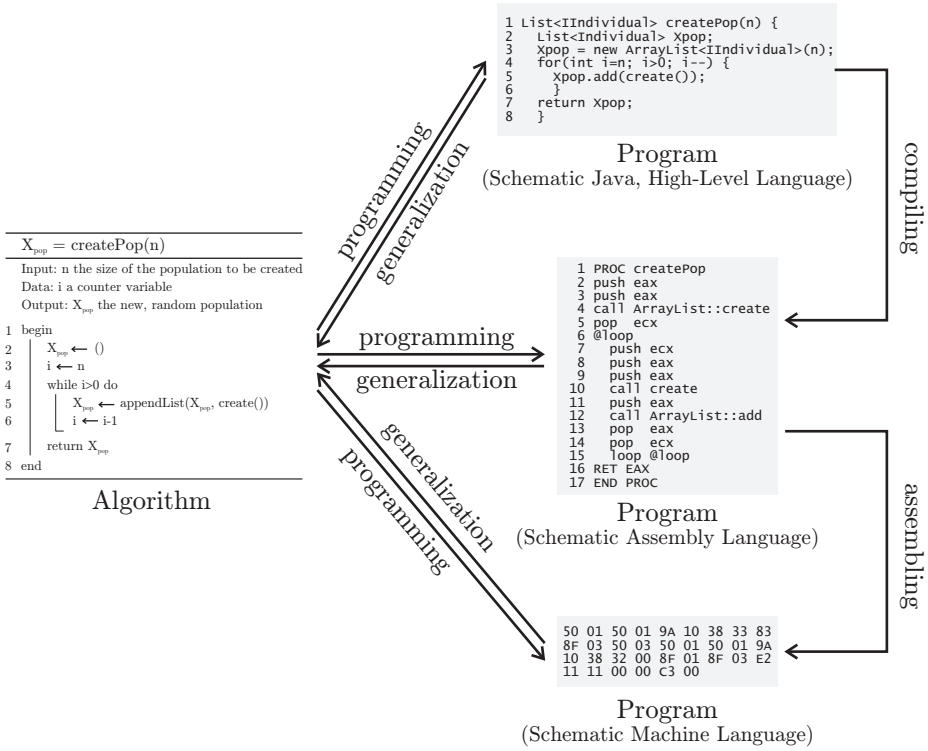


Figure 30.1: The relation between algorithms and programs.

While an algorithm is a set of directions in a representation which is usually understandable for human beings, programs are intended to be processed by machines and are therefore expressed in a more machine-friendly form. Originally, this was machine code. Nevertheless, for more than sixty years [312], huge effort is being spent in order to allow us to write programs in more and more comprehensible syntax. A program is basically an algorithm realized for a given computer or execution environment, as illustrated in Figure 30.1. The difference between programs and algorithms hence today lies primarily in the degree of independence from a given platform and the intention.

[5] http://www.nist.gov/dads/HTML/algorithm.html [accessed 2007-07-03]
[6] http://mathworld.wolfram.com/Algorithm.html [accessed 2007-07-03]

Definition 30.5 (Program). A program[7] is a set of instructions that describe a task or an algorithm to be carried out on a computer. Therefore, the primitive instructions of the algorithm must be expressed either in a form that the machine can process directly (machine code[8] [1622]), in a form that can be translated (1:1) into such code (assembly language [1793], Java byte code [837], etc.), or in a high-level programming language[9] [1350] which can be translated (n:m) into the latter using special software (compiler) [1162].

In Genetic Programming, programs are grown and not algorithms since the evolved structures are always bound to one specific simulation environment. The results may be transformed to algorithms by removing this binding. This process can become very complicated, especially for assembly language or machine code-like programs and there is no automated way for doing this to the knowledge of the author.

Definition 30.6 ((Software) Process). In terms of software, a process[10] is a program that is currently executed. While a program only is a description of what to do, a process is the procedure of actually doing it. In a program for example the number and types of variables are described – in a process they are allocated and used.

Here we should also mention one of the most fundamental principle of electronic data processing[11], the IPO Model[12]. As sketched in Figure 30.2, it consists of three parts:

Figure 30.2: A process in the IPO model.

1. The input (I<u>P</u>O) is an external information or stimulus that enters the system.
2. The processing step (I<u>P</u>O) is the set of all actions taken upon/using the input. In terms of software, these actions are performed by a process which is the running instance of a program.
3. The output (IP<u>O</u>) comprises the results of the computation (processing phase) that leave the system.

30.1.2 Properties of Algorithms

Besides these definitions, algorithms all share the following properties. Well, with few exceptions that we also will elaborate on.

Definition 30.7 (Abstraction). An algorithm describes the process of solving a problem on a certain level of abstraction which is determined by the elementary algorithms and elementary objects it uses and the applied formalism. One of the most important methods of abstraction is the definition and reuse of sub-algorithms.

Definition 30.8 (Discrete). A discrete algorithm works step-wise, i. e., is build on base of atomic executable instructions.

[7] http://en.wikipedia.org/wiki/Computer_program [accessed 2007-07-03]
[8] http://en.wikipedia.org/wiki/Machine_code [accessed 2007-07-04]
[9] http://en.wikipedia.org/wiki/High-level_programming_language [accessed 2007-07-03]
[10] http://en.wikipedia.org/wiki/Process_%28computing%29 [accessed 2007-07-03]
[11] http://en.wikipedia.org/wiki/Electronic_data_processing [accessed 2007-07-03]
[12] http://en.wikipedia.org/wiki/IPO_Model [accessed 2007-07-03]

Definition 30.9 (Finite). The definition of a (static) finite algorithm has a limited length. The sequence of instructions of static finite algorithms is thus finite. During its execution, a (dynamic) finite algorithm uses only a limited amount of memory to store its interim results.

Definition 30.10 (Termination). Each execution of an algorithm terminates after a finite number of steps and returns its results.

Definition 30.11 (Determinism). In each execution step of a deterministic algorithm, there exists at most one way to proceed. If no way to proceed exists, the algorithm has terminated.

Deterministic algorithms do not contain instructions that use random numbers in order to decide what to do or how to modify data. Most of the optimization techniques included in this book are randomized algorithms. They hence are not deterministic. We give an introduction into this matter in Definition 30.18 on page 552.

Definition 30.12 (Determined). An algorithm is determined if it always yields the same results (outputs) for the same inputs.

30.1.3 Complexity of Algorithms

For most problems, there exists more than one approach that will lead to a correct solution. In order to find out which one is the "best", we need some sort of metrics which we can compare [2166, 2223].

The most important measures obtained by analyzing an algorithm[13] are the time that it takes to produce the wanted outcome and the storage space needed for internal data [446]. We call those the time complexity and the space complexity dimensions. The time-complexity denotes how many steps algorithms need until they return their results. The space complexity determines how much memory an algorithm consumes at most in one run. Of course, these measures depend on the input values passed to the algorithm. If we have an algorithm that should decide whether a given number is prime or not, the number of steps needed to find that out will differ if the inputs are 1 or $2^{32582657} - 1$. Therefore, for both dimensions, the best-case, average-case, and the worst-case complexity exist.

In order to compare the time and space requirements of algorithms, some approximative notations have been introduced [1159, 1894, 1162]. As we just have seen, the time and space requirements of an algorithm normally depend on the size of its inputs. We can describe this dependency as a function of this size. In real systems however, the knowledge of the exact dependency is not needed. If we, for example, know that sorting n data elements with the Quicksort algorithm[14] [933, 1163] takes in average something about $n \log_2 n$ steps, this is sufficient enough, even if the correct number is $2n \ln n \approx 1.39n \log_2 n$.

The Big-**O**-family notations introduced by Bachmann [96] and made popular by Landau [1236] allow us to group functions together that rise at approximately the same speed.

Definition 30.13 (Big-O notation). The big-**O**[15] notation is a mathematical notation used to describe the asymptotical upper bound of functions.

$$f(x) \in \mathbf{O}(g(x)) \Leftrightarrow \exists\, x_0, m \in \mathbb{R} : m > 0 \wedge |f(x)| \leq m|g(x)| \,\forall\, x > x_0 \tag{30.1}$$

In other words, a function $f(x)$ is in **O** of another function $g(x)$ if and only if there exists a real number x_0 and a constant, positive factor m so that the absolute value of $f(x)$ is smaller (or equal) than m-times the absolute value of $g(x)$ for all x that are greater than x_0.

[13] http://en.wikipedia.org/wiki/Analysis_of_algorithms [accessed 2007-07-03]
[14] http://en.wikipedia.org/wiki/Quicksort [accessed 2007-07-03]
[15] http://en.wikipedia.org/wiki/Big_O_notation [accessed 2007-07-03]

Therefore, $x^3 + x^2 + x + 1 = f(x) \in \mathbf{O}(x^3)$ since for $m = 5$ and $x_0 = 2$ it holds that $5x^3 > x^3 + x^2 + x + 1 \; \forall x \geq 2$.

In terms of algorithmic complexity, we specify the amount of steps or memory an algorithm needs in dependency on the size of its inputs in the big-**O** notation. A discussion of this topic and some examples can be found in Table 30.1.

class	examples	description
$\mathbf{O}(1)$	$f_1(x) = 2^{222}$, $f_2(x) = \sin x$	Algorithms that have constant runtime for all inputs are $\mathbf{O}(1)$.
$\mathbf{O}(\log n)$	$f_3(x) = \log x$, $f_4(x) = f_4(\frac{x}{2}) + 1; f_4(x < 1) = 0$	Logarithmic complexity is often a feature of algorithms that run on binary trees or search algorithms in ordered sets. Notice that $\mathcal{O}(\log n)$ implies that only parts of the input of the algorithm is read/regarded, since the input has length n and we only perform $m \log n$ steps.
$\mathbf{O}(n)$	$f_5(x) = 23n + 4$, $f_6(x) = \frac{n}{2}$	Algorithms of $\mathbf{O}(n)$ require to access and process their input a constant number of times. This is for example the case when searching in a linked list.
$\mathbf{O}(n \log n)$	$f_7(x) = 23x + x \log 7x$	Many sorting algorithms like quicksort and mergesort are in $\mathbf{O}(n \log n)$
$\mathbf{O}(n^2)$	$f_8(x) = 34x^2$, $f_9(x) = \sum_{i=0}^{x+3} x - 2$	Some sorting algorithms like selection sort have this complexity. For many problems, $\mathbf{O}(n^2)$-solutions are acceptable good.
$\mathbf{O}(n^i) : i > 1, i \in \mathbb{R}$	$f_{10}(x) = x^5 - x^2$	The general polynomial complexity. In this group we find many algorithms that work on graphs.
$\mathbf{O}(2^n)$	$f_{11}(x) = 23 * 2^x$	Algorithms with exponential complexity perform slowly and fast become unfeasible with increasing input size. For many hard problems, there exist only algorithms of this class. Their solution can otherwise only be *approximated* by the means of randomized global optimization techniques.

Table 30.1: Some examples of the big-**O** notation

Definition 30.14 (Big-Ω notation). The big-**Ω** notation is a mathematical notation used to describe the asymptotical lower bound of functions.

$$f(x) \in \mathbf{\Omega}(g(x)) \Leftrightarrow \exists \, x_0, m \in \mathbb{R} : m > 0 \wedge |f(x)| \geq m|g(x)| \, \forall \, x > x_0 \tag{30.2}$$

$$f(x) \in \mathbf{\Omega}(g(x)) \Leftrightarrow g(x) \in \mathbf{O}(f(x)) \tag{30.3}$$

Definition 30.15 (Θ notation). The **Θ** notation is a mathematical notation used to describe both, an upper and a lower asymptotical bound of functions.

$$f(x) \in \mathbf{\Theta}(g(x)) \Leftrightarrow f(x) \in \mathbf{O}(g(x)) \wedge f(x) \in \mathbf{\Omega}(g(x)) \tag{30.4}$$

Definition 30.16 (Small-o notation). The small-**o** notation is a mathematical notation used to define that a function is asymptotical negligible compared to another one.

$$f(x) \in \mathbf{o}(g(x)) \Leftrightarrow \lim_{n \to \infty} \left| \frac{f(x)}{g(x)} \right| = 0 \tag{30.5}$$

Definition 30.17 (Small-ω notation). The small-ω notation is a mathematical notation used to define that another function is asymptotical negligible compared to a special function.

$$f(x) \in \omega(gx) \Leftrightarrow \lim_{n\to\infty} \left|\frac{f(x)}{g(x)}\right| = \infty \tag{30.6}$$

$$f(x) \in \omega(g(x)) \Leftrightarrow g(x) \in \mathbf{o}(f(x)) \tag{30.7}$$

30.1.4 Randomized Algorithms

Deterministic algorithms[16] will always produce the same results when given the same inputs. Such behavior comes closest to the original intention behind the definition of algorithms. The execution of a recipe should always yield the same meal, sorting identical lists should always result in, again identical, sorted lists. So in general, algorithms are considered to be deterministic. For many problems however, deterministic algorithms are unfeasible. In global optimization (see Section 1.1.1 on page 22), the problem space $\mathbb{X}$ is often extremely large and the relation of an element's structure and its utility as solution is not obvious. Hence, the search space $\mathbb{G}$ often cannot be partitioned wisely and an exhaustive search would be the only deterministic option left. Such an approach would take an infeasible long time. Here, the only way out is using a randomized algorithm.

Definition 30.18 (Randomized Algorithm). A randomized algorithm[17] includes at least one instruction that acts on the basis of random numbers. In other words, a randomized algorithm violates the constraint of determinism. Randomized algorithms are also often called probabilistic algorithms [1473, 965, 1438, 964, 1474].

There are two general classes of randomized algorithms: Las Vegas and Monte Carlo algorithms.

Definition 30.19 (Las Vegas Algorithm). A Las Vegas algorithm[18] is a randomized algorithm that never returns a wrong result [86, 1473, 965, 964].

Either it returns the correct result, reports a failure, or does not return at all. If a Las Vegas algorithm returns, its outcome is deterministic (but not the algorithm itself). The termination (see Definition 30.10 on page 550) however cannot be *guaranteed*. There usually exists an *expected* runtime limit for such algorithms – their actual execution however may take arbitrarily long. In summary, we can say that a Las Vegas algorithm terminates with a positive probability and is (partially) correct.

Definition 30.20 (Monte Carlo Algorithm). A Monte Carlo algorithm[19] always terminates. Its result however can be correct or incorrect [1473, 965, 964]. In contrast to Las Vegas algorithms, Monte Carlo algorithms always terminate but are (partially) correctly only with a positive probability.

Definition 30.21 (Monte Carlo Method). Monte Carlo methods[20] are a class of Monte Carlo algorithms used for simulating the behavior of systems of different types. Therefore, Monte Carlo methods are nondeterministic and often incorporate random numbers [845, 1339, 1744, 1294].

[16] http://en.wikipedia.org/wiki/Deterministic_computation [accessed 2007-07-03], see also Definition 30.11 on page 550

[17] http://en.wikipedia.org/wiki/Randomized_algorithm [accessed 2007-07-03]

[18] http://en.wikipedia.org/wiki/Las_Vegas_algorithm [accessed 2007-07-03]

[19] http://en.wikipedia.org/wiki/Monte_carlo_algorithm [accessed 2007-07-03]

[20] http://en.wikipedia.org/wiki/Monte_Carlo_method [accessed 2007-07-03]

30.2 Distributed Systems and Distributed Algorithms

Various definitions have been issued for the terms *distributed system* and *distributed algorithms* by several researchers such as Bal [122], Lamport [1235], Tanenbaum and van Steen [2006], Mattern [1370], Tel [2010], Barbosa [146], Coulouris et al. [457], Ghosh [799], and Mühl [1475]. These definitions most often only differ in minor details and can be summarized as follows.

Definition 30.22 (Distributed System). A distributed system is a set of autonomous systems (nodes) which are connected by a network and communicate via the exchange of messages. [1475]

Definition 30.23 (Distributed Algorithm). Distributed algorithms [1370, 2010, 146] are algorithms which are executed by multiple computers in a distributed system and cooperatively try to solve a given problem.

Distributed algorithms can be distinguished from sequential algorithms because they run on multiple nodes in parallel in order to cooperatively solve one problem. They can be distinguished from mere parallel algorithms since each node in the distributed system executes instances of the same algorithm with a (usually) different view on the global state [2010, 122, 2006].

The reason for this lack of a common view on the global state is that each node has only knowledge about the information locally available on it. Information on the other nodes can only be obtained via communication which usually comprises the exchange of messages.

Latency is the time difference between the moment where something is initiated and the moment when its effects becoming observable [457]. Communication usually involves latency. Whenever a process sends a message, its contents are handed down to the operating system or a middleware. From this moment on, the process considers the message as *sent*. The operating system now must initialize the communication, prepare the message for the transmission medium, and send it to its destination(s).

The laws of physics induce an additional delay, preventing the message from instantaneously occurring at its target. This delay is normally negligible. Yet it is observable in satellite communication, for example when the host of a news show talks with a reporter on the other side of the globe.

Once a message arrives at the destination node, it is reassembled from the medium and the operating system or middleware passes its contents to the receiving process. From the moment on where the execution of this process is resumed, the message is considered as *received*.

Of course, with technical effort such as special clocking, latency could be made transparent for the system. In general computer networks (let alone MANETs or sensor networks) this is not possible and messages are always delayed. Because of this latency, the nodes cannot have exactly the same view on the world and it is not possible to have an exact, globally synchronized system time available. Furthermore, networks may induce arbitrary errors into the message's content and messages can even get lost, i. e., have an infinite latency.

Distributed algorithms can provide the following advantages (depending on their design): modularity, flexibility, resource-sharing, no central point of failure, scalability because of decentralization, robustness, high availability, and fault-tolerance. In turn, they may have the following drawbacks, again depending on their design: higher complexity, no common view on the global state, no global time, processes may fail, latency and faults in communication, problems in termination detection, deadlocks, and race conditions.

Whether a distributed algorithm is adequate or not depends on the degree to which it exploits the advantages of the distribution and how strongly the mentioned drawbacks are present in its design. The quality of an *adequate* distributed algorithm can be determined by its functionality, its communications complexity, i. e., how many messages need to be

exchanged in order to solve its task, or its time complexity, i. e., how many computational steps need to be performed on the single nodes.

TODO

Definition 30.24 (Scalability). Scalability[21] is a measure describing how good a system can grow or be extended for processing a higher computational load.

Definition 30.25 (Central Point Of Failure). A central (or single) point of failure is a subsystem or process that, if it fails, leads to the collapse of the whole distributed system. An example for central point of failures is central servers.

Definition 30.26 (Bottleneck). The bottleneck[22] of a distributed application is the part that has the most limiting influence on its performance.

Imagine, for instance, an hourglass. Here, the dilution in its center is the *bottleneck* that limits the amount of sand that can fall down per time unit.

TODO

30.2.1 Network Topologies

Definition 30.27 (Network Topology). Network topology[23] is the study of arrangement of the components of a network such as connections and nodes. The network layout itself can also be referred to as *topology*.

In the further text, we will use the term *edge* synonymously for link and connection and the term *vertex* as synonym for node or computer since network topology is closely related to graph theory.

Each computer network has exactly one physical topology which is the layout of its physical components (computers, cables). This physical structure defines which nodes can communicate directly with each other and which not. On top of that physical design, several virtual/overlay topologies may be built.

Definition 30.28 (Overlay Network). An overlay network[24] is a virtual network which is built on top of another computer network. The nodes in the overlay network are connected by virtual or logical links [50].

IP addresses[25], for instance, form an overlay topology on top of MAC addresses[26] in Ethernets[27]. A peer-to-peer network is an overlay network because it runs on top of the internet. Several distributed algorithms require the nodes to be arranged in special topologies like stars or rings. This can be achieved in arbitrary networks by defining an overlay structure which performs according routing and address translations.

When speaking of topology, one would normally think about a hardwired network of computers, connected with each other through Ethernet cabling and such and such. If we consider a WLAN[28] or a wireless sensor network as described in Definition 30.32 on page 559

[21] http://en.wikipedia.org/wiki/Scalability [accessed 2008-02-08]
[22] http://en.wikipedia.org/wiki/Bottleneck [accessed 2007-07-03]
[23] http://en.wikipedia.org/wiki/Network_topology [accessed 2007-07-03]
[24] http://en.wikipedia.org/wiki/Overlay_network [accessed 2007-07-03]
[25] http://en.wikipedia.org/wiki/IP_address [accessed 2008-02-09]
[26] http://en.wikipedia.org/wiki/Mac_address [accessed 2008-02-09]
[27] http://en.wikipedia.org/wiki/Ethernet [accessed 2008-02-09]
[28] http://en.wikipedia.org/wiki/Wireless_LAN [accessed 2008-02-08]

on the other hand, there is of course no such thing as cabling. But still, there is a certain topology: not all nodes may be able to directly contact each other since their radio transmission ranges are limited. They may only be able to exchange messages directly with some nodes in their physical neighborhood only. Hence, we can span a graph over this network, where each node is connected to his neighbors in communication range only. This graph then defines the topology.

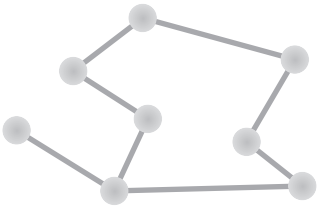


Fig. 30.3.a: unrestricted topology

Fig. 30.3.b: bus

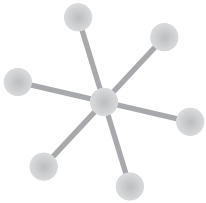


Fig. 30.3.c: star

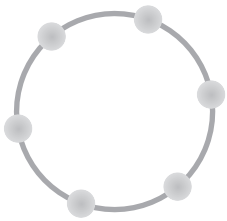


Fig. 30.3.d: ring

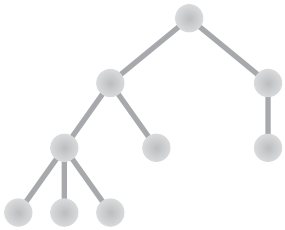


Fig. 30.3.e: hierarchy

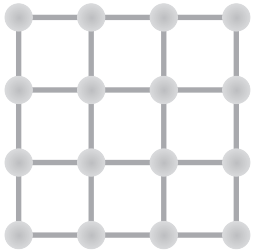


Fig. 30.3.f: grid

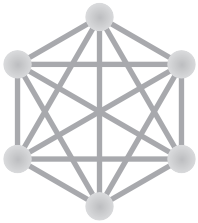


Fig. 30.3.g: fully connected

Figure 30.3: Some simple network topologies

Unrestricted

In an unrestricted network topology as the one sketched in Fig. 30.3.a, we make only the general assumption that there is no network partition. In other words, for all nodes n in the network, there exists at least one path to each other node in the network. This path may, of course, consist of multiple hops over multiple connections.

Bus

All nodes in a bus system (illustrated in Fig. 30.3.b) are connected to the same transmission medium in a linear arrangement. All messages sent over the medium can be considered to

be broadcasts that potentially can be received by all nodes more or less simultaneously. The transmission medium has exactly two ends.

Star

Fig. 30.3.c shows an example for a star topology. Here, all nodes are connected to a single node in the center of the network. This center could, for example, be an Ethernet hub[29] or switch[30] that retransmits the messages received to their correct destination. It could as well be a server that performs some specific tasks for the notes. For a detailed discussion of the client-server architecture see Section 30.2.2.

Ring

In this topology, each node is connected to exactly two other nodes in a way that no partition exists. The nodes are arranged in a sequence where the first and the last node are connected with each other. An instance of the ring topology is illustrated in Fig. 30.3.d.

Hierarchy

Fig. 30.3.e illustrates a hierarchical topology where the nodes of the network are arranged in form of a tree.

Grid

The nodes in a grid are laid out in a two-dimensional lattice so that each node (except those at the borders of the grid) is connected with four neighbors: one to the left, one to the right, one above and one below. Fig. 30.3.f is an instance of such a topology.

Fully Connected

In a fully connected network, as sketched in Fig. 30.3.g, each node is directly connected with each other node.

30.2.2 Some Architectures of Distributes Systems

Client-Server Systems

Definition 30.29 (Client-Server). Client-server[31] is a network architecture that separates two types of nodes: the client(s) and the server(s) [2321, 72]. A client[32] utilizes a service provided by a server[33]. It does so by sending a request to the server. This request contains details of the task to be carried out, for example the URL of a website to be returned. The server then executes appropriate actions and, in most cases, sends a response to the client. Usually, there is a small number of servers (normally one) which servers many clients.

Client-server architectures like the one illustrated in Figure 30.4 are the most basic and the most common application logical architecture in distributed computing [457, 1370, 2006]. They are part of almost all internet applications like:

[29] http://en.wikipedia.org/wiki/Ethernet_hub [accessed 2007-07-03]
[30] http://en.wikipedia.org/wiki/Ethernet_switch [accessed 2007-07-03]
[31] http://en.wikipedia.org/wiki/Client_server [accessed 2007-07-03]
[32] http://en.wikipedia.org/wiki/Client_%28computing%29 [accessed 2007-07-03]
[33] http://en.wikipedia.org/wiki/Server_%28computing%29 [accessed 2007-07-03]

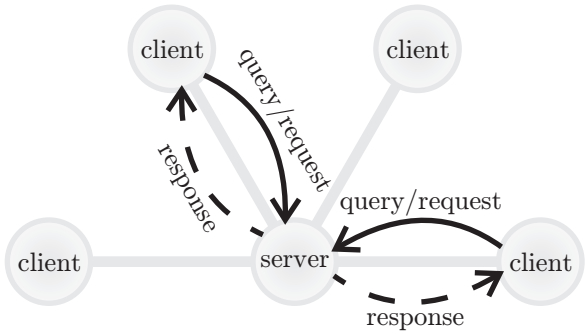


Figure 30.4: Multiple clients connected with one server

1. Websites[34] in the world wide web[35] are obtained by using the HTTP[36] protocol for communication between a web browser[37] and a web server[38].
2. Application servers[39] contain the business logic of corporations. They support online shops[40] with an underlying business model, for example.
3. Database servers[41] provide computers in a network with access to large data sets. Furthermore, they allow their clients to send structured queries that allow aggregation and selection of specific data.
4. ...

The major advantages of client-server systems are their simplicity. Local algorithms can often be integrated into servers without too many problems while their adaptation to more complicated architectures is, well, more difficult and error-prone. The heaviest weakness of the client-server scheme is that the server represents a bottleneck (see Definition 30.25) and a single point of failure (see Definition 30.25 on page 554).

Peer-To-Peer Networks

Definition 30.30 (Peer-To-Peer Network). Instead of being composed of client and server nodes, a peer-to-peer[42] network consists only of equal peer nodes. A peer node works as a server for its fellow peers by providing certain functionality and simultaneous acts as client utilizing an similar service from its peers [457, 1370, 2006, 1959, 39]. Therefore, a peer node is often also called *servent*[43], a combination of the words server and client. The expression peer-to-peer is often abbreviated by P2P.

Peer-to-peer networks may have an arbitrary structure like the one sketched in Figure 30.5. While client-server systems are limited to providing communication between the clients and the server solely, peer-to-peer networks may resemble any sort of underlying communication graph.

Peer-to-peer architectures circumvent the existence of single points of failure and can be constructed to be very robust against bottlenecks. They furthermore are often ad hoc, i. e.,

[34] http://en.wikipedia.org/wiki/Website [accessed 2007-07-03]
[35] http://en.wikipedia.org/wiki/Www [accessed 2007-07-03]
[36] http://en.wikipedia.org/wiki/Http [accessed 2007-07-03]
[37] http://en.wikipedia.org/wiki/Web_browser [accessed 2007-07-03]
[38] http://en.wikipedia.org/wiki/Web_server [accessed 2007-07-03]
[39] http://en.wikipedia.org/wiki/Application_server [accessed 2007-07-03]
[40] http://en.wikipedia.org/wiki/Online_shop [accessed 2007-07-03]
[41] http://en.wikipedia.org/wiki/Database_server [accessed 2007-07-03]
[42] http://en.wikipedia.org/wiki/Peer-to-peer [accessed 2007-07-03]
[43] http://en.wikipedia.org/wiki/Servent [accessed 2007-07-03]

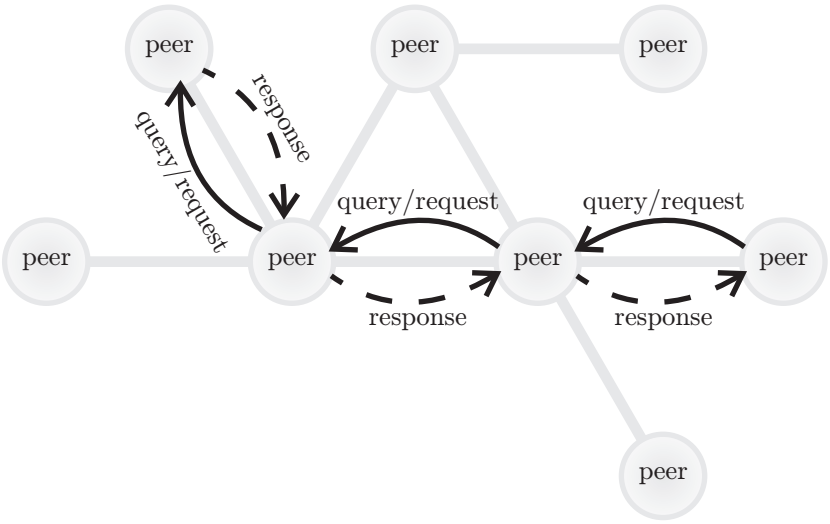


Figure 30.5: A peer-to-peer system in an unstructured network

new peers may join the network at any time and leave it whenever they decide to do so. This can also be regarded as a drawback since the structure (and thus, its computational power and connectivity) of network may fluctuate heavily as well as the availability of data provided by the peers.

If obeying the definition exactly, there are no centralized components in a peer-to-peer network. There however exist hybrid networks where the peers for example register at a dedicated server which keeps track on the users online. Also, there exist different hierarchical or non-hierarchical overlay networks.

Important peer-to-peer-based applications are

1. File and content sharing systems [60, 1807] are the most influential and wide-spread P2P systems. Millions of users today share music, videos, documents and software over networks like Gnutella[44] [420, 1740], Bittorrent[45], appleJuice[46] and the famous but shut-down Napster[47] network.
2. Many scientific applications like Seti@home[48], Einstein@home[49], and Folding@home[50] rely on users all over the world that voluntarily provide their unused computational resources. Such applications are most often constructed as screensavers that, after becoming active, download some pieces of data from a server and perform computations on them. After finishing the work on the received data, a response is issued to the server.
3. Many instant messaging[51] systems like talk[52] utilize peer-to-peer protocols . Most often, the clients need to log on and send status information to a server. Communication then either works client-server-based or in P2P-manner. Especially when audio or video chats come into play, peer-to-peer approaches are usually preferred.
4. ...

[44] http://en.wikipedia.org/wiki/Gnutella [accessed 2007-07-03]
[45] http://en.wikipedia.org/wiki/BitTorrent [accessed 2007-07-03]
[46] http://www.applejuicenet.de/ [accessed 2007-07-03]
[47] http://en.wikipedia.org/wiki/Napster [accessed 2007-07-03]
[48] http://en.wikipedia.org/wiki/Seti_at_home [accessed 2007-07-03]
[49] http://en.wikipedia.org/wiki/Einstein%40Home [accessed 2007-07-03]
[50] http://en.wikipedia.org/wiki/Folding%40home [accessed 2007-07-03]
[51] http://en.wikipedia.org/wiki/Instant_messaging [accessed 2007-07-03]
[52] http://en.wikipedia.org/wiki/Talk_%28Unix%29 [accessed 2007-07-03]

Sensor Networks

Definition 30.31 (Sensor Network). A sensor network[53] [1012, 1966, 469, 1025] is a network of autonomous devices which are equipped with sensors and together measure physical entities like temperature, sound, vibrations, pressure, motion, or such and such.

Definition 30.32 (Wireless Sensor Network). A wireless sensor network (WSN) [1697, 326, 2317, 1092, 1873] is a sensor network where the single nodes are connected wirelessly, using techniques like wireless LAN[54], Bluetooth[55], or radio[56].

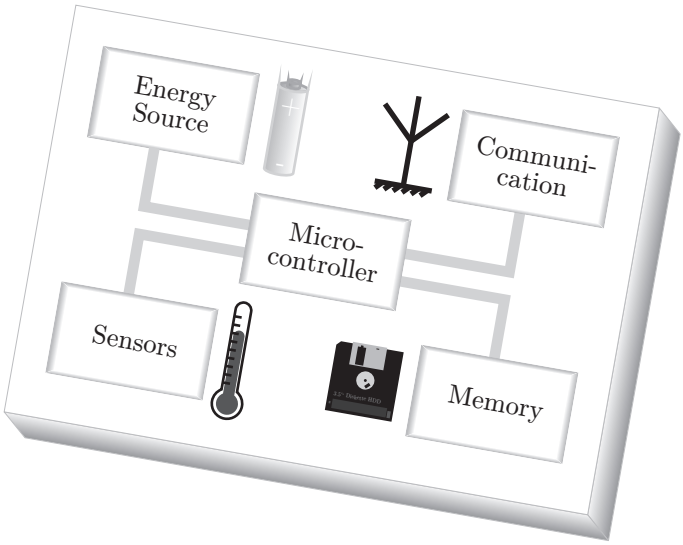


Figure 30.6: A block diagram outlining building blocks of a sensor node.

Figure 30.6 sketches the building blocks of a sensor node. For communication with other nodes, short range radios, Bluetooth, or wireless LAN adapters are often added. The core of a sensor node is a microcontroller attached with RAM and ROM memory for code and data. The purpose of senor networks is to measure some environmental parameters like temperature, humidity, or brightness. Thus, sensor nodes have one or multiple sensors attached.

Since they are autonomous devices usually not connected to power lines, sensor nodes have to be equipped with some sort of energy source. Chemical batteries are used to store energy, but often power scavenging units [574, 1769, 1610, 1698] like, for example, solar cells [1881, 1050, 1537], thermal [1988] or kinetic energy harvesters [2149, 1609, 1229] are added. The field of energy supply of sensor nodes is critical and subject to active research [1843, 688, 382, 767]. Batteries have limited capacity and are hard to replace after the network has been deployed. If no additional power scavenging unit is available, the sensor nodes will eventually stop functioning and become useless after all their energy is consumed. For extending this lifetime, energy intense operations like communication via radio transmissions need to be reduced as much as possible.

The size of the sensor nodes ranges from shoe box down to matchbox dimensions. There is a strong wish to produce smaller and smaller nodes. Small sensors are recognized less

[53] http://en.wikipedia.org/wiki/Sensor_network [accessed 2007-07-03]
[54] http://en.wikipedia.org/wiki/Wireless_lan [accessed 2007-07-03]
[55] http://en.wikipedia.org/wiki/Bluetooth [accessed 2007-07-03]
[56] http://en.wikipedia.org/wiki/Radio [accessed 2007-07-03]

obviously and blend better in their environment. Since they require fewer raw material, they might become much cheaper than their larger pendants. On the other hand, with this movement in the direction of sensor that are really tiny, some hard constraints arise. The size of the battery limits the amount of energy that can be stored, as well as the extent of a solar cell limits its energy production. The node size also restricts the dimensions of the memory and the sensors of the node [397].

Other important research topics are data fusion and transportation in a WSN [611, 846] as deployment and maintenance [2009, 957].

Widespread sensor node architectures are:

1. *BTNodes*[57] are autonomous wireless communication and computing platforms based on a Bluetooth radio and a microcontroller. Developed at the ETH Zurich, BTNodes serve especially as demonstration, teaching, and research platforms. Fig. 30.7.a shows a BTNode.
2. Crossbow's *MICA2*[58] motes are multipurpose nodes. These systems are applied widely in real-world applications like environmental control in agriculture and outdoor sports as well as for indoor sports and military purposes. A picture of the Mica2Dot platform can be found in Fig. 30.7.b.
3. Scatterweb[59] provide both, a research platform (*MSB* nodes, illustrated in Fig. 30.7.c) and an industrial sensor network (*ScatterNode*s).
4. Dust Networks[60] developed their *SmartMesh* for building wireless solutions for the global market. Their nodes use the Time Synchronized Mesh Protocol and middle-range radio to provide the reliability of a typical WLAN in their sensor networks. Fig. 30.7.d shows a Dust Networks Evaluation Mote.
5. ...

A small example application demonstrating the use of sensor networks is discussed in Section 24.1.2 on page 414.

Properties of Peer-To-Peer Systems and Sensor Networks

1. Current peer-to-peer networks are often large-scale, with tens of thousands [1807] up to millions [2262] of users/nodes online. Although networks of thousands of sensors are a future goal, the number of nodes in sensor networks has not yet reached this extent. However, systems of several hundreds of nodes have already been deployed [468, 1990].
2. Since wireless sensor networks have limited transmission range, it is possible that not all nodes in a network can communicate directly with each other. The same issue exists in the internet, but there it is solved in a transparent manner by routers. In sensor networks however, dedicated hardware routers normally do not exist. Therefore, special routing protocols [1978, 1387, 386] are applied. Here we see a strong relation between sensor networks and peer-to-peer systems: Each sensor may act as a sender of messages as well as router for other nodes. There is no generic hierarchy or division between senders or routers.
3. Especially in peer-to-peer applications, there are strong fluctuations in the network membership. In content sharing networks for example, new users continuously join and leave the network. In sensor networks on the other hand, volatility in the network structure arises from newly deployed nodes or nodes that become inactive because they ran out of battery power. A sensor node spends much of its time in sleep mode (so do I) and may be regarded as inactive in this time. When it triggers back to active mode, it again

[57] http://www.btnode.ethz.ch/ [accessed 2007-07-03]

[58] http://www.xbow.com/Products/productdetails.aspx?sid=156 [accessed 2007-07-03]

[59] http://www.inf.fu-berlin.de/inst/ag-tech/scatterweb_net/ [accessed 2007-07-03] and http://www.scatterweb.com/ [accessed 2007-07-03]

[60] http://www.dustnetworks.com/ [accessed 2007-07-03]

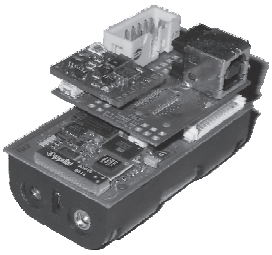


Fig. 30.7.a: BTNode

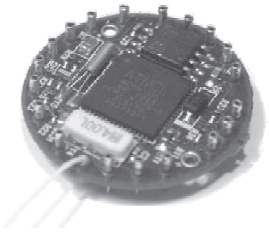


Fig. 30.7.b: Mica2Dot

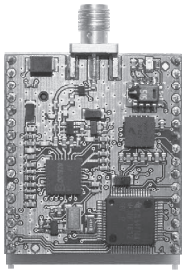


Fig. 30.7.c: MSB Mote

Fig. 30.7.d: Dust Networks Evaluation Mote

Figure 30.7: Images of some sensor network platforms

becomes member of the network. Furthermore, networks of mobile sensors have large fluctuations in their topology per default.

4. Since sensor networks utilize sleep cycles in order to reduce energy consumption, messages that are routed may arbitrarily be delayed or even get lost.
5. P2P networks often represent very heterogeneous environments, consisting of computers of different architectures and operating systems. Sensor networks on the other hand are most often homogeneous systems.

30.3 Grammars and Languages

Languages are the most important means for communication between higher animals [61]. Formal languages can also be used define the formats for data being stored by or exchanged between computers and/or human beings. When analyzing a statement in a given language, we distinguish between its syntax and semantic.

Definition 30.33 (Syntax). The syntax[62] of a language is the set of rules that governs its *structure*. Each valid statement of a language must obey its syntactical structure. The sentence *"I am reading a book."* is a sequence of a subject, a predicate, and an object.

Definition 30.34 (Semantic). The semantic[63] refers to the *meaning* of a statement. The sentence *"I am reading a book."* has the meaning that the writer of it is visually obtaining information from a set of bounded pages filled with written words.

[61] http://en.wikipedia.org/wiki/Language [accessed 2007-07-04]
[62] http://en.wikipedia.org/wiki/Syntax [accessed 2007-07-03]
[63] http://en.wikipedia.org/wiki/Semantics [accessed 2007-07-03]

30.3.1 Syntax and Formal Languages

Let us now take a closer look on the syntax of formal languages [381, 1166].

Definition 30.35 (Alphabet). A finite set Σ of symbols (characters) $\alpha \in \Sigma$ with a total order (see Section 27.7.2 on page 463) defined on it is called an alphabet.

Definition 30.36 (Character String). A character string[64] (or word) over an alphabet Σ is any finite sequence of symbols $\alpha \in \Sigma$. Character strings have the following properties:

1. The empty character string ε is a character string over Σ.
2. If x is a character string over Σ, then αx is also a character string over Σ for all $\alpha \in \Sigma$.
3. β is a character string over the alphabet Σ if and only if it can be created using the two rules above.

Definition 30.37 (Concatenation). The concatenation[65] $\alpha \circ \beta$ of two character strings $\alpha = \alpha_1\alpha_2\alpha_3 \ldots \alpha_n$ and $\beta = \beta_1\beta_2\beta_3 \ldots \beta_m$ over the alphabet Σ is the character string $\alpha \circ \beta = \alpha_1\alpha_2\alpha_3..\alpha_n\beta_1\beta_2\beta_3..\beta_m$ which begins with α immediately followed (and ended by) β.

The set of all strings of length l over the alphabet Σ is called Σ^l with $\Sigma^0 = \{\varepsilon\}\ \forall \Sigma$. The set of all strings on Σ is called Σ^*, i. e., $\Sigma^* = \cup_{l=0}^{\infty} \Sigma^l$. It is also called Kleene star[66] (or Kleene closure).

Definition 30.38 (Lexeme). A lexeme[67] is the lowest level of syntactical unit of a language [381]. It denotes a set of words that have the same meaning, like *run, runs, ran, and running* in English. A lexeme belongs to a particular syntactical category and has a semantic meaning.

Based on these definitions, we can consider a sentence to be a sequence of lexemes which, in turn, are character strings over some alphabet.

Definition 30.39 (Language). A language $\mathbb{L}$ over the alphabet Σ is a subset of Σ^* [1166]. $\mathbb{L}$ is the set of all sentences over an alphabet Σ that are valid according to its rules in syntax (the grammar) [395].

When describing the formal syntax of a language, there are two possible approaches:

1. We can define recognizers that determine the structure of a sentence and can decide whether it belongs to the language or not. Recognizers are, for instance, used in compilers [865].
2. A generative grammar can be defined from which all possible sentences of a language can be constructed.

30.3.2 Generative Grammars

A generative grammar G of a language $\mathbb{L}$ is a formal specification that allows us to construct every single sentence in $\mathbb{L}$ by applying recursive replacement rules. Therefore, we define non-terminal symbols (also called variables) which do not occur in the language's text and terminal symbols that do. One example of such a grammar is:

```
1 sentence ⟶ subject verb object
2 subject  ⟶ Alice ∨ Bob
3 verb     ⟶ writes ∨ reads
4 object   ⟶ cipher-text ∨ plain text
```

Listing 30.1: A simple generative grammar.

[64] http://en.wikipedia.org/wiki/Character_string [accessed 2007-07-03]
[65] http://en.wikipedia.org/wiki/Concatenation [accessed 2007-07-10]
[66] http://en.wikipedia.org/wiki/Kleene_star [accessed 2007-07-03]
[67] http://en.wikipedia.org/wiki/Lexeme [accessed 2007-07-03]

Here we have four productions, the terminal symbols `Alice`, `Bob`, `writes`, `reads`, `cipher-text`, `plain-text`, plus five non-terminal symbols (`sentence`, `subject`, `verb`, and `object`).

Definition 30.40 (Formal Grammar). A formal grammar[68] $G = (N, \Sigma, P, S)$ is a 4-tuple consisting of:

1. a finite set N of non-terminal symbols (variables),
2. the alphabet Σ, a finite set of terminal symbols,
3. a finite set P of productions (also called rules), and
4. at least one start symbol $S \in N$ which belongs to the set of non-terminal symbols N.

Notice that the alphabet Σ here is not limited to letters or numerals, but may contain words, sentences, or even arbitrarily long texts. Additionally, we call the set $V = N \cup \Sigma$ including terminal and non-terminal symbols the grammar symbols.

The Chomsky Hierarchy

The Chomsky hierarchy stands for a hierarchy of formal grammars that generate a formal language. It was first described by the linguist Chomsky [394] in 1956 [394, 396, 1175] and distinguishes four different classes of grammars. Starting with an unbounded grammar (type-0), more and more restrictions are imposed on the allowed production rules. Hence, each type contains all grammar types on higher levels fully.

Grammar	Allowed Rules	Languages
Type-0	$\alpha \to \beta$,$\alpha, \beta \in V^*, \alpha \neq \varepsilon$	recursive enumerable
Type-1	$\alpha A \beta \to \alpha \gamma \beta$, $A \in N$, $\alpha, \beta, \gamma \in V^*$, $\gamma \neq \varepsilon$	context-sensitive (CSG)
Type-2	$A \to \gamma$, $A \in N$, $\gamma \in V^*$	context-free ()
Type-3	$A \to aB$ (right-regular) or $A \to Ba$ (left-regular), $A \to a$, $A, B \in N$, $a \in \Sigma$	regular

Table 30.2: The Chomsky Hierarchy

Table 30.2 illustrates the Chomsky hierarchy. As already mentioned, V is the set containing all terminal and non-terminal symbols and V^* is its Kleene closure.

30.3.3 Derivation Trees

A derivation tree[69] is a common way to describe how a sentence in a context-free language can be derived from the start symbol of a given generative grammar. The inner nodes of a derivation tree are the non-terminal symbols in N, the root is the start symbol S, and the leaves are the terminal symbols from the alphabet Σ. Each edge constitutes one expansion according to a production of the grammar.

Assume an example grammar $G = (N, \Sigma, P, S)$ with $N = \{\texttt{T}\}$, $\Sigma = \{\texttt{1}, +, \texttt{a}\}$, $S = \texttt{T}$, and the productions P as defined the below.

```
1 T ⟶ T+T
2 T ⟶ 1
3 T ⟶ a
```

Listing 30.2: An example context-free generative grammar G.

[68] http://en.wikipedia.org/wiki/Formal_grammar [accessed 2007-07-03]
[69] http://en.wikipedia.org/wiki/Context-free_grammar#Derivations_and_syntax_trees [accessed 2007-07-16]

With this grammar we can construct the following sentence:

```
1 T      ⟶ T+T
2 T+T    ⟶ T+T+T
3 T+T+T  ⟶ a+T+T
4 a+T+T  ⟶ a+1+T
5 a+1+T  ⟶ a+1+a
```

Listing 30.3: An example expansion of G.

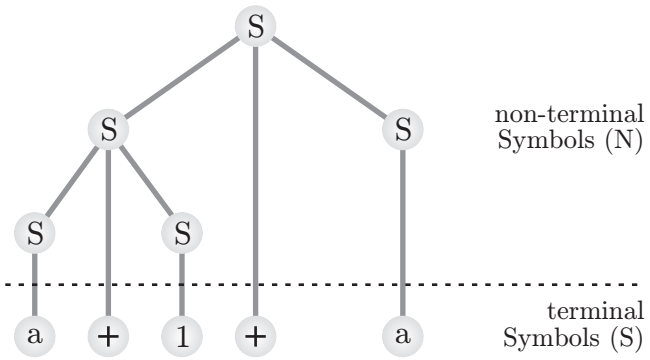


Figure 30.8: The derivation of the example expansion of the grammar G.

Figure 30.8 illustrates the derivation tree that belongs to this example expansion of the example grammar G.

30.3.4 Backus-Naur Form

The Backus-Naur (BNF) form[70] is a metasyntax used to express context-free grammars [109, 1156]. Such Chomsky Type-2 grammars are the theoretical basis of most common programming languages and data formats, like for example C and XML[71]. The BNF allows specifying production rules in simple, human and machine-understandable manner.

In BNF specifications, each rule consists of two parts: a non-terminal symbol on the left-hand side and an expansion on the right-hand side. Non-terminal symbols are contained in arrow brackets and terminal symbols are written plain. For expansions, the BNF provides two constructs: a sequence of symbols and the alternative which is denoted with a pipe character "|".

Beginning with the start symbol $S = \texttt{S}$, the example below allows us to generate arbitrary natural numbers from $\mathbb{N}$. A `nonZero` is either `1`,`2`,.., or `9` and a normal `number` may also be zero. A natural number is either a `nonZero` number or a natural number with a `number` at the end. Notice that expanding `nonZero` will always lead to the first digit being a non-zero digit since a fully expanded rule cannot contain any variables (non-terminal symbols). As start symbol $S = \texttt{S}$, we use `natural`.

```
1 <nonZero> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
2 <number>  ::= 0 | <nonZero>
3 <natural> ::= <nonZero> | <natural> <number>
4 <S>       ::= <natural>
```

Listing 30.4: Natural numbers – a small BNF example.

[70] http://en.wikipedia.org/wiki/Backus%E2%80%93Naur_form [accessed 2007-07-03]
[71] http://www.w3.org/TR/2006/REC-xml-20060816/ [accessed 2007-07-03]

30.3.5 Extended Backus-Naur Form

The extended Backus-Naur form[72] is an extension of the BNF metasyntax that provides additional operators and simplifications [722, 1623, 1166]. Unlike the Backus-Naur form, the terminal symbols are included in quotation marks and the non-terminal symbols are written without arrow brackets. The items of sequences *can* now be separated by commas and each rule ends with a semicolon. The EBNF adds options, which are denoted by square brackets. The sequence inside such options may either occur zero or one time in the expanded rule. Curly brackets define expressions that can be left away or repeated arbitrary often during expansion.

The example below demonstrates the application of these new features by providing a grammar for natural numbers equal to the one shown for the BNF. The rules `natural` and `natural2` are equivalent. Here we also specify a rule for all integer numbers from $\mathbb{Z}$ by prefixing a natural number with an optional `-`.

```
1 nonZero   ::= "1" | "2" | "3" | "4" | "5" | "6" | "7" |
2               "8" | "9" ;
3 number    ::= "0" | nonZero ;
4 natural   ::= nonZero | natural, number ;
5 natural2  ::= nonZero | nonZero, {number} ;
6 integer   ::= ["-"], natural | "0" ;
7 S         ::= integer ;
```

Listing 30.5: Integer numbers – a small EBNF example.

The ISO norm ISO/IEC 14977 [722] for EBNF defines additional extension mechanisms which we will not discuss here.

30.3.6 Attribute Grammars

An attribute grammar[73] (AG) is a context-free grammar enriched with attributes, rules, and conditions for these attributes [1157, 1158, 1160, 1596]. With attributes attached to non-terminal symbols, it becomes possible to provide context-sensitive information. Attribute grammars are often used in compilers to check rules that cannot be validated with the means of mere context-free grammars. With attribute grammars, syntax trees can be translated directly into intermediate languages or into code for some specific machine.

An attribute grammar $AG = (G, A, R)$ consists of three components:

1. a context-free grammar G, where $G = (N, \Sigma, P, S)$ as specified in Definition 30.40 on page 563,
2. a finite set of attributes A where each attribute $a \in A$ has a set of possible values $a = \{a_1, a_2, .., a_n\}$, and
3. a set of semantic rules R.

To each grammar symbol $X \in V$, a finite set of attributes $A(X) \subseteq A$ is associated. This set is partitioned into two disjoint subsets, the inherited attributes $I(X) \subseteq A(X)$ and the synthesized attributes $T(X) \subseteq A(X)$. The value of a synthesized attribute is determined by the attributes attached to the children of the symbol in the derivation tree it is assigned to. Inherited attributes get their value from the parent or siblings of the symbols they belong to. In the original definition by Knuth [1157], this was the other way round but the form discussed here has prevailed [1596]. The start symbol $S \in N$ and the terminal symbols in Σ do not have inherited attributes ($I(S) = \emptyset$, $\forall \sigma \in \Sigma \Rightarrow I(\sigma) = \emptyset$).

A good example for synthesized attributes is given in [21] from where I will borrow. AGs are most often not used as generative grammars but as guidelines for parsers that read for instance source code of a programming language.

[72] http://en.wikipedia.org/wiki/Ebnf [accessed 2007-07-03]
[73] http://en.wikipedia.org/wiki/Attribute_grammar [accessed 2007-07-03]

Let us consider a simple grammar for integer mathematics with the two expressions `+` and `*`.

```
E ::= F        "+" E |
      F
F ::= integer "*" F |
      integer
```

Listing 30.6: A simple context-free grammar.

For each symbol X in V let $X.val$ be the numeric value associated with it. For terminal symbols of the type `integer`, this is simply the lexeme provided by the lexical analyzer. The two other terminal characters `+` and `*` have no value assigned. The values of the non-terminal symbols `E` and `F` should be the results of the expressions defined by them. These attributes are computed (synthesized) by the semantic rules from the attributes of their child nodes.

```
Production                Rule
E ::= F        "+" E |    E.val = F.val        + E2.val
      F                   E.val = F.val
F ::= integer "*" F |     F.val = integer.val * F2.val
      integer             F.val = integer.val
```

Listing 30.7: A small example for attribute grammars.

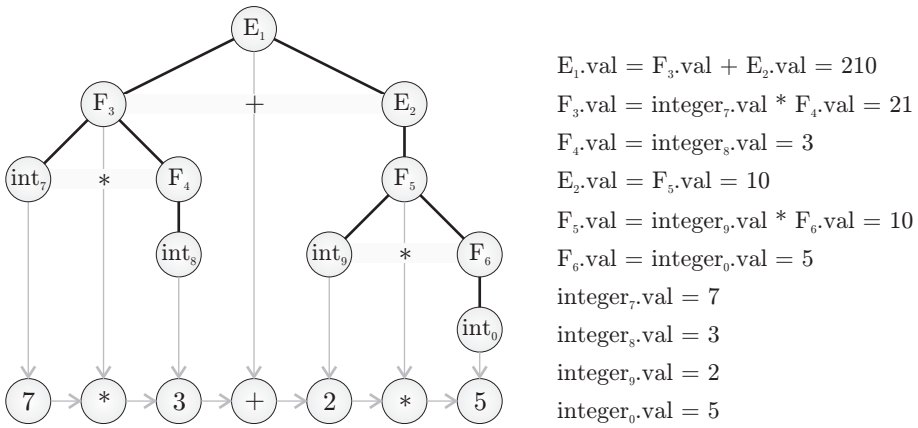


Figure 30.9: An instantiation of the grammar from Listing 30.7.

Figure 30.9 illustrates the derivation tree of a sentence of the simple attribute grammar from Listing 30.7. The non-terminal symbols are sometimes annotated with subscript numbers (like E_2) which have no meaning and only serve for clarity. While this Listing 30.7 is an example for the usage of synthesized attributes, symbol tables used in compilers are instances of inherited attributes.

A special form of attribute grammars, the reflective attribute grammar, is the basis of the Gads 2 Genetic Programming system discussed in Section 4.5.7 on page 185.

L-Attributed Grammars

L-attributed grammars[74] are a class of attribute grammars that can be parsed in one left-to-right traversal of the abstract syntax tree (see Section 4.1.1 on page 158). Such grammars are the foundations for many programming languages and allow convenient top-down parsing[75].

[74] http://en.wikipedia.org/wiki/L-attributed_grammar [accessed 2007-07-04]
[75] http://en.wikipedia.org/wiki/Top-down_parsing [accessed 2007-07-04]

S-Attributed Grammars

An attribute grammar is called S-attributed[76] if it allows only synthesized attributes [452]. Because of this restriction, such grammars can be parsed top-down as well as directly bottom-up[77] and are supported by various tools like Bison[78] and Flex[79].

30.3.7 Extended Attribute Grammars

Extended Attribute Grammars developed by Watt [2162] and Madsen [1344] (EAGs) are a form of attribute grammars where the semantic (attribute-concerning) rules are no longer separated from the syntax productions [1874]. Instead, both are combined in a declarative form where each non-terminal symbol is accompanied by its attributes listed in a predetermined order. The new syntax for non-terminal symbols is

```
1 <n ↕a ↕b ↕c ...>
```

Listing 30.8: Syntax of an Extended Attribute Grammar symbol.

While $\texttt{n} \in N$ is a non-terminal symbol and `a`, `b`, and `c` are values of attributes α, β, and γ defined as expressions over their respective attribute value domain. In an extended attribute grammar, we can define a set of inherited attributes $I(\texttt{n})$ and a set of synthesized attributes $T(\texttt{n})$ for each non-terminal symbol `n`. In our example Listing 30.8, ↕ therefore has to be replaced with either ↓ which means that the following attribute is inherited ($\downarrow a \Leftrightarrow \alpha \in I(\texttt{n})$) or ↑ denoting a synthesized attribute ($\uparrow a \Leftrightarrow \alpha \in T(\texttt{n.parent})$) where `n.parent` stands for the parent node of `n` in the derivation tree. Terminal symbols cannot have attributes. Again, notice that the identifiers `a`, `b`, and `c` do not denote the attribute names but expressions that define their values. Attributes in EAGs are solely identified by their position in the non-terminal symbol specifications.

How this approach works is best understood when again, using a simple example borrowed from [1874]. Assume the grammar $G_1 = (N, \Sigma, P, S)$ with the non-terminal symbols $N = \{\texttt{S}, \texttt{X}, \texttt{Y}, \texttt{Z}\}$, the alphabet $\Sigma = \{\texttt{x}, \texttt{y}, \texttt{z}, \varepsilon\}$, productions P as defined below and the start symbol $S = \texttt{S}$. Additionally, `X`, `Y`, and `Z` are equipped with one synthesized attribute $\texttt{v} \in \mathbb{N}_0$.

```
1 <S>       ::= <X ↑v><Y ↑v><Z ↑v>
2 <X ↑v+1> ::= <X ↑v>"x"
3 <Y ↑v+1> ::= <Y ↑v>"y"
4 <Z ↑v+1> ::= <Z ↑v>"z"
5 <X ↑0>    ::= ε
6 <Y ↑0>    ::= ε
7 <Z ↑0>    ::= ε
```

Listing 30.9: The small example G_1 for Extended Attribute Grammars.

In the listing, below a typical expansion of G_1 is illustrated. Since the same attribute `v` is attached to all three non-terminals `X`, `Y`, and `Z`, the terminal symbols `x`, `y`, and `z` will always occur equally often. The context-sensitive grammar specified in Listing 30.9 thus defines sentences in the form $\texttt{x}^n\texttt{y}^n\texttt{z}^n$.

```
1 <S> ⟶ <X ↑2><Y ↑2><Z ↑2> ⟶ <X ↑1>x<Y ↑2><Z ↑2>
2     ⟶ <X ↑0>xx<Y ↑2><Z ↑2> ⟶ xx<Y ↑2><Z ↑2>
3     ⟶ xx<Y ↑1>y<Z ↑2> ⟶ xx<Y ↑0>yy<Z ↑2>
4     ⟶ xxyy<Z ↑2> ⟶ xxyy<Z ↑1>z ⟶ xxyy<Z ↑0>zz
5     ⟶ xxyyzz
```

Listing 30.10: A typical expansion of G_1.

[76] http://en.wikipedia.org/wiki/S-attributed_grammar [accessed 2007-07-04]
[77] http://en.wikipedia.org/wiki/Bottom-up_parsing [accessed 2007-07-04]
[78] http://en.wikipedia.org/wiki/GNU_Bison [accessed 2007-07-04]
[79] http://en.wikipedia.org/wiki/Flex_lexical_analyser [accessed 2007-07-04]

Another example for extended attribute grammars, again borrowed from [1874], demonstrates the specification of binary numbers. We can define a grammar $G_2 = (N, \Sigma, P, S)$ for all binary numbers. In this grammar, the start symbol `S` will have an attribute including the value of number represented by the generated sentence. Here we need three non-terminal symbols $N = \{\texttt{S}, \texttt{T}, \texttt{B}\}$ and only two terminal symbols $\Sigma = \{\texttt{0}, \texttt{1}\}$. The productions P are specified as follows:

```
1 <S ↑b>       ::= <T ↓0 ↑b>
2 <T ↓a ↑b>    ::= <B ↓a ↑b>
3 <T ↓a ↑b+c> ::= <T ↓a+1 ↑b><B ↓a ↑c>
4 <B ↓a ↑0>    ::= "0"
5 <B ↓a ↑2^a>   ::= "1"
```

Listing 30.11: An extended attribute grammar G_2 for binary numbers.

Figure 30.10 illustrates one possible expansion of the start symbol S = `S` with the extended attribute grammar G_2. As you can see, `S` has attached the (decimal) value `10` corresponding to the (binary) value `1010` of the binary string represented by the generated sentence.

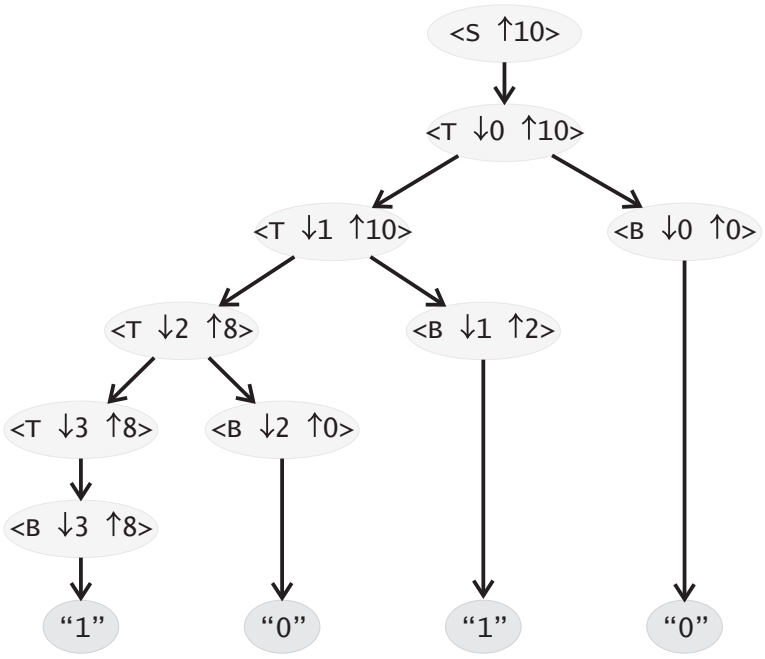


Figure 30.10: One possible expansion of the example grammar G_2.

Extended Attribute Grammars are sufficient to specify the syntax and semantics of many programming languages [2162].

30.3.8 Adaptive Grammars

Definition 30.41 (Adaptive Grammar). An Adaptive Grammar[80] $G = (N, \Sigma, P, S)$ is a formal grammar developed by Shutt [1874] in which the set of non-terminal symbols N, the set of terminal symbols Σ and the set of productions P may vary during parsing.

Shutt [1874] furthermore discusses Recursive Adaptive Grammars (RAG) which are a Turing-complete formalism but yet retain the elegance of context-free grammars.

[80] http://en.wikipedia.org/wiki/Adaptive_grammar [accessed 2007-07-13]

30.3.9 Christiansen Grammars

Christiansen [402] introduces an adaptable grammar model that combines Extended Attribute Grammars with the ability to adapt according to Definition 30.41 [402, 405, 406].

Unfortunately, Christiansen [403] calls his adaptable attribute grammars "generative grammars" [403, 404] which has already another meaning (see Section 30.3.2 on page 562). We therefore resort to the term "Christiansen Grammars" coined by Shutt [1874] from whom we again will borrow the examples. As described in [402, 406], a Christiansen Grammar is an Extended Attribute Grammar where the first attribute of each non-terminal symbol $n \in N$ is an inherited Christiansen Grammar itself. This attribute is called *language attribute* and the expansion of the non-terminal symbol it belongs to must be done according to the grammar represented by it.

```
1 <n ↓g ↕a ↕b ...>
```

The statement `X<n ↓g ↕ ...>Z ::= XYZ` (with $\mathtt{X}, \mathtt{Y}, \mathtt{Z} \in V$ and $\mathtt{n} \in N$) hence only holds if `<n ↓g ↕a ...>::= Y` according to the grammar attribute `g`.

Let us start with a simple example grammar $G_3 = (N, \Sigma, P, S)$ with the non-terminal symbols `alpha-list` and `alpha`, the Latin alphabet as set of terminal symbols Σ, the `alpha-list` as start symbol S and the set of productions P as specified below.

```
1 <alpha-list ↓g ↑w>      ::= <alpha ↓g ↑w>
2 <alpha-list ↓g ↑w1∘w2> ::= <alpha ↓g ↑w1><alpha-list ↓g ↑w2>
3 <alpha ↓g ↑"a">         ::= "a"
4 ...
5 <alpha ↓g ↑"z">         ::= "z"
```

Listing 30.12: A Christiansen Grammar creating character strings.

It clearly generates the character strings over the Latin alphabet. The start symbol has two attributes: The inherited Christiansen Grammar `g` will be handed down to all generated symbols. The attribute `w` on the other hand is synthesized from these symbols and contains the character string generated.

Basing on this grammar which still is a mere EAG in principle, we build the Christiansen Grammar $G_4 = (N, \Sigma, P, S)$ for a subset of the C (or Java) programming language where all value assignments are valid:

```
1  ...
2  <program ↓g0>          ::= "{"<decl-list ↓g0 ↑g1>
3                                <stmnt-list ↓g1>"}"
4  <decl-list ↓g ↑g>      ::= ε
5  <decl-list ↓g0 ↑g2>    ::= <decl ↓g0 ↑g2><decl-list ↓g1 ↑g2>
6  <decl ↓g ↑g+new-rule> ::= "int" <alpha-list ↓g ↑w> ";"
7             where new-rule is <id ↓h> ::= w
8  <stmnt-list ↓g>        ::= ε
9  <stmnt-list ↓g>        ::= <stmnt ↓g><stmnt-lst ↓g>
10 <stmnt ↓g>             ::= <id ↓g> "=" <id ↓g> ";"
```

Listing 30.13: Christiansen Grammar for a simple programming language.

Whenever the non-terminal symbol `decl` is expanded, it also adds a new rule to the grammar. By introducing a new production for the symbol `id`, the declared variable becomes available in `stmt` since the grammar is synthesized upwards to the production for `program` and then inherited downwards into `stmt-lst`. A more thorough example of Christiansen Grammar in the context of Genetic Programming can be found in Listing 4.7.

30.3.10 Tree-Adjoining Grammars

Tree-adjoining grammars[81] (TAG, also called tree-adjunct grammars) are another method for defining formal grammars which has been developed by Joshi [1072]. [1704, 1073] Different

[81] http://en.wikipedia.org/wiki/Tree-adjoining_grammar [accessed 2007-07-03]

from BNF and EBNF, they are based on trees instead of plain strings. The inner nodes of the (fully expanded) trees correspond to non-terminal symbols and the leaves to terminal symbols.

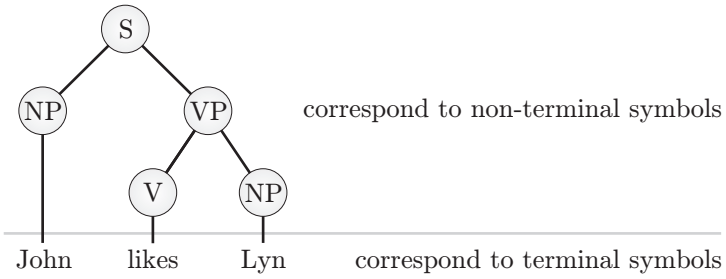


Figure 30.11: An example TAG tree.

The simple TAG tree illustrated in Figure 30.11 is borrowed from [1073] as well as some of the following examples. The tree structure of tree-adjoining grammars has one striking advantage compared to the flat rules in context-free grammars: the increased *domain of locality* [1704]. If we process for example an EBNF rule, we can only expand the non-terminal symbols at our current "level" of the derivation tree. Below we show that a text in an EBNF grammar similar to the one of Figure 30.11 could be resolved step by step. The variable `VP` expanded in line 8 for instance cannot be accessed or modified in line 10 anymore, although it is clearly part of the sentence construction.

```
1  S  ::= NP, VP ;
2  NP ::= "John" | "Lyn" ;
3  VP ::= V, NP ;
4  V  ::= "likes" ;
5
6  text ⟶ S
7  text ⟶ NP VP
8  text ⟶ "John" VP
9  text ⟶ "John" V NP
10 text ⟶ "John" "likes" NP
11 text ⟶ "John" "likes" "Lyn"
```

Listing 30.14: Another simple context-free grammar.

The extended domain of locality (EDL) in TAG trees is utilized with the two modification operators *substitution* and *adjunction*.

We can substitute a tree β into a tree α if there is a non-terminal leaf symbol ν in α that has the same label as the root of β. The stump of β then replaces the node ν in α. In Figure 30.12 we outline how two trees β_1 and β_2 are substituted into a TAG tree α and a new tree α' is created.

Substitution is equivalent to the non-terminal expansion in BNF. The adjunction operator however adds access to the aforementioned layers which are buried in context-free grammars. In order to perform an adjunction, the tree α has to include one non-terminal symbol ν at some random place. The root of the *auxiliary tree* is also labeled with ν and so is at least one of its leafs. We now can replace the node marked with ν in α with tree β. Whatever was attached to ν before now replaces the leaf node ν in β. The leaf node ν in beta often is additionally marked with an asterisk ($*$). Figure 30.13 sketches such a replacement, with the result that the new sentence α' now contains the word `"really"`.

With adjunction, TAGs are somewhere in between context-sensitive and context-free grammars. In the definition of a tree-adjoining grammar $G = (N, \Sigma, A, I, S)$, A is the set

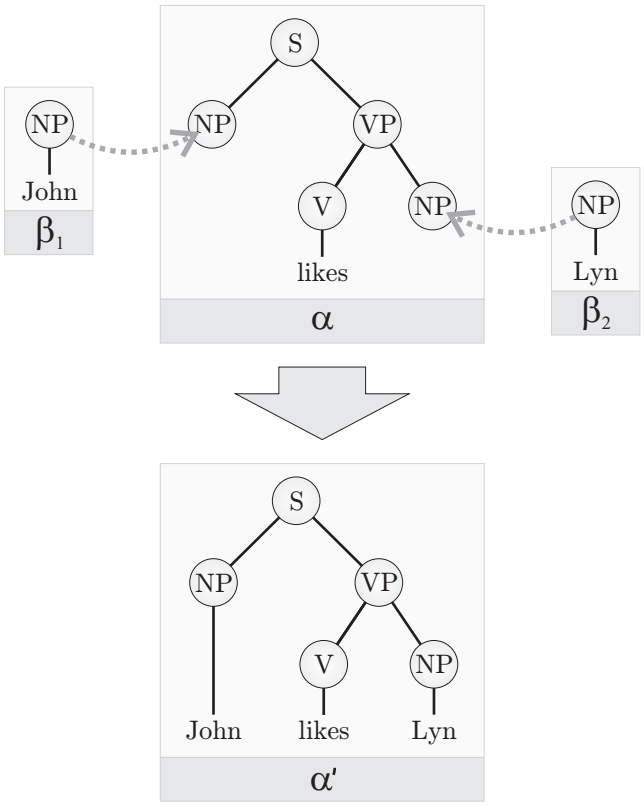


Figure 30.12: An example for the substitution operation.

of auxiliary trees to be used in the adjunction operations. I is the set of initial trees that can be substituted into existing trees. The unison of I and A, $E = I \cup A$ is called the set of elementary trees and replaces the set of productions P used in Chomsky grammars. N and Σ retain their meaning as set of non-terminal and terminal symbols respectively. Trees with the non-terminal symbol $X \in N$ as root are called X-type trees. $S \in N$ denotes the starting symbol and there must be at least one S-type elementary tree.

Definition 30.42 (Lexicalized Tree-Adjoining Grammars). A lexicalized tree-adjoining grammar (LTAG) is a tree-adjoining grammar where each elementary tree $t \in E$ contains a terminal symbol $X \in \Sigma$. Although they are more restricted, LTAGs are equivalent to TAGs.

A discussion on derivation trees of tree-adjoining grammars can be found in Section 4.5.9 on page 189.

30.3.11 S-expressions

S-expressions[82] (where S stands for symbolic) or *sexp* are data structures for presenting complex data. They are probably best known for their usage in the Lisp[83] [1377, 1379, 1378] and Scheme[84] [612] programming languages. Their most common feature is that they are parenthesized prefix notations (often also known as Polish notation[85]).

[82] http://en.wikipedia.org/wiki/S-expression [accessed 2007-07-03]
[83] http://en.wikipedia.org/wiki/Lisp_programming_language [accessed 2007-07-03]
[84] http://en.wikipedia.org/wiki/Scheme_%28programming_language%29 [accessed 2007-07-03]
[85] http://en.wikipedia.org/wiki/Polish_notation [accessed 2007-07-04]

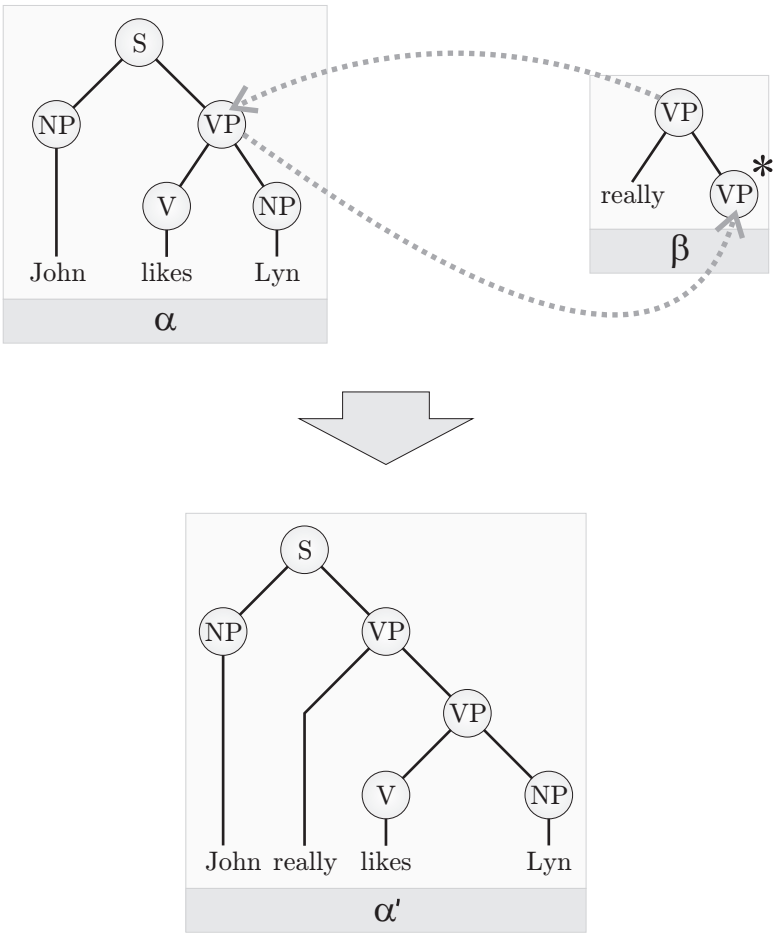


Figure 30.13: An example for the adjunction operation.

In 1997, Rivest [1742] handed in a standardization draft for S-expressions to be considered for publication as RFC. It was, however, never approved but still is the foundation for many other publications and RFCs.

```
(defun fibonacci (N)
  (if (or (zerop N) (= N 1))
      1
    (+ (fibonacci (- N 1)) (fibonacci (- N 2)))))
```

Listing 30.15: A small Lisp-example: How to compute Fibonacci numbers.

Appendices

A

GNU Free Documentation License (FDL)

Version 1.2, November 2002

A.1 Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

A.2 Applicability and Definitions

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "Entitled XYZ" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "Acknowledgements", "Dedications", "Endorsements", or "History".) To "Preserve the Title" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

A.3 Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in Section A.4.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

A.4 Copying in Quantity

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers

in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

A.5 Modifications

You may copy and distribute a Modified Version of the Document under the conditions of Section A.3 and Section A.43 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.
D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
H. Include an unaltered copy of this License.
I. Preserve the section Entitled "History", Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
K. For any section Entitled "Acknowledgements" or "Dedications", Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
M. Delete any section Entitled "Endorsements". Such a section may not be included in the Modified Version.
N. Do not retitle any existing section to be Entitled "Endorsements" or to conflict in title with any Invariant Section.
O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section Entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

A.6 Combining Documents

You may combine the Document with other documents released under this License, under the terms defined in Section A.5 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled "History" in the various original documents, forming one section Entitled "History"; likewise combine any sections Entitled "Acknowledgements", and any sections Entitled "Dedications". You must delete all sections Entitled "Endorsements."

A.7 Collections of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

A.8 Aggregation with Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an "aggregate" if the copyright resulting from the compilation is not used to limit the legal rights of the compilation's users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of Section A.4 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document's Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

A.9 Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of Section A.5. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled "Acknowledgements", "Dedications", or "History", the requirement (Section A.5) to Preserve its Title (Section A.2) will typically require changing the actual title.

A.10 Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

A.11 Future Revisions of this License

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

Copyright (c) YEAR YOUR NAME.
Permission is granted to copy, distribute and/or modify this document
under the terms of the GNU Free Documentation License, Version 1.2
or any later version published by the Free Software Foundation;

with no Invariant Sections, no Front-Cover Texts, and no Back-Cover Texts.
A copy of the license is included in the section entitled "GNU
Free Documentation License".

If you have Invariant Sections, Front-Cover Texts and Back-Cover Texts, replace the "with...Texts." line with this:

with the Invariant Sections being LIST THEIR TITLES, with the
Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.

If you have Invariant Sections without Cover Texts, or some other combination of the three, merge those two alternatives to suit the situation.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.

B

GNU Lesser General Public License (LPGL)

Version 2.1, February 1999

51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

[This is the first released version of the Lesser GPL. It also counts as the successor of the GNU Library Public License, version 2, hence the version number 2.1]

B.1 Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public Licenses are intended to guarantee your freedom to share and change free software–to make sure the software is free for all its users.

This license, the Lesser General Public License, applies to some specially designated software packages–typically libraries–of the Free Software Foundation and other authors who decide to use it. You can use it too, but we suggest you first think carefully about whether this license or the ordinary General Public License is the better strategy to use in any particular case, based on the explanations below.

When we speak of free software, we are referring to freedom of use, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish); that you receive source code or can get it if you want it; that you can change the software and use pieces of it in new free programs; and that you are informed that you can do these things.

To protect your rights, we need to make restrictions that forbid distributors to deny you these rights or to ask you to surrender these rights. These restrictions translate to certain responsibilities for you if you distribute copies of the library or if you modify it.

For example, if you distribute copies of the library, whether gratis or for a fee, you must give the recipients all the rights that we gave you. You must make sure that they, too, receive or can get the source code. If you link other code with the library, you must provide complete object files to the recipients, so that they can relink them with the library after making changes to the library and recompiling it. And you must show them these terms so they know their rights.

We protect your rights with a two-step method: (1) we copyright the library, and (2) we offer you this license, which gives you legal permission to copy, distribute and/or modify the library.

To protect each distributor, we want to make it very clear that there is no warranty for the free library. Also, if the library is modified by someone else and passed on, the recipients should know that what they have is not the original version, so that the original author's reputation will not be affected by problems that might be introduced by others.

Finally, software patents pose a constant threat to the existence of any free program. We wish to make sure that a company cannot effectively restrict the users of a free program by obtaining a restrictive license from a patent holder. Therefore, we insist that any patent license obtained for a version of the library must be consistent with the full freedom of use specified in this license.

Most GNU software, including some libraries, is covered by the ordinary GNU General Public License. This license, the GNU Lesser General Public License, applies to certain designated libraries, and is quite different from the ordinary General Public License. We use this license for certain libraries in order to permit linking those libraries into non-free programs.

When a program is linked with a library, whether statically or using a shared library, the combination of the two is legally speaking a combined work, a derivative of the original library. The ordinary General Public License therefore permits such linking only if the entire combination fits its criteria of freedom. The Lesser General Public License permits more lax criteria for linking other code with the library.

We call this license the "Lesser" General Public License because it does Less to protect the user's freedom than the ordinary General Public License. It also provides other free software developers Less of an advantage over competing non-free programs. These disadvantages are the reason we use the ordinary General Public License for many libraries. However, the Lesser license provides advantages in certain special circumstances.

For example, on rare occasions, there may be a special need to encourage the widest possible use of a certain library, so that it becomes a de-facto standard. To achieve this, non-free programs must be allowed to use the library. A more frequent case is that a free library does the same job as widely used non-free libraries. In this case, there is little to gain by limiting the free library to free software only, so we use the Lesser General Public License.

In other cases, permission to use a particular library in non-free programs enables a greater number of people to use a large body of free software. For example, permission to use the GNU C Library in non-free programs enables many more people to use the whole GNU operating system, as well as its variant, the GNU/Linux operating system.

Although the Lesser General Public License is Less protective of the users' freedom, it does ensure that the user of a program that is linked with the Library has the freedom and the wherewithal to run that program using a modified version of the Library.

The precise terms and conditions for copying, distribution and modification follow. Pay close attention to the difference between a "work based on the library" and a "work that uses the library". The former contains code derived from the library, whereas the latter must be combined with the library in order to run.

B.2 Terms and Conditions for Copying, Distribution and Modification

1. This License Agreement applies to any software library or other program which contains a notice placed by the copyright holder or other authorized party saying it may be distributed under the terms of this Lesser General Public License (also called "this License"). Each licensee is addressed as "you".
   A "library" means a collection of software functions and/or data prepared so as to be conveniently linked with application programs (which use some of those functions and data) to form executables.
   The "Library", below, refers to any such software library or work which has been distributed under these terms. A "work based on the Library" means either the Library or any derivative work under copyright law: that is to say, a work containing the Library or a portion of it, either verbatim or with modifications and/or translated straightforwardly into another language. (Hereinafter, translation is included without limitation in the term "modification".)
   "Source code" for a work means the preferred form of the work for making modifications to it. For a library, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the library.
   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running a program using the Library is not restricted, and output from such a program is covered only if its contents constitute a work based on the Library (independent of the use of the Library in a tool for writing it). Whether that is true depends on what the Library does and what the program that uses the Library does.
2. You may copy and distribute verbatim copies of the Library's complete source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each

copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and distribute a copy of this License along with the Library.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

3. You may modify your copy or copies of the Library or any portion of it, thus forming a work based on the Library, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:
   a) The modified work must itself be a software library.
   b) You must cause the files modified to carry prominent notices stating that you changed the files and the date of any change.
   c) You must cause the whole of the work to be licensed at no charge to all third parties under the terms of this License.
   d) If a facility in the modified Library refers to a function or a table of data to be supplied by an application program that uses the facility, other than as an argument passed when the facility is invoked, then you must make a good faith effort to ensure that, in the event an application does not supply such function or table, the facility still operates, and performs whatever part of its purpose remains meaningful.
   (For example, a function in a library to compute square roots has a purpose that is entirely well-defined independent of the application. Therefore, Subsection 2d requires that any application-supplied function or table used by this function must be optional: if the application does not supply it, the square root function must still compute square roots.)
   These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Library, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Library, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.
   Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Library.
   In addition, mere aggregation of another work not based on the Library with the Library (or with a work based on the Library) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.
4. You may opt to apply the terms of the ordinary GNU General Public License instead of this License to a given copy of the Library. To do this, you must alter all the notices that refer to this License, so that they refer to the ordinary GNU General Public License, version 2, instead of to this License. (If a newer version than version 2 of the ordinary GNU General Public License has appeared, then you can specify that version instead if you wish.) Do not make any other change in these notices.
   Once this change is made in a given copy, it is irreversible for that copy, so the ordinary GNU General Public License applies to all subsequent copies and derivative works made from that copy.
   This option is useful when you wish to copy part of the code of the Library into a program that is not a library.
5. You may copy and distribute the Library (or a portion or derivative of it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange.
   If distribution of object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place satisfies the requirement to distribute the source code, even though third parties are not compelled to copy the source along with the object code.
6. A program that contains no derivative of any portion of the Library, but is designed to work with the Library by being compiled or linked with it, is called a "work that uses the Library". Such a work, in isolation, is not a derivative work of the Library, and therefore falls outside the scope of this License.

However, linking a "work that uses the Library" with the Library creates an executable that is a derivative of the Library (because it contains portions of the Library), rather than a "work that uses the library". The executable is therefore covered by this License. Section 6 states terms for distribution of such executables.

When a "work that uses the Library" uses material from a header file that is part of the Library, the object code for the work may be a derivative work of the Library even though the source code is not. Whether this is true is especially significant if the work can be linked without the Library, or if the work is itself a library. The threshold for this to be true is not precisely defined by law.

If such an object file uses only numerical parameters, data structure layouts and accessors, and small macros and small inline functions (ten lines or less in length), then the use of the object file is unrestricted, regardless of whether it is legally a derivative work. (Executables containing this object code plus portions of the Library will still fall under Section 6.)

Otherwise, if the work is a derivative of the Library, you may distribute the object code for the work under the terms of Section 6. Any executables containing that work also fall under Section 6, whether or not they are linked directly with the Library itself.

7. As an exception to the Sections above, you may also combine or link a "work that uses the Library" with the Library to produce a work containing portions of the Library, and distribute that work under terms of your choice, provided that the terms permit modification of the work for the customer's own use and reverse engineering for debugging such modifications.

   You must give prominent notice with each copy of the work that the Library is used in it and that the Library and its use are covered by this License. You must supply a copy of this License. If the work during execution displays copyright notices, you must include the copyright notice for the Library among them, as well as a reference directing the user to the copy of this License. Also, you must do one of these things:

   a) Accompany the work with the complete corresponding machine-readable source code for the Library including whatever changes were used in the work (which must be distributed under Sections 1 and 2 above); and, if the work is an executable linked with the Library, with the complete machine-readable "work that uses the Library", as object code and/or source code, so that the user can modify the Library and then relink to produce a modified executable containing the modified Library. (It is understood that the user who changes the contents of definitions files in the Library will not necessarily be able to recompile the application to use the modified definitions.)
   b) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (1) uses at run time a copy of the library already present on the user's computer system, rather than copying library functions into the executable, and (2) will operate properly with a modified version of the library, if the user installs one, as long as the modified version is interface-compatible with the version that the work was made with.
   c) Accompany the work with a written offer, valid for at least three years, to give the same user the materials specified in Subsection 6a, above, for a charge no more than the cost of performing this distribution.
   d) If distribution of the work is made by offering access to copy from a designated place, offer equivalent access to copy the above specified materials from the same place.
   e) Verify that the user has already received a copy of these materials or that you have already sent this user a copy.

   For an executable, the required form of the "work that uses the Library" must include any data and utility programs needed for reproducing the executable from it. However, as a special exception, the materials to be distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

   It may happen that this requirement contradicts the license restrictions of other proprietary libraries that do not normally accompany the operating system. Such a contradiction means you cannot use both them and the Library together in an executable that you distribute.

8. You may place library facilities that are a work based on the Library side-by-side in a single library together with other library facilities not covered by this License, and distribute such a combined library, provided that the separate distribution of the work based on the Library and of the other library facilities is otherwise permitted, and provided that you do these two things:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities. This must be distributed under the terms of the Sections above.
b) Give prominent notice with the combined library of the fact that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

9. You may not copy, modify, sublicense, link with, or distribute the Library except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense, link with, or distribute the Library is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.
10. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Library or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Library (or any work based on the Library), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Library or works based on it.
11. Each time you redistribute the Library (or any work based on the Library), the recipient automatically receives a license from the original licensor to copy, distribute, link with or modify the Library subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties with this License.
12. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Library at all. For example, if a patent license would not permit royalty-free redistribution of the Library by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Library.
    If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply, and the section as a whole is intended to apply in other circumstances.
    It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.
    This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.
13. If the distribution and/or use of the Library is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Library under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.
14. The Free Software Foundation may publish revised and/or new versions of the Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.
    Each version is given a distinguishing version number. If the Library specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Library does not specify a license version number, you may choose any version ever published by the Free Software Foundation.
15. If you wish to incorporate parts of the Library into other free programs whose distribution conditions are incompatible with these, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we

sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

B.3 No Warranty

1. BECAUSE THE LIBRARY IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE LIBRARY, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE LIBRARY "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE LIBRARY IS WITH YOU. SHOULD THE LIBRARY PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.
2. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE LIBRARY AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE LIBRARY (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE LIBRARY TO OPERATE WITH ANY OTHER SOFTWARE), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

B.4 How to Apply These Terms to Your New Libraries

If you develop a new library, and you want it to be of the greatest possible use to the public, we recommend making it free software that everyone can redistribute and change. You can do so by permitting redistribution under these terms (or, alternatively, under the terms of the ordinary General Public License).

To apply these terms, attach the following notices to the library. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

one line to give the library's name and an idea of what it does. Copyright (C) year name of author

This library is free software; you can redistribute it and/or modify it under the terms of the GNU Lesser General Public License as published by the Free Software Foundation; either version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License along with this library; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA

Also add information on how to contact you by electronic and paper mail.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the library, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the library 'Frob' (a library for tweaking knobs) written by James Random Hacker.

signature of Ty Coon, 1 April 1990 Ty Coon, President of Vice

That's all there is to it!

C

Credits and Contributors

In this section I want to give credits to whom they deserve (by directly or indirectly contributing to this book). So its props to:

Stefan Achler
For working together at the 2007 DATA-MINING-CUP Contest.
See Section 22.1.2 on page 374 and [2178].

Steffen Bleul
For working together at the 2006, 2007, and 2008 Web Service Challenge.
See Section 22.2 on page 383 and [225, 2179, 226, 2184]

Raymond Chiong
For co-authoring a bookchapter corresponding to Section 1.4 and by doing so, helping to correct many mistakes.
See Section 1.4 on page 56

Distributed Systems Group, University of Kassel
To the whole Distributed Systems Group at the University of Kassel for being supportive and coming over again and again with new ideas and helpful advices.
Each and every research project described in this book.

Qiu Li
For careful reading and pointing out inconsistencies and spelling mistakes.
See Chapter 27

Gan Min
For careful reading and pointing out inconsistencies and spelling mistakes.
See especially the Pareto optimization area Section 1.2.2 on page 31.

Kurt Geihs
For being supportive and contributing to many of my research projects.
See for instance Section 22.1.2 on page 374 and [2176, 2177, 2180, 225, 226]

Martin Göb
For working together at the 2007 DATA-MINING-CUP Contest.
See Section 22.1.2 on page 374 and [2178].

Ibrahim Ibrahim
For careful proofreading.
See Chapter 1 on page 21.

Antonio Nebro

For co-authoring a bookchapter corresponding to Section 1.4 and by doing so, helping to correct many mistakes.
See Section 1.2 on page 25

Stefan Niemczyk
For careful proofreading and scientific support.
See for instance Chapter 1 on page 21, Section 21.2.7 on page 341, and [2185, 1533].

Roland Reichle
For fruitful discussions and scientific support.
See for instance Section 28.7.2 on page 500, Section 21.2.7 on page 341, and [2185].

Laurent Rodriguez
For contributing corrections in some of the chapters.
See for instance Section 28.2 on page 466 and Section 27.6 on page 459.

Richard Seabrook
For pointing out mistakes in some of the set theory definitions.
See Chapter 27 on page 455

Hendrik Skubch
For fruitful discussions and scientific support.
See for instance Section 1.3 on page 40, Section 21.2.7 on page 341, and [2185].

Christian Voigtmann
For working together at the 2007 and 2008 DATA-MINING-CUP Contest.
See Section 22.1.2 on page 374 and [2178].

Wibul Wongpoowarak
For valuable suggestions in different areas, like including random optimization.
See for instance Section 28.9.1 on page 526 [2255], Chapter 11 on page 259.

Michael Zapf
For helping me to make many sections more comprehensible.
See for instance Section 24.1.2 on page 415, Section 22.1.2 on page 374, Section 4.5 on page 176, and [2182, 2310].

D

Citation Suggestion

```
@book{W2009GOEB,
    author       = {Thomas Weise},
    title        = {Global Optimization Algorithms - Theory and Application},
    year         = {2009},
    month        =  jun # {~26,},
    howpublished = {Online as e-book},
    edition      = {Second},
    institution  = {University of Kassel, Distributed Systems Group},
    organization = {University of Kassel, Distributed Systems Group},
    publisher    = {Self-Published},
    copyright    = {Copyright (c) 2006-2009 Thomas Weise, licensed under GNU FDL},
    keywords     = {Global Optimization, Evolutionary Computation, Evolutionary Algorithms,
                    Genetic Algorithms, Genetic Programming, Evolution Strategy, Learning
                    Classifier Systems, Extremal Optimization, Raindrop Method, Ant Colony
                    Optimization, Particle Swarm Optimization, Downhill Simplex, Simulated
                    Annealing, Hill Climbing, Sigoa, DGPF, Java},
    language     = {en},
    url          = {http://www.it-weise.de/},
    url          = {http://www.sigoa.org/},
    url          = {http://www.vs.uni-kassel.de/staff/researchers/weise/},
    note         = {Online available at http://www.it-weise.de/}
  }
```

```
%0    Book
%A    Weise, Thomas
%T    Global Optimization Algorithms - Theory and Application
%F    Thomas Weise: "Global Optimization Algorithms - Theory and Application"
%I    Thomas Weise
%D    Second
%8    Second
%7    Second
%9    Online available as e-book at http://www.it-weise.de/
```

References

A

[1] *Proceedings of the The Fifth Conference on Innovative Applications of Artificial Intelligence (IAAI 1993)*, July 11–15, 1993, Washington, DC, USA. AAAI. ISBN: 0-9292-8046-6. See http://www.aaai.org/Conferences/IAAI/iaai93.php [accessed 2007-09-06].

[2] *Agent Modeling, Papers from the 1996 AAAI Workshop*, 1996. AAAI Press. ISBN: 1-5773-5008-1. See [410].

[3] *Deep Blue Versus Kasparov: The Significance for Artificial Intelligence, Collected Papers from the 1997 AAAI Workshop*, 1997. AAAI Press. ISBN: 1-5773-5031-6.

[4] Hervé Abdi. Centroïd, center of gravity, center of mass, barycenter. In Neil J. Salkind, editor, *Encyclopedia of Measurement and Statistics*, volume 1, page 128 ff. Thousand Oaks (California): Sage Publications, Inc, October 2006. ISBN: 978-1-41291-611-0. Online available at http://www.utdallas.edu/~herve/Abdi-Centroid2007-pretty.pdf [accessed 2007-08-11].

[5] Hal Abelson, editor. *Workshop on Amorphous Computing*, September 13–14, 1999, DARPA ITO, MIT Artificial Intelligence Laboratory, Cambridge, MA, USA. MIT AI Lab. Online available at http://www-swiss.ai.mit.edu/projects/amorphous/workshop-sept-99/ [accessed 2008-07-27].

[6] Ajith Abraham and Mario Köppen, editors. *Hybrid Information Systems, Proceedings of the First International Workshop on Hybrid Intelligent Systems*, Advances in Soft Computing, December 11–12, 2002, Adelaide University's main campus, Adelaide, Australia. Physica-Verlag. ISBN: 3-7908-1480-6. see http://his01.hybridsystem.com/ [accessed 2007-09-01].

[7] Ajith Abraham, Javier Ruiz del Solar, and Mario Köppen, editors. *Soft Computing Systems – Design, Management and Applications, HIS 2002*, volume 87 of *Frontiers in Artificial Intelligence and Applications*, December 1–4, 2002, Santiago, Chile. IOS Press. ISBN: 1-5860-3297-6.

[8] Ajith Abraham, Mario Köppen, and Katrin Franke, editors. *Design and Application of Hybrid Intelligent Systems, HIS03, Proceedings of the Third International Conference on Hybrid Intelligent Systems*, volume 105 of *Frontiers in Artificial Intelligence and Applications*, December 14–17, 2003, Monash Conference Centre, Level 7, 30 Collins Street, Melbourne, Australia. IOS Press. ISBN: 1-5860-3394-8. see http://his03.hybridsystem.com/ [accessed 2007-09-01].

[9] Ajith Abraham, Lakhmi C. Jain, and Robert Goldberg, editors. *Evolutionary Multiobjective Optimization – Theoretical Advances and Applications*. Advanced Information and Knowledge Processing. Springer, Berlin, July 30, 2005. ISBN: 978-1-85233-787-2, 1-8523-3787-7. Partly online available at http://books.google.

de/books?id=Ei7q1YSjiSAC [accessed 2008-06-07]. Series editors: Lakhmi C. Jain and Xindong Wu.

[10] Faris N. Abuali, Dale A. Schoenefeld, and Roger L. Wainwright. Designing telecommunications networks using genetic algorithms and probabilistic minimum spanning trees. In *SAC '94: Proceedings of the 1994 ACM Symposium on Applied computing*, pages 242–246, March 6–8, 1994, Phoenix, Arizona, United States. ACM, New York, NY, USA. ISBN: 0-8979-1647-6. doi:10.1145/326619.326733. Online available at http://doi.acm.org/10.1145/326619.326733 and http://euler.mcs.utulsa.edu/~rogerw/papers/Abuali-Prufer-CSC-94.pdf [accessed 2008-08-28].

[11] Faris N. Abuali, Dale A. Schoenefeld, and Roger L. Wainwright. Terminal assignment in a communications network using genetic algorithms. In *CSC'94: Proceedings of the 22nd annual ACM computer science conference on Scaling up: meeting the challenge of complexity in real-world computing applications*, pages 74–81, March 8–10, 1994, Phoenix, Arizona, United States. ACM, New York, NY, USA. ISBN: 0-8979-1634-4. doi:10.1145/197530.197559. Online available at http://euler.mcs.utulsa.edu/~rogerw/papers/Abuali-Terminal-CSC-94.pdf [accessed 2008-08-28].

[12] David H. Ackley. *A Connectionist Machine for Genetic Hillclimbing*, volume 28 of *Kluwer International Series in Engineering and Computer Science / The Springer International Series in Engineering and Computer Science*. Kluwer Academic Publishers / Springer US, Norwell/Boston, MA, USA, August 31, 1987. ISBN: 0-8983-8236-X, 978-0-89838-236-5. Originally presented as the author's thesis (Ph. D.) – Carnegie Mellon University, Pittsburgh, 1987.

[13] Christoph Adami, Richard K. Belew, Hiroaki Kitano, and Charles E. Taylor, editors. *Artificial Life VI: Proceedings of the sixth international conference on Artificial life*, volume 6 of *Complex Adaptive Systems*, June 1998, Madison, Wisconsin, United States. MIT Press, Cambridge, MA, USA. ISBN: 0-2625-1099-5, 978-0-26251-099-8.

[14] Aesop. The boy and the filberts, around 600 BC. Online available at http://www.aesopfables.com/cgi/aesop1.cgi?1&TheBoyandtheFilberts [accessed 2008-11-29].

[15] Alan Agresti. A survey of exact inference for contingency tables. *Statistical Science*, 7(1):131–153, 1992. Online available at http://projecteuclid.org/handle/euclid.ss/1177011454 [accessed 2008-12-08].

[16] Alan Agresti. *An Introduction to Categorical Data Analysis*. Wiley-Interscience, first edition, January 1996. ISBN: 978-0-47111-338-6.

[17] Arturo Hernández Aguirre, Bill P. Buckles, and Carlos Artemio Coello Coello. A genetic programming approach to logic function synthesis by means of multiplexers. In *Evolvable Hardware, Proceedings of 1st NASA/DoD Workshop on Evolvable Hardware (EH'99)*, pages 46–53. IEEE Computer Society, July 1999, Pasadena, CA, USA. ISBN: 0-7695-0256-3. Online available at http://citeseer.ist.psu.edu/521808.html [accessed 2007-09-09].

[18] Chang Wook Ahn and R. S. Ramakrishna. Augmented compact genetic algorithm. In Roman Wyrzykowski, Jack Dongarra, Marcin Paprzycki, and Jerzy Wasniewski, editors, *PPAM 2003: Proceedings of 5th International Conference on Parallel Processing and Applied Mathematics, revised papers*, volume 3019(1–2)/2004 of *Lecture Notes in Computer Science (LNCS)*, pages 560–565. Springer Berlin / Heidelberg, September 2003, Czestochowa, Poland. ISBN: 978-3-54021-946-0. doi:10.1007/b97218. Published 2004.

[19] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison Wesley, international ed edition, January 1974. ISBN: 978-0-20100-029-0.

[20] J. H. Ahrens and U. Dieter. Computer methods for sampling from gamma, beta, poisson and bionomial distributions. *Computing*, 12(3):223–246, September 10, 1974. ISSN: 0010-485X (Print) 1436-5057 (Online). doi:10.1007/BF02293108.

Online available at http://www.springerlink.com/content/712tn58716485674/fulltext.pdf [accessed 2007-09-15].

[21] Alex Aiken. Lecture notes cs 264, Spring 1995. Online available at http://www.cs.berkeley.edu/~aiken/cs264/lectures/attribute-grammars [accessed 2007-07-03].

[22] Jan Aikins and Howard Shrobe, editors. *Proceedings of the The Seventh Conference on Innovative Applications of Artificial Intelligence (IAAI 1995)*, August 21–31, 1995, Montréal, Québec, Canada. ISBN: 978-0-92928-081-3. See http://www.aaai.org/Conferences/IAAI/iaai95.php [accessed 2007-09-06].

[23] P. Aiyarak, A. S. Saket, and Mark C. Sinclair. Genetic programming approaches for minimum cost topology optimisation of optical telecommunication networks. In *Second International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, GALESIA*, 1997. In proceedings [990]. Online available at http://citeseer.ist.psu.edu/30631.html and http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/aiyarak_1997_GPtootn.html [accessed 2008-07-29].

[24] Miguel Alabau, Lhassane Idoumghar, and René Schott. New hybrid genetic algorithms for the frequency assignment problem. In *Proceedings of the 13th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'01)*, page 136 ff. IEEE Computer Society, November 7–9, 2001, Dallas, TX, USA. ISBN: 0-7695-1417-0.

[25] Jarmo T. Alander. An indexed bibliography of genetic algorithms in signal and image processing. Technical Report 94-1-SIGNAL, Department of Information Technology and Production Economics, University of Vaasa, P.O. Box 700, FIN-65101, Finland, May 18, 1998. Online available at http://citeseer.ist.psu.edu/319835.html and ftp://ftp.uwasa.fi/cs/report94-1/gaSIGNALbib.ps.Z [accessed 2008-05-18].

[26] Jarmo Tapani Alander, editor. *Proceedings of the 1st Finnish Workshop on Genetic Algorithms and Their Applications*, November 4–5, 1992, Espoo, Finland. Department of Computer Science, Helsinki University of Technology, Helsinki, Finland. Technical Report TKO-A30, partly in Finnish, published 1993.

[27] Jarmo Tapani Alander, editor. *Proceedings of the Second Finnish Workshop on Genetic Algorithms and Their Applications (2FWGA)*, March 16–18, 1994, Vaasa, Finland. Department of Information Technology and Production Economics, University of Vaasa, P.O. Box 700, FIN-65101 Vaasa, Finland. ISBN: 9-5168-3526-0. Technical Report 94-2. Online available at ftp://ftp.uwasa.fi/cs/report94-2 [accessed 2008-07-24].

[28] Jarmo Tapani Alander, editor. *Proceedings of the First Nordic Workshop on Genetic Algorithms (1NWGA, The Third Finnish Workshop on Genetic Algorithms, 3FWGA)*, Proceedings of the University of Vaasa, Nro. 2, June 9–13, 1995, Vaasa, Finland. Department of Information Technology and Production Economics, University of Vaasa, P.O. Box 700, FIN-65101 Vaasa, Finland. Technical Report 95-1. Online available at ftp://ftp.uwasa.fi/cs/1NWGA [accessed 2008-07-24].

[29] Jarmo Tapani Alander, editor. *Proceedings of the Second Nordic Workshop on Genetic Algorithms (2NWGA)*, Proceedings of the University of Vaasa, Nro. 13, August 19–23, 1996, Vaasa, Finland. Department of Information Technology and Production Economics, University of Vaasa, P.O. Box 700, FIN-65101 Vaasa, Finland. In collaboration with the Finnish Artificial Intelligence Society. Online available at ftp://ftp.uwasa.fi/cs/2NWGA [accessed 2008-07-24].

[30] Jarmo Tapani Alander, editor. *Proceedings of the Third Nordic Workshop on Genetic Algorithms (3NWGA)*, August 20–22, 1997, Helsinki, Finland. Finnish Artificial Intelligence Society (FAIS). Online available at ftp://ftp.uwasa.fi/cs/3NWGA [accessed 2008-07-24].

[31] Enrique Alba and J. Francisco Chicano. Evolutionary algorithms in telecommunications. In *IEEE Mediterranean Electrotechnical Conference, MELECON 2006*, pages 795–798. IEEE, May 16-19, 2006, Benalmádena (Málaga), Spain.

doi:10.1109/MELCON.2006.1653218. Online available at http://dea.brunel.ac.uk/ugprojects/docs/melecon06.pdf [accessed 2008-08-01].

[32] Enrique Alba and Bernabé Dorronsoro. Solving the vehicle routing problem by using cellular genetic algorithms. In *Proceedings of the 4th European Conference on Evolutionary Computation in Combinatorial Optimization, EvoCOP 2004*, pages 11–20, 2004. In proceedings [842]. Online available at http://www.lcc.uma.es/~eat/pdf/evocop04.pdf [accessed 2008-10-27].

[33] Enrique Alba and Bernabé Dorronsoro. *Cellular Genetic Algorithms*, volume 42 of *Operations Research/Computer Science Interfaces Series*. Springer-Verlag Gmbh, June 2008. ISBN: 0-3877-7609-5, 978-0-38777-609-5.

[34] Enrique Alba and Marco Tomassini. Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):443–462, October 2002. doi:10.1109/TEVC.2002.800880.

[35] Enrique Alba, C. Cotta, J. Francisco Chicano, and Antonio Jesús Nebro Urbaneja. Parallel evolutionary algorithms in telecommunications: Two case studies. In *Proceedings of the Argentinean Conference of Computer Science (CACIC'02)*, October 2002, Buenos Aires, Argentina. Online available at http://www.lcc.uma.es/~eat/./pdf/cacic02.pdf and http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.8.4073 [accessed 2008-08-01].

[36] Rudolf F. Albrecht, Colin R. Reeves, and Nigel C. Steele, editors. *Proceedings of Artificial Neural Nets and Genetic Algorithms, ANNGA*, April 1993, Innsbruck, Austria. Springer, New York. ISBN: 3-2118-2459-6, 0-3878-2459-6.

[37] John Aldrich. R.a. fisher and the making of maximum likelihood 1912-1922. *Statistical Science*, 3(12):162–176, January 1995. Online available at http://projecteuclid.org/euclid.ss/1030037906 [accessed 2007-09-15].

[38] M. Montaz Ali, Charoenchai Khompatraporn, and Zelda B. Zabinsky. A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems. *Journal of Global Optimization*, 31(4):635–672, April 2005. ISSN: 0925-5001 (Print) 1573-2916 (Online). doi:10.1007/s10898-004-9972-2. Online available at http://www.springerlink.com/content/q825578116kn6k72/fulltext.pdf [accessed 2008-07-23].

[39] Luc Onana Alima, Sameh El-Ansary, Per Brand, and Seif Haridi. $\mathcal{DKS}(\mathcal{N}, k, f)$: A family of low communication, scalable and fault-tolerant infrastructures for p2p applications. In *3rd IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2003)*, pages 344–350. IEEE Computer Society, 2003. ISBN: 0-7695-1919-9. Online available at http://dks.sics.se/pub/ccgrid-dks.pdf [accessed 2007-08-13].

[40] Jean-Marc Alliot, Evelyne Lutton, and Marc Schoenauer, editors. *European Conference on Artificial Evolution, AE 94*, 1994, Toulouse, France. Cepadues. ISBN: 978-2-85428-411-9. Published in January 1995.

[41] Jean-Marc Alliot, Evelyne Lutton, Edmund M. A. Ronald, Marc Schoenauer, and Dominique Snyers, editors. *Selected Papers of the European Conference on Artificial Evolution, AE 95*, volume 1063 of *Lecture Notes in Computer Science (LNCS)*, September 4–6, 1995, Brest, France. Springer Berlin/Heidelberg. ISBN: 3-5406-1108-8. Published in 1996.

[42] Lee Altenberg. The schema theorem and price's theorem. In *Foundations of Genetic Algorithms 3*, pages 23–49, 1994. In proceedings [2214]. Online available at http://dynamics.org/Altenberg/PAPERS/STPT/ and http://citeseer.ist.psu.edu/altenberg95schema.html [accessed 2008-07-20].

[43] Lee Altenberg. Genome growth and the evolution of the genotype-phenotype map. In *Evolution and Biocomputation – Computational Models of Evolution*, pages 205–259. Springer-Verlag, 1995. In collection [137]. Online available at http://citeseer.ist.psu.edu/254722.html [accessed 2007-07-29]: corrections and revisions, October 1997.

[44] Lee Altenberg. Nk fitness landscapes. In *Handbook of Evolutionary Computation*, chapter B2.7.2. Oxford University Press, November 27, 1996. In collection [104]. Online available at http://www.cmi.univ-mrs.fr/~pardoux/LeeNKFL.pdf and http://dynamics.org/Altenberg/FILES/LeeNKFL.pdf [accessed 2008-10-06].

[45] Lee Altenberg. Fitness distance correlation analysis: An instructive counterexample. In *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA97)*, pages 57–64, 1997. In proceedings [98]. Online available at http://dynamics.org/Altenberg/PAPERS/FDCAAIC/ and http://www.cs.uml.edu/~giam/91.510/Papers/Altenberg1997.pdf [accessed 2008-07-20].

[46] Guilherme Bastos Alvarenga and Geraldo Robson Mateus. Hierarchical tournament selection genetic algorithm for the vehicle routing problem with time windows. In *4th International Conference on Hybrid Intelligent Systems*, pages 410–415, 2004. doi:10.1109/ICHIS.2004.53. In proceedings [991].

[47] Anita Amberg, Wolfgang Domschke, and Stefan Voß. Multiple center capacitated arc routing problems: A tabu search algorithm using capacitated trees. *European Journal of Operational Research (EJOR)*, 124(2):360–376, July 16, 2000. ISSN: 0377-2217. doi:10.1016/S0377-2217(99)00170-8. Online available at http://dx.doi.org/10.1016/S0377-2217(99)00170-8 [accessed 2008-10-27].

[48] Ciro Amitrano, Luca Peliti, and M. Saber. Population dynamics in a spin-glass model of chemical evolution. *Journal of Molecular Evolution*, 29(6):513–525, December 1989. ISSN: 0022-2844 (Print) 1432-1432 (Online). doi:10.1007/BF02602923. Online available at http://www.springerlink.com/content/n172u715884w0632/fulltext.pdf [accessed 2008-07-02].

[49] Heni Ben Amor and Achim Rettinger. Intelligent exploration for genetic algorithms: Using self-organizing maps in evolutionary computation. In *GECCO'05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1531–1538, 2005. doi:10.1145/1068009.1068250. In proceedings [202]. Online available at http://doi.acm.org/10.1145/1068009.1068250 [accessed 2007-08-29].

[50] David G. Andersen, Hari Balakrishnan, Frans Kaashoek, and Robert Morris. Resilient overlay networks. In *18th ACM SOSP*, October 2001. Banff, Canada. Online available at http://nms.csail.mit.edu/ron/ and http://nms.lcs.mit.edu/papers/ron-sosp2001.pdf [accessed 2007-08-13].

[51] Philip Warren Anderson. Suggested model for prebiotic evolution: The use of chaos. *Proceedings of the National Academy of Science of the United States of America (PNAS)*, 80(11):3386–3390, June 1, 1983. Online available at http://www.pnas.org/cgi/reprint/80/11/3386.pdf and http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=394048 [accessed 2008-07-02].

[52] David Andre. Evolution of mapmaking ability: Strategies for the evolution of learning, planning, and memory using genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, pages 250–255, 1994. In proceedings [1411].

[53] David Andre. Learning and upgrading rules for an OCR system using genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, 1994. In proceedings [1411]. Online available at http://citeseer.ist.psu.edu/31976.html and http://citeseer.ist.psu.edu/cache/papers/cs/802/http:zSzzSzwww.cs.berkeley.eduzSz~dandrezSzpaperszSzAndre_WCCI_94_OCR_Boundary.pdf/learning-and-upgrading-rules.pdf [accessed 2007-10-03].

[54] David Andre. The automatic programming of agents that learn mental models and create simple plans of action. In *IJCAI-95 Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, volume 1, pages 741–747, 1995. MAPMAKER searches for gold. Online available at http://citeseer.ist.psu.

edu/188996.html and http://dli.iiit.ac.in/ijcai/IJCAI-95-VOL%201/pdf/097.pdf [accessed 2007-10-03]. In proceedings [1453].

[55] David Andre. The evolution of agents that build mental models and create simple plans using genetic programming. In *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pages 248–255, 1995. ISBN: 1-5586-0370-0. In proceedings [636].

[56] David Andre. Learning and upgrading rules for an optical character recognition system using genetic programming. In *Handbook of Evolutionary Computation*. Oxford University Press, 1997. ISBN: 0-7503-0392-1. In collection [104].

[57] David Andre and Astro Teller. Evolving team darwin united. In Minoru Asada and Hiroaki Kitano, editors, *RoboCup-98: Robot Soccer World Cup II*, volume 1604 of *Lecture Notes in Artificial Intelligence, subseries of Lecture Notes in Computer Science (LNCS)*, pages 346–351. Springer Verlag, July 1998, Paris, France. ISBN: 3-5406-6320-7. Published in 1999. Online available at http://www.cs.cmu.edu/afs/cs/usr/astro/public/papers/Teller_Astro.ps and http://citeseer.ist.psu.edu/322830.html [accessed 2007-10-03].

[58] David Andre, Forrest H. Bennett III, and John R. Koza. Discovery by genetic programming of a cellular automata rule that is better than any known rule for the majority classification problem. In *Proceedings of the First Annual Conference Genetic Programming (GP-96)*, pages 3–11, 1996. In proceedings [1207] and also [59]. Online available at http://citeseer.ist.psu.edu/33008.html [accessed 2007-08-01].

[59] David Andre, Forrest H. Bennett III, and John R. Koza. Evolution of intricate long-distance communication signals in cellular automata using genetic programming. In *Artificial Life V: Proceedings of the Fifth International Workshop on the Synthesis and Simulation of Living Systems*, volume 1. MIT Press, Cambridge, MA, USA, May 16–18, 1996, Nara, Japan. See also [58]. Online available at http://www.genetic-programming.com/jkpdf/alife1996gkl.pdf and http://citeseer.ist.psu.edu/andre96evolution.html [accessed 2007-10-03].

[60] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.*, 36(4):335–371, 2004. ISSN: 0360-0300. Online available at http://www.spinellis.gr/pubs/jrnl/2004-ACMCS-p2p/html/AS04.pdf and http://portal.acm.org/citation.cfm?id=1041681 [accessed 2007-08-13].

[61] Peter J. Angeline and Kenneth E. Kinnear, Jr, editors. *Advances in Genetic Programming*, volume 2 of *Complex Adaptive Systems*. MIT Press, Cambridge, MA, USA, October 26, 1996. ISBN: 0-2620-1158-1, 978-0-26201-158-7.

[62] Peter John Angeline. Subtree crossover: Building block engine or macromutation? In *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 9–17, 1997. In proceedings [1208].

[63] Peter John Angeline. A historical perspective on the evolution of executable structures. *Fundamenta Informaticae*, 35(1–4):179–195, August 1998. ISSN: 0169-2968. Special volume: Evolutionary Computation. Online available at http://citeseer.ist.psu.edu/angeline98historical.html and http://www.natural-selection.com/Library/1998/gphist.pdf [accessed 2008-05-29]. See also [63].

[64] Peter John Angeline. Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences. In *EP'98: Proceedings of the 7th International Conference on Evolutionary Programming VII*, pages 601–610, 1998. In proceedings [1670].

[65] Peter John Angeline. Subtree crossover causes bloat. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 745–752. Morgan Kaufmann, 1998. In proceedings [1209].

[66] Peter John Angeline and Jordan Pollack. Evolutionary module acquisition. In *Proceedings of the Second Annual Conference on Evolutionary Programming*, pages 154–

163, 1993. In proceedings [702]. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.57.3280 and https://eprints.kfupm.edu.sa/38410/ [accessed 2008-10-18] and online available at http://www.demo.cs.brandeis.edu/papers/ep93.pdf and http://www.natural-selection.com/Library/1993/ep93.ps.Z [accessed 2007-11-03].

[67] Peter John Angeline and Jordan B. Pollack. Coevolving high-level representations. In *Artificial Life III*, pages 55–71, 1992. Online available at http://demo.cs.brandeis.edu/papers/alife3.pdf and http://citeseer.ist.psu.edu/angeline94coevolving.html [accessed 2007-10-05]. In proceedings [1247].

[68] Peter John Angeline, Robert G. Reynolds, John Robert McDonnell, and Russel C. Eberhart, editors. *Proceedings of the 6th International Conference on Evolutionary Programming*, volume 1213/1997 of *Lecture Notes in Computer Science (LNCS)*, 1997, Indianapolis, Indiana, USA. Springer Verlag Telos. ISBN: 978-3-54062-788-3.

[69] Peter John Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, editors. *Proceedings of the IEEE Congress on Evolutionary Computation, CEC99*, volume 1-3, July 6–9, 1999, Mayflower Hotel, Washington D.C., USA. IEEE Press, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA. ISBN: 0-7803-5536-9, 0-7803-5537-7. Library of Congress Control Number: 99-61143. CEC-99 – A joint meeting of the IEEE, Evolutionary Programming Society, Galesia, and the IEE.

[70] S. Anily and Awi Federgruen. Ergodicity in parametric non-stationary markov chains: An application to simulated annealing methods. *Operations Research*, (35):867–874, 1987.

[71] Anupriya Ankolekar, Mark Burstein, Grit Denker, Daniel Elenius, Jerry Hobbs, Lalana Kagal, Ora Lassila, Drew McDermott, Deborah McGuinness, Sheila McIlraith, Massimo Paolucci, Bijan Parsia, Terry Payne, Marta Sabou, Craig Schlenoff, Evren Sirin, Monika Solanki, Naveen Srinivasan, Katia Sycara, and Randy Washington. *OWL-S 1.1 Release, OWL-based Web Service Ontology.* Web-Ontology Working Group at the World Wide Web Consortium, 2004. Online available at http://www.daml.org/services/owl-s/1.1/ [accessed 2007-09-02].

[72] G. Degli Antoni, D. Cabianca, M. Vaccari, M. Benini, and F. Casablanca. Linearity of client/server systems. *Bulletin of the European Association for Theoretical Computer Science*, 57:201–214, April 1995. Online available at http://citeseer.ist.psu.edu/antoni95linearity.html and http://en.scientificcommons.org/130995 [accessed 2007-08-13].

[73] Hendrik James Antonisse. A grammar-based genetic algorithm. In *Proceedings of Foundations of Genetic Algorithms FOGA-90*, pages 193–204, 1990. In proceedings [1924].

[74] Hendrik James Antonisse. Unsupervised credit assignment in knowledge-based sensor fusionsystems. *IEEE Transactions on Systems, Man and Cybernetics*, 20(5):1153–1171, September/October 1990. doi:10.1109/21.59978.

[75] Bruno Apolloni, Robert J. Howlett, and Lakhmi C. Jain, editors. *Proceedings of 11th International Conference on the Knowledge-Based Intelligent Information and Engineering Systems, KES 2007, and the XVII Italian Workshop on Neural Networks, Part I*, volume 4692 of *Lecture Notes in Computer Science (LNCS)*, September 12–14, 2007, Vietri sul Mare, Italy. Springer. ISBN: 978-3-54074-817-5. See also [76, 77].

[76] Bruno Apolloni, Robert J. Howlett, and Lakhmi C. Jain, editors. *Proceedings of 11th International Conference on the Knowledge-Based Intelligent Information and Engineering Systems, KES 2007, and the XVII Italian Workshop on Neural Networks, Part II*, volume 4693 of *Lecture Notes in Computer Science (LNCS)*, September 12–14, 2007, Vietri sul Mare, Italy. Springer. ISBN: 978-3-54074-826-7. See also [75, 77].

[77] Bruno Apolloni, Robert J. Howlett, and Lakhmi C. Jain, editors. *Proceedings of 11th International Conference on the Knowledge-Based Intelligent Information and*

Engineering Systems, KES 2007, and the XVII Italian Workshop on Neural Networks, Part III, volume 4694 of *Lecture Notes in Computer Science (LNCS)*, September 12–14, 2007, Vietri sul Mare, Italy. Springer. ISBN: 978-3-54074-828-1. See also [75, 76].

[78] David L. Applegate, Robert E. Bixby, Vašek Chvátal, and William J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton Series in Applied Mathematics. Princeton University Press, 41 William Street, Princeton, New Jersey, USA, 2006. ISBN: 978-0-69112-993-8, 0-6911-2993-2.

[79] *4th International Conference on Hybrid Artificial Intelligence Systems (HAIS'09)*, Lecture Notes in Artificial Intelligence (LNAI), subseries of Lecture Notes in Computer Science (LNCS), June 10–12, 2009, Salamanca, Spain. Applied Computational Intelligence Group (GICAP), University of Burgos, Springer. co-located with 10th International Work-Conference on Artificial Neural Networks (IWANN2009).

[80] Hamid R. Arabnia, Jack Y. Yang, and Mary Qu Yang, editors. *Proceedings of the 2007 International Conference on Genetic and Evolutionary Methods, GEM 2007*, June 25–28 2007, Las Vegas Nevada, USA. CSREA Press. ISBN: 1-6013-2038-8.

[81] Hamid R. Arabnia, Jack Y. Yang, and Mary Qu Yang, editors. *Proceedings of the 2008 International Conference on Genetic and Evolutionary Methods, GEM 2008*, June 14–17 2008, Monte Carlo Resort, Las Vegas Nevada, USA.

[82] Victoria S. Aragón and Susana C. Esquivel. An evolutionary algorithm to track changes of optimum value locations in dynamic environments. *Journal of Computer Science & Technology (JCS&T)*, 4(3)(12):127–133, October 2004. ISSN: 1666-6038. Invited paper. Online available at http://journal.info.unlp.edu.ar/Journal/journal12/papers.html [accessed 2007-08-19].

[83] Maribel G. Arenas, Pierre Collet, Ágoston E. Eiben, Márk Jelasity, Juan J. Merelo, Ben Paechter, Mike Preuß, and Marc Schoenauer. A framework for distributed evolutionary algorithms. In *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature – PPSN VII*, pages 665–675, 2002. In proceedings [867]. Online available at http://citeseer.ist.psu.edu/arenas02framework.html [accessed 2008-04-06].

[84] S. Asharafa and M. Narasimha Murtyb. An adaptive rough fuzzy single pass algorithm for clustering large data sets. *Pattern recognition (Pattern recogn.)*, 36(12): 3015–3018, 2003. ISSN: 0031-3203.

[85] Daniel Ashlock. *Evolutionary Computation for Modeling and Optimization*. Springer, Berlin, Germany, December 1, 2005. ISBN: 0-3872-2196-4, 978-0-38722-196-0. Partly online available at http://books.google.de/books?id=74Q5WBOaoz8C [accessed 2008-04-03].

[86] Mikhail J. Atallah and Susan Fox, editors. *Algorithms and Theory of Computation Handbook*. CRC Press, Inc., Boca Raton, FL, USA, first edition, November 1998. ISBN: 978-0-84932-649-3. Produced By Suzanne Lassandro.

[87] Douglas A. Augusto and Helio J. C. Barbosa. Symbolic regression via genetic programming. In *Proceedings of Sixth Brazilian Symposium on Neural Networks (SBRN'00)*, pages 173–178, 2000. IEEE Computer Society, Los Alamitos, CA, USA. ISBN: 0-7695-0856-1.

[88] Ran Avnimelech and Nathan Intrator. Boosting regression estimators. *Neural Computation*, 11(2):499–520, 1999. Online available at http://neco.mitpress.org/cgi/reprint/11/2/499.pdf and http://citeseer.ist.psu.edu/avnimelech99boosting.html [accessed 2007-9-15].

[89] Mordecai Avriel. *Nonlinear Programming: Analysis and Methods*. Dover Publishing Inc., September 16, 2003. ISBN: 0-4864-3227-0, 978-0-48643-227-4. Partly online available at http://books.google.de/books?id=byF4Xb1QbvMC [accessed 2008-06-14].

B

[90] Sara Baase and Allen Van Gelder. *Computer Algorithms: Introduction to Design and Analysis*. Addison Wesley, third edition, November 1999. ISBN: 978-0-20161-244-8.

[91] N. Baba. Convergence of a random optimization method for constrained optimization problems. *Journal of Optimization Theory and Applications*, 33(4):451–461, April 1981. ISSN: 0022-3239 (Print) 1573-2878 (Online). Communicated by N. Avriel, Online available at http://www.springerlink.com/content/q1g0k627688684k3/ [accessed 2007-08-26].

[92] Jaume Bacardit i Peñarroya. *Pittsburgh Genetics-Based Machine Learning in the Data Mining era: Representations, generalization, and run-time*. PhD thesis, Computer Science Department, Enginyeria i Architectura La Salle, Ramon Llull University, Barcelona, Catalonia, Spain, October 8, 2004. Advisor: Josep Maria Garrell i Guiu. Online available at http://www.cs.nott.ac.uk/~jqb/publications/thesis.pdf [accessed 2007-09-12].

[93] Jaume Bacardit i Peñarroya. Analysis of the initialization stage of a pittsburgh approach learning classifier system. In *GECCO'05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1843–1850, 2005. In proceedings [202]. Online available at http://doi.acm.org/10.1145/1068009.1068321 and http://www.cs.nott.ac.uk/~jqb/publications/gecco2005.pdf [accessed 2007-09-12].

[94] Jaume Bacardit i Peñarroya and Natalio Krasnogor. Smart crossover operator with multiple parents for a pittsburgh learning classifier system. In *GECCO'06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1441–1448, 2006. In proceedings [352]. Online available at http://doi.acm.org/10.1145/1143997.1144235 and http://www.cs.nott.ac.uk/~jqb/publications/gecco2006-sx.pdf [accessed 2007-09-12].

[95] Vincent Bachelet and El-Ghazali Talbi. A parallel co-evolutionary metaheuristic. In *IPDPS'00: Proceedings of the 15 IPDPS 2000 Workshops on Parallel and Distributed Processing*, pages 628–635, 2000. Springer-Verlag, London, UK. ISBN: 3-5406-7442-X. Online available at http://ipdps.cc.gatech.edu/2000/biosp3/18000629.pdf and http://citeseer.ist.psu.edu/bachelet00parallel.html [accessed 2007-09-10].

[96] Paul Gustav Heinrich Bachmann. *Die Analytische Zahlentheorie / Dargestellt von Paul Bachmann*, volume Zweiter Theil of *Zahlentheorie: Versuch einer Gesamtdarstellung dieser Wissenschaft in ihren Haupttheilen*. B. G. Teubner, Leipzig, Germany, 1894. ISBN: 1-4181-8540-X, 978-1-41818-540-4. Online available at http://gallica.bnf.fr/ark:/12148/bpt6k994750 [accessed 2008-05-20]. Republished by Scholarly Publishing Office, University of Michigan Library, 2006.

[97] Thomas Bäck. Generalized convergence models for tournament- and (μ, λ)-selection. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 2–8, 1995. In proceedings [636]. Online available at http://citeseer.ist.psu.edu/1463.html [accessed 2008-07-21].

[98] Thomas Bäck, editor. *Proceedings of The Seventh International Conference on Genetic Algorithms, ICGA'97*, July 19–23, 1997, Michigan State University, East Lansing, Michigan, USA. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN: 1-5586-0487-1. see http://garage.cse.msu.edu/icga97/ [accessed 2007-09-01].

[99] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press US, January 1996. ISBN: 0-1950-9971-0, 978-0-19509-971-3. Partly online available at http://books.google.de/books?id=EaN7kvl5coYC [accessed 2009-02-03].

[100] Thomas Bäck and Ulrich Hammel. Evolution strategies applied to perturbed objective functions. In *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, volume 1, pages 40–

45, 1994. doi:10.1109/ICEC.1994.350045. In proceedings [1411]. Online available at http://citeseer.ist.psu.edu/16973.html [accessed 2008-07-19].

[101] Thomas Bäck and Martin Schütz. Evolution strategies for mixed-integer optimization of optical multilayer systems. In *Evolutionary Programming IV: Proceedings of the Fourth Annual Conference on Evolutionary Programming*, pages 33–51, 1995. In proceedings [1380]. Online available at http://citeseer.ist.psu.edu/101945.html and http://de.scientificcommons.org/114486 [accessed 2007-08-27].

[102] Thomas Bäck and Hans-Paul Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1–23, Spring 1993. ISSN: 1063-6560.

[103] Thomas Bäck, Frank Hoffmeister, and Hans-Paul Schwefel. A survey of evolution strategies. In *Proceedings of the 4th International Conference on Genetic Algorithms ICGA'91*, pages 2–9, 1991. In proceedings [170]. Online available at http://de.scientificcommons.org/50038 and http://citeseer.ist.psu.edu/19412.html [accessed 2007-08-27].

[104] Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors. *Handbook of Evolutionary Computation*. Computational Intelligence Library. Oxford University Press in cooperation with the Institute of Physics Publishing / CRC Press, Bristol, New York, ringbound edition, April 1997. ISBN: 0-7503-0392-1, 978-0-75030-392-7, 0-7503-0895-8, 978-0-75030-895-3. Partly online available at http://books.google.de/books?id=n5nuiIZvmpAC [accessed 2008-11-15].

[105] Thomas Bäck, Ulrich Hammel, and Hans-Paul Schwefel. Evolutionary computation: comments on the history and current state. *IEEE Transactions on Evolutionary Computation*, 1(1):3–17, April 1997. Online available at http://sci2s.ugr.es/docencia/doctobio/EC-History-IEEETEC-1-1-1997.pdf and http://citeseer.ist.psu.edu/601414.html [accessed 2007-08-24].

[106] Thomas Bäck, Zbigniew Michalewicz, and Xin Yao, editors. *IEEE International Conference on Evolutionary Computation, CEC97*, April 13–16, 1997, Indianapolis, IN, USA. IEEE Press, Piscataway, NJ. ISBN: 0-7803-3949-5. See http://ieeexplore.ieee.org/servlet/opac?punumber=4619 [accessed 2007-09-06].

[107] Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors. *Evolutionary Computation 1: Basic Algorithms and Operators*. Institute of Physics Publishing (IOP), Dirac House, Temple Back, Bristol BS1 6BE, United Kingdom, January 2000. ISBN: 978-0-75030-664-5, 0-7503-0664-5. Partly online available at http://books.google.de/books?id=4HMYCq9US78C [accessed 2008-06-11].

[108] Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz, editors. *Evolutionary Computation 2: Advanced Algorithms and Operators*. Institute of Physics Publishing (IOP), Dirac House, Temple Back, Bristol BS1 6BE, United Kingdom, November 2000. ISBN: 978-0-75030-665-2, 0-7503-0665-3.

[109] J. W. Backus, J. H. Wegstein, A. van Wijngaarden, M. Woodger, F. L. Bauer, J. Green, C. Katz, J. McCarthy, A. J. Perlis, H. Rutishauser, K. Samelson, and B. Vauquois. Report on the algorithmic language algol 60. *Communications of the ACM*, 3(5):299–314, May 1960. ISSN: 0001-0782. See also http://www.masswerk.at/algol60/ [accessed 2007-09-15] and [110].

[110] J. W. Backus, F. L. Bauer, J. Green, C. Katz, J. McCarthy, A. J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, J. H. Wegstein, A. van Wijngaarden, and M. Woodger. Revised report on the algorithm language algol 60. *Communications of the ACM*, 6(1):1–17, 1963. ISSN: 0001-0782. doi:10.1145/366193.366201. Online available at http://doi.acm.org/10.1145/366193.366201 and http://www.masswerk.at/algol60/report.htm [accessed 2007-09-15]. See also [109].

[111] Liviu Badea and Monica Stanciu. Refinement operators can be (weakly) perfect. In *Inductive Logic Programming, Proceedings of 9th International Workshop, ILP-99 Bled, Slovenia*, volume 1634/1999 of *Lecture Notes in Computer Science (LNCS)*,

pages 21–32. Springer Berlin/Heidelberg, June 24–27, 1999. ISBN: 978-3-54066-109-2. doi:10.1007/3-540-48751-4_4. Online available at http://www.ai.ici.ro/papers/ilp99.ps.gz [accessed 2008-02-22].

[112] P. Badeau, Michel Gendreau, F. Guertin, Jean-Yves Potvin, and Éric D. Taillard. A parallel tabu search heuristic for the vehicle routing problem with time windows. *Transportation Research Part C: Emerging Technologies*, 5(2):109–122, April 1997. ISSN: 0968-090X.

[113] Adedeji Bodunde Badiru. *Handbook of Industrial and Systems Engineering*. CRC Press Inc, January 2006. ISBN: 0-8493-2719-9, 978-0-84932-719-3. Partly online available at http://books.google.de/books?id=4FkmK2A6wuUC [accessed 2008-03-27].

[114] Philipp Andreas Baer and Roland Reichle. Communication and collaboration in heterogeneous teams of soccer robots. In *Robotic Soccer*, chapter 1, pages 1–28. I-Tech Education and Publishing, Vienna, Austria, 2007. In collection [1286]. Online available at http://s.i-techonline.com/Book/Robotic-Soccer/ISBN978-3-902613-21-9-rs01.pdf [accessed 2008-04-23].

[115] Philipp Andreas Baer, Roland Reichle, Michael Zapf, Thomas Weise, and Kurt Geihs. A Generative Approach to the Development of Autonomous Robot Software. In *Proceedings of 4th IEEE Workshop on Engineering of Autonomic and Autonomous Systems (EASe 2007)*. IEEE, March 26-29, 2007, Tucson, AZ, U.S.A. ISBN: 0-7695-2809-0. doi:10.1109/EASE.2007.2. Online available at http://www.it-weise.de/documents/files/BRZWG2007AR.pdf [accessed 2009-06-26].

[116] John Daniel Bagley. *The Behavior of Adaptive Systems which employ Genetic and Correlation Algorithms*. PhD thesis, The University of Michigan, College of Literature, Science, and the Arts, Computer and Communication Sciences Department, Ann Arbor, MI, USA, December 1967. Order No. AAI6807556. Published as Technical Report and in Dissertations International 28(12), 5106B, University Microfilms No. 68-7556. Online available at http://hdl.handle.net/2027.42/3354 [accessed 2007-10-31]. ID: bab0779.0001.001.

[117] Javier Bajo, Emilio S. Corchado, Álvaro Herrero, and Juan M. Corchado, editors. *Proceedings of the 1st International Workshop on Hybrid Artificial Intelligence Systems (HAIS 2006)*, October 27, 2006, Ribeirão Preto, SP, Brazil. University of Salamanca, Salamanca, Spain. ISBN: 8-4934-1819-6.

[118] Per Bak and Kim Sneppen. Punctuated equilibrium and criticality in a simple model of evolution. *Physical Review Letters*, 71(24):4083–4086, December 13, 1993. doi:10.1103/PhysRevLett.71.4083. Online available at http://prola.aps.org/abstract/PRL/v71/i24/p4083_1 [accessed 2008-08-24].

[119] Per Bak, Chao Tang, and Kurt Wiesenfeld. Self-organized criticality. *Physical Review A*, 38(1):364–374, July 1, 1988. doi:10.1103/PhysRevA.38.364. Online available at http://prola.aps.org/abstract/PRA/v38/i1/p364_1 [accessed 2008-08-23].

[120] James E. Baker. Adaptive selection methods for genetic algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 101–111, 1985. In proceedings [856].

[121] James E. Baker. Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the second International Conference on Genetic Algorithms*, pages 14–21, 1987. In proceedings [857].

[122] Henri Elle Bal. *Programming Distributed Systems*. Silicon Press/Prentice Hall International, 1990/1991. ISBN: 0-9293-0605-8, 978-0-92930-605-6, 0-1372-2083-9. Revision of the author's Ph.D. thesis. Partly online available at http://books.google.de/books?id=LMg_ZL7CLAAC [accessed 2008-11-09].

[123] James Mark Baldwin. A new factor in evolution. *The American Naturalist*, 30: 441–451, June 1896. ISSN: 0003-0147. Online available at http://members.aol.com/jorolat/baldwin2.html and http://www.brocku.ca/MeadProject/Baldwin/Baldwin_1896_h.html [accessed 2008-09-10]. See also [124].

[124] James Mark Baldwin. A new factor in evolution. In *Adaptive individuals in evolving populations: models and algorithms*, pages 59–80. Addison-Wesley Longman Publishing Co., Inc., 1996. In collection [171]. See also [123].

[125] Alexandre M. Baltar and Darrell G. Fontane. A generalized multiobjective particle swarm optimization solver for spreadsheet models: application to water quality. In *Proceedings of AGU Hydrology Days 2006*, pages 1–12, March 20–22, 2006, Fort Collins, Colorado. Online available at http://www.lania.mx/~ccoello/EMOO/baltar06.pdf.gz and http://hydrologydays.colostate.edu/Proceedings_2006.htm [accessed 2007-08-21].

[126] Robert Balzer, editor. *Proceedings of the 1st Annual National Conference on Artificial Intelligence, AAAI*, August 18–21, 1980, Stanford University, California, USA. AAAI Press/MIT Press. ISBN: 0-2625-1050-2. See http://www.aaai.org/Conferences/AAAI/aaai80.php [accessed 2007-09-06].

[127] Jean-Pierre Banâre, Pascal Fradet, and Yann Radenac. Chemical specification of autonomic systems. In *Proceedings of the ISCA 13th International Conference on Intelligent and Adaptive Systems and Software Engineering (IASSE)*, pages 72–79, July 1–3, 2004, Nice, France. Online available at http://hal.inria.fr/docs/00/05/22/05/PDF/IASSE04.pdf and http://download.scientificcommons.org/140384 [accessed 2009-02-21].

[128] Jean-Pierre Banâtre and Daniel Le Métayer. A new computational model and its discipline of programming. Rapports de Recherche 556, INRIA (Institut National de Recherche en Informatique et en Automatique) Centre de Rennes, IRISA (Institut de Recherche en Informatique et Systémes Aléatoires), Domaine de Voluceau Rocquencourt BP105, 78153 Le Chasney Cédex, France / Campus Universitaire de Beaulieu, Avenue du Général Leclerc, 35042 Rennes Cédex, France, September 1986. Online available at http://hal.inria.fr/docs/00/07/59/88/PDF/RR-0566.pdf [accessed 2009-02-21].

[129] Jean-Pierre Banâtre and Daniel Le Métayer. Gamma and the chemical reaction model. Publication Interne 984, INRISA (Institut de Recherche en Informatique et Systémes Aléatoires), Campus Universitaire de Beaulieu, Avenue du Général Leclerc, 35042 Rennes Cédex, France, February 1996. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.38.5802 [accessed 2009-02-21].

[130] Jean-Pierre Banâtrea, Pascal Fradet, and Yann Radenac. Principles of chemical programming. In Slim Abdennadher and Christoph Ringeissen, editors, *Proceedings of the Fifth International Workshop on Rule-Based Programming (RULE 2004)*, June 1, 2004, Aachen, Germany. Online available at http://www-i2.informatik.rwth-aachen.de/RULE04/RULE-proceedings.ps and http://pop-art.inrialpes.fr/~fradet/PDFs/RULE04.pdf [accessed 2009-02-21]. Proceedings appeared in Aachener Informatik Berichte, ISSN 0935-3232, AIB-2004-06, RWTH Aachen, Department of Computer Science, June 2004. See also [131].

[131] Jean-Pierre Banâtrea, Pascal Fradet, and Yann Radenac. Principles of chemical programming. *Electronic Notes in Theoretical Computer Science*, 124(1):133–147, March 2, 2005. ISSN: 1571-0661. doi:doi:10.1016/j.entcs.2004.07.019. Online available at http://dx.doi.org/10.1016/j.entcs.2004.07.019 [accessed 2009-02-21]. See also [130].

[132] Saswatee Banerjee and Lakshminarayan N. Hazra. Thin lens design of cooke triplet lenses: application of a global optimization technique. In Kevin D. Bell, Michael K. Powers, and Jose M. Sasian, editors, *Proceedings of SPIE, Society of Photo-Optical Instrumentation Engineers (SPIE) Conference – Novel Optical Systems and Large-Aperture Imaging*, volume 3430, pages 175–183, December 1998. doi:10.1117/12.332474.

[133] *Classification, Clustering, and Data Mining Applications: Proceedings of the Meeting of the International Federation of Classification Societies (IFCS), ... Data Analysis,*

and Knowledge Organization), 2004. Springer-Verlag New York, Inc., Secaucus, NJ, USA. ISBN: 3-5402-2014-3.

[134] Jerry Banks, editor. *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice*. Wiley & Sons and Engineering & Management Press, A division of the Institute of Industrial Engineers, New York, USA, September 29, 1998. ISBN: 0-4711-3403-1, 978-0-47113-403-9. Partly online available at http://books.google.de/books?id=dMZ1Zj3TBgAC [accessed 2008-10-14].

[135] Wolfgang Banzhaf. Genetic programming for pedestrians. In *Proceedings of the 5th International Conference on Genetic Algorithms, ICGA-93*, pages 17–21, 1993. In proceedings [730]. Also: MRL Technical Report 93-03. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.38.1144 and http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/banzhaf_mrl.html [accessed 2008-09-17].

[136] Wolfgang Banzhaf. Genotype-phenotype-mapping and neutral variation – A case study in genetic programming. In *Parallel Problem Solving from Nature III*, pages 322–332, 1994. In proceedings [492]. Online available at http://citeseer.ist.psu.edu/banzhaf94genotypephenotypemapping.html [accessed 2007-09-09].

[137] Wolfgang Banzhaf and Frank H. Eeckman, editors. *Evolution and Biocomputation – Computational Models of Evolution*, volume 899/1995 of *Lecture Notes in Computer Science (LNCS)*. Springer-Verlag, London, UK, kindle edition, April 13 1995, Monterey, California, USA. ISBN: 3-5405-9046-3, 0-3875-9046-3, 978-3-54059-046-0. doi:10.1007/3-540-59046-3. ASIN: B000V1DAP0. Partly online available at http://books.google.de/books?id=dZf-yrmSZPkC [accessed 2008-04-30].

[138] Wolfgang Banzhaf and Christian W. G. Lasarczyk. Genetic programming of an algorithmic chemistry. In *Genetic Programming Theory and Practice II*, pages 175–190, 2005. doi:10.1007/0-387-23254-0_11. In proceedings [1583]. Online available at http://www.cs.mun.ca/~banzhaf/papers/algochem.pdf and http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/banzhaf_2004_GPTP.html [accessed 2008-04-30].

[139] Wolfgang Banzhaf and Colin R. Reeves, editors. *Proceedings of the Fifth Workshop on Foundations of Genetic Algorithms (FOGA)*, July 22–25, 1998, Madison, WI, USA. Morgan Kaufmann, San Mateo, CA, USA. ISBN: 1-5586-0559-2. Published April 2, 1991.

[140] Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, and Frank D. Francone. *Genetic Programming: An Introduction – On the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann Publishers / dpunkt.verlag, San Francisco, CA, USA / Heidelberg, Germany, November 30, 1997. ISBN: 978-1-55860-510-7, 1-5586-0510-X, 3-9209-9358-6. Partly online available at http://books.google.de/books?id=1697qefFdtIC [accessed 2008-09-16].

[141] Wolfgang Banzhaf, Riccardo Poli, Marc Schoenauer, and Terence C. Fogarty, editors. *Proceedings of the First European Workshop on Genetic Programming, EuroGP'98*, volume 1391/1998 of *Lecture Notes in Computer Science (LNCS)*, April 14-15, 1998, Paris, France. Springer, Berlin/Heidelberg, Germany. ISBN: 3-5406-4360-5. doi:10.1007/BFb0055923, 978-3-540-64360-9. See alse [1663].

[142] Wolfgang Banzhaf, Jason M. Daida, Ágoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark J. Jakiela, and Robert E. Smith, editors. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 1999)*, July 13–17, 1999, Orlando, Florida, USA. Morgan Kaufmann. ISBN: 1-5586-0611-4. See also [1584, 1889].

[143] Wolfgang Banzhaf, Thomas Christaller, Peter W. G. Dittrich, Jan T. Kim, and Jens Ziegler, editors. *Advances in Artificial Life, Proceedings of the 7th European Conference, ECAL 2003*, volume 2801 of *Lecture Notes in Computer Science (LNCS)*, September 14–17, 2003, Dortmund, Germany. Springer. ISBN: 3-5402-0057-6.

[144] Wolfgang Banzhaf, Guillaume Beslon, Steffen Christensen, James A. Foster, François Képès, Virginie Lefort, Julian Francis Miller, Miroslav Radman, and Jeremy J. Rams-

den. From artificial evolution to computational evolution: a research agenda. *Nature Reviews Genetics*, 7(9):729–735, September 2006. ISSN: 1471-0056, 1471-0064. doi:10.1038/nrg1921. Guidelines/Perspectives. Online available at http://www.cs.mun.ca/~banzhaf/papers/nrg2006.pdf [accessed 2008-07-20].

[145] Helio Joseé Correa Barbosa, Fernanda M. P. Raupp, Carlile Campos Lavor, H. Lima, and Nelson Maculan. A hybrid genetic algorithm for finding stable conformations of small molecules. In Felipe M. G. França and Carlos H. C. Ribeiro, editors, *Proceedings of the VI Brazilian Symposium on Neural Networks (SBRN 2000)*, pages 90–94, November 22–25, 2000, Rio de Janiero, RJ, Brazil. IEEE Computer Society, Washington, DC, USA. ISBN: 0-7695-0856-1. doi:10.1109/SBRN.2000.889719.

[146] Valmir C. Barbosa. *An Introduction to Distributed Algorithms*. The MIT Press, 55 Hayward Street, Cambridge, MA 02142, USA, September 1996. ISBN: 0-2620-2412-8, 978-0-26202-412-9. Partly online available at http://books.google.de/books?id=fPaGrO8KFqQC [accessed 2008-11-25].

[147] *Proceedings of the EUROGEN2003 Conference: Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems*, September 15–17, 2003, Barcelona, Spain. ISBN: 8-4959-9933-1. Published on CD. See http://congress.cimne.upc.es/eurogen03/ [accessed 2007-09-16].

[148] Lionel Barnett. Tangled webs: Evolutionary dynamics on fitness landscapes with neutrality. Master's thesis, School of Cognitive Science, University of East Sussex, Brighton, UK, 1997. Supervisor: Inman Harvey. Online available at http://citeseer.ist.psu.edu/barnett97tangled.html and ftp://ftp.cogs.susx.ac.uk/pub/users/inmanh/lionelb/ [accessed 2007-08-13].

[149] Lionel Barnett. Ruggedness and neutrality – the nkp family of fitness landscapes. In *Artificial Life VI: Proceedings of the sixth international conference on Artificial life*, pages 18–27, 1998. In proceedings [13]. Online available at ftp://ftp.informatics.sussex.ac.uk/pub/users/lionelb/publications/alife6_paper.pdf and http://citeseer.ist.psu.edu/barnett98ruggedness.html [accessed 2008-06-01], online available at http://citeseer.ist.psu.edu/144047.html [accessed 2008-02-27].

[150] Nils Aaall Barricelli. Esempi numerici di processi di evoluzione. *Methodos*, 6(21–22):45–68, 1954.

[151] Nils Aaall Barricelli. Symbiogenetic evolution processes realized by artificial methods. *Methodos*, 9(35–36):143–182, 1957.

[152] Nils Aaall Barricelli. Numerical testing of evolution theories. part i. theroetical introduction and basic tests. *Acta Biotheoretica*, 16(1/2):69–98, March 1962. ISSN: 0001-5342 (Print) 1572-8358 (Online). doi:10.1007/BF01556771. Received: 27 November 1961, See also [153]. Online available at http://www.springerlink.com/content/y502315688024453/fulltext.pdf [accessed 2007-10-31].

[153] Nils Aaall Barricelli. Numerical testing of evolution theories. part ii. preliminary tests of performance. symbiogenesis and terrestrial life. *Acta Biotheoretica*, 16(3/4):99–126, September 1963. ISSN: 0001-5342 (Print) 1572-8358 (Online). doi:10.1007/BF01556602. Received: 27 November 1961, See also [152]. Online available at http://www.springerlink.com/content/h85817217u25w6q7/fulltext.pdf [accessed 2007-10-31].

[154] Alwyn M. Barry, editor. *GECCO 2002: Proceedings of the Bird of a Feather Workshops, Genetic and Evolutionary Computation Conference*, July 8, 2002, The Roosevelt Hotel, 45th and Madison Avenue, New York, NY, USA. AAAI, 445 Burgess Drive, Menlo Park, CA 94025. Bird-of-a-feather Workshops, GECCO-2002. A joint meeting of the eleventh International Conference on Genetic Algorithms (ICGA-2002) and the seventh Annual Genetic Programming Conference (GP-2002). [331, 1245, 1572].

[155] Thomas Barth, Bernd Freisleben, Manfred Grauer, and Frank Thilo. Distributed solution of optimal hybrid control problems on networks of workstations. In *Proceedings of the IEEE International Conference on Cluster Computing*, pages 162–169, November 28–December 1, 2000, Chemnitz, Germany. ISBN: 0-7695-0896-0. doi:10.1109/CLUSTR.2000.889027.

[156] Peter Bartlett, Yoav Freund, Wee Sun Lee, and Robert E. Schapire. Boosting the margin: a new explanation for the effectiveness of voting methods. *Annals of Statistics*, 26(5):1651–1686, 1998. Online available at http://cs.nyu.edu/~cil217/ML/BoostingMargin.pdf and http://dx.doi.org/10.1214/aos/1024691352 [accessed 2007-09-15]. See also [1826].

[157] William Bateson. *Mendel's Principles of Heredity*. Cambridge University Press, Cambridge, 1909. ISBN: 978-1-42864-819-7. 1930: fourth impression of the 1909 edition.

[158] R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA Journal on Computing*, 6(2):126–140, 1994. Online available at http://citeseer.ist.psu.edu/141556.html and http://rtm.science.unitn.it/~battiti/archive/reactive-tabu-search.ps.gz [accessed 2007-09-15].

[159] James C. Bean and Atidel Ben Hadj-Alouane. A dual genetic algorithm for bounded integer programs. Technical Report 92-53, Department of Industrial and Operations Engineering, The University of Michigan, Ann Arbour, MI 48109-2117, USA, October 1992. Revised in June 1993. Online available at http://hdl.handle.net/2027.42/3480 [accessed 2008-11-15].

[160] Steven J. Beaty. Genetic algorithms for instruction sequencing and scheduling. In *Workshop on Computer Architecture Technology and Formalism for Computer Science Research and Applications*, April 1992. Online available at http://citeseer.ist.psu.edu/16699.html and http://emess.mscd.edu/~beaty/Dossier/Papers/italy.pdf [accessed 2007-07-29].

[161] William Beaudoin, Sébastien Verel, Philippe Collard, and Cathy Escazut. Deceptiveness and neutrality the nd family of fitness landscapes. In *GECCO'06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 507–514, 2006. doi:10.1145/1143997.1144091. In proceedings [352]. Online available at http://doi.acm.org/10.1145/1143997.1144091 [accessed 2007-08-25].

[162] Jörg D. Becker and Ignaz Eisele, editors. *Proceedings of WOPPLOT 83, Parallel Processing: Logic, Organization and Technology*, volume 196/1984 of *Lecture notes in physics*, June 27–29, 1983, Federal Armed Forces University Munich (HSBw M), Neubiberg, Bavaria, Germany. Springer-Verlag GmbH. ISBN: 3-5401-2917-0, 978-3-54012-917-2. doi:10.1007/BFb0018249. Published in 1984.

[163] Jörg D. Becker and Ignaz Eisele, editors. *Proceedings of the Workshop WOPPLOT 86 Parallel Processing: Logic, Organization, and Technology*, volume 253/1987 of *Lecture Notes in Computer Science (LNCS)*, July 2–4, 1986, Neubiberg, Federal Republic of Germany. Springer Berlin / Heidelberg. ISBN: 978-3-54018-022-7, 3-5401-8022-2, 0-3871-8022-2. doi:10.1007/3-540-18022-2. Published in 1987.

[164] Jörg D. Becker, Ignaz Eisele, and E W. Mündemann, editors. *Parallelism, Learning, Evolution: Proceedings of the Workshop Evolutionary Models and Strategies and the Workshop on Parallel Processing: Logic, Organization, and Technology – WOPPLOT 89*, volume 565/1991 of *Lecture Notes in Computer Science (LNCS)*, March 10–11 and July24–28, 1989, Neubiberg, Germany and Wildbad Kreuth, Germany. Springer Berlin / Heidelberg. ISBN: 978-3-54055-027-3, 3-5405-5027-5, 0-3875-5027-5. doi:10.1007/3-540-55027-5. Published in 1991.

[165] Mark A. Bedau. Artificial life: organization, adaptation and complexity from the bottom up. *TRENDS in Cognitive Sciences*, 7(11):505–512, November 2003. Online available at http://people.reed.edu/~mab/publications/papers/BedauTICS03.pdf and http://dx.doi.org/10.1016/j.tics.2003.09.012 [accessed 2007-12-13].

[166] Witold Bednorz, editor. *Advances in Greedy Algorithms*. IN-TECH Education and Publishing, Kirchengasse 43/3, 1070 Vienna, Austria, November 2008. ISBN: 978-9-53761-927-5. Online available at http://intechweb.org/downloadfinal.php?is=978-953-7619-27-5&type=B [accessed 2009-01-13].

[167] Niko Beerenwinkel, Lior Pachter, and Bernd Sturmfels. Epistasis and shapes of fitness landscapes, 2006. Eprint arXiv:q-bio/0603034 (Quantitative Biology, Populations and Evolution). Online available at http://arxiv.org/abs/q-bio.PE/0603034 [accessed 2007-08-05].

[168] Boris Beizer. *Software Testing Techniques*. International Thomson Computer Press, second edition, June 1990. ISBN: 1-8503-2880-3, 978-1-85032-880-3.

[169] Richard K. Belew. When both individuals and populations search: adding simple learning to the genetic algorithm. In *Proceedings of the third international conference on Genetic algorithms*, pages 34–41, 1989. In proceedings [1820].

[170] Richard K. Belew and Lashon Bernard Booker, editors. *Proceedings of the 4th International Conference on Genetic Algorithms*, July 1991, San Diego, CA, USA. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN: 1-5586-0208-9.

[171] Richard K. Belew and Melanie Mitchell, editors. *Adaptive individuals in evolving populations: models and algorithms*, volume 26 of *Santa Fe Institue Studies in the Sciences of Complexity*. Addison-Wesley Longman Publishing Co., Inc. / Westview Press, Boston, MA, USA, January 15, 1996. ISBN: 0-2014-8369-6, 978-0-20148-369-7.

[172] Richard K. Belew and Michael D. Vose, editors. *Proceedings of the 4th Workshop on Foundations of Genetic Algorithms (FOGA)*, August 5, 1996, University of San Diego, San Diego, CA, USA. Morgan Kaufmann, San Francisco, California, USA. ISBN: 1-5586-0460-X. Published March 1, 1997.

[173] Bartłomiej Beliczyński, Andrzej Dzieliński, Marcin Iwanowski, and Bernardete Ribeiro, editors. *Proceedings of Adaptive and Natural Computing Algorithms, 8th International Conference, ICANNGA 2007, Part I*, volume 4431/2007 of *Lecture Notes in Computer Science (LNCS)*, April 11–14, 2007, Warsaw University of Technology, Warsaw, Poland. Springer Berlin Heidelberg New York. ISBN: 978-3-54071-589-4. doi:10.1007/978-3-540-71618-1. See also [174], http://icannga07.ee.pw.edu.pl/ [accessed 2007-08-31], and http://www.springerlink.com/content/978-3-540-71590-0/ [accessed 2007-08-31].

[174] Bartłomiej Beliczyński, Andrzej Dzieliński, Marcin Iwanowski, and Bernardete Ribeiro, editors. *Proceedings of Adaptive and Natural Computing Algorithms, 8th International Conference, ICANNGA 2007, Part II*, volume 4432/2007 of *Lecture Notes in Computer Science (LNCS)*, April 11–14, 2007, Warsaw University of Technology, Warsaw, Poland. Springer Berlin Heidelberg New York. ISBN: 978-3-54071-590-0. doi:10.1007/978-3-540-71629-7. See also [173], http://icannga07.ee.pw.edu.pl/ [accessed 2007-08-31], and http://www.springerlink.com/content/978-3-540-71589-4/ [accessed 2007-08-31].

[175] Mordechai Ben-Ari. *Principles of Concurrent and Distributed Programming*. Prentice-Hall International Series in Computer Science. Addison Wesley / Pearson Education, 2nd edition, March 6, 2006. ISBN: 0-3213-1283-X, 978-0-32131-283-9. Partly online available at http://books.google.de/books?id=oP-2hpMEdb8C [accessed 2008-10-02].

[176] Mordechai Ben-Ari. *Principles of the Spin Model Checker*. Springer, Berlin, Germany, January 2008. ISBN: 1-8462-8769-3, 978-1-84628-769-5.

[177] Aharon Ben-Tal. Characterization of pareto and lexicographic optimal solutions. In *Proceedings of the Third Conference on Multiple Criteria Decision Making: Theory and Application*, pages 1–11, 1979. In proceedings [1467].

[178] J.H. Bennett, editor. *The Collected Papers of R.A. Fisher*, volume I–V. University of Adelaide, SA 5005, Australia, 1971–1974. Collected Papers of R.A. Fisher, Relating to Statistical and Mathematical Theory and Applications (excluding Genetics): Online

available at http://digital.library.adelaide.edu.au/coll/special/fisher/stat_math.html [accessed 2008-12-08].

[179] Forrest H. Bennett III. Emergence of a multi-agent architecture and new tactics for the ant colony foraging problem using genetic programming. In *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior: From animals to animats 4*, pages 430–439, 1996. In proceedings [1346].

[180] Forrest H Bennett III. Automatic creation of an efficient multi-agent architecture using genetic programming with architecture-altering operations. In *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 30–38, 1996. In proceedings [1207].

[181] Peter J. Bentley, editor. *Evolutionary Design by Computers*. Morgan Kaufmann, San Francisco, USA, June 15, 1999. ISBN: 1-5586-0605-X. See also http://www.cs.ucl.ac.uk/staff/P.Bentley/evdes.html [accessed 2007-10-03].

[182] Aviv Bergman and Marcus W. Feldman. Recombination dynamics and the fitness landscape. *Physica D: Nonlinear Phenomena*, 56(1):57–67, April 1992. ISSN: 0167-2789. doi:10.1016/0167-2789(92)90050-W. Online available at http://dx.doi.org/10.1016/0167-2789(92)90050-W [accessed 2008-07-20].

[183] Pavel Berkhin. Survey of clustering data mining techniques. Technical Report, Accrue Software, San Jose, CA, 2002. Online available at http://www.ee.ucr.edu/~barth/EE242/clustering_survey.pdf and http://citeseer.ist.psu.edu/berkhin02survey.html [accessed 2007-08-27].

[184] Omer Berkman, Dany Breslauer, Zvi Galil, Baruch Schieber, and Uzi Vishkin. Highly parallelizable problems. In *STOC'89: Proceedings of the twenty-first annual ACM symposium on Theory of computing*, pages 309–319, 1989, Seattle, Washington, United States. ACM Press, New York, NY, USA. ISBN: 0-8979-1307-8. doi:10.1145/73007.73036. Online available at http://doi.acm.org/10.1145/73007.73036 and http://libra.msra.cn/paperDetail.aspx?id=907688 [accessed 2007-08-13].

[185] M.C. Jones Bernard. W. Silverman. Fix, e. and hodges, j. l. (1951): An important contribution to nonparametric discriminant analysis and density estimation: Commentary on fix and hodges (1951). *International Statistical Review / Revue Internationale de Statistique*, 57(3):233–247, April 1989.

[186] Michael J. A. Berry and Gordon S. Linoff. *Mastering Data Mining: The Art and Science of Customer Relationship Management*. Wiley, first edition, December 1999. ISBN: 978-0-47133-123-0.

[187] Michael W. Berry, editor. *Survey of Text Mining: Clustering, Classification, and Retrieval*. Springer, September 2003. ISBN: 978-0-38795-563-6.

[188] Alberto Bertoni and Marco Dorigo. Implicit parallelism in genetic algorithms. *Artificial Intelligence*, 61(2):307–314, 1993. Online available at http://citeseer.ist.psu.edu/bertoni93implicit.html [accessed 2007-07-29].

[189] P. Besson, J.-M. Vesin, V. Popovici, and M. Kunt. Differential evolution applied to a multimodal information theoretic optimization problem. In *Applications of Evolutionary Computing, Proceedings of the 8th European Workshop on Evolutionary Computation in Image Analysis and Signal Processing (evoIASP)*, pages 505–509, 2006. In proceedings [1768].

[190] Albert Donally Bethke. *Genetic algorithms as function optimizers*. PhD thesis, University of Michigan, University of Michigan, Ann Arbor, MI, USA, 1980. Order No. AAI8106101, Dissertation Abstracts International, 41(9), 3503B (University Microfilms No. 8106101). See also http://hdl.handle.net/2027.42/3572 [accessed 2007-10-28].

[191] Pete Bettinger and Jianping Zhu. A new heuristic method for solving spatially constrained forest planning problems based on mitigation of infeasibilities radiating outward from a forced choice. *Silva Fennica*, 40(2):315–333, 2006. ISSN: 0037-5330. Online available at http://www.metla.fi/silvafennica/full/sf40/sf402315.pdf [accessed 2008-08-24].

[192] Gregory Beurier, Fabien Michel, and Jacques Ferber. A morphogenesis model for multiagent embryogeny. In *ALIFE X, Tenth International Conference on the Simulation and Synthesis of Living Systems*, June 2006. Bloomington, Indiana, USA. Online available at http://leri.univ-reims.fr/~fmichel/publi/pdfs/beurier06alifeX.pdf [accessed 2007-08-17].

[193] Hans-Georg Beyer. Some aspects of the 'evolution strategy' for solving TSP-like optimization problems appearing at the design studies os a $0.5teve^+e^-$-linear collider. In *Parallel problem solving from nature 2*, pages 361–370, 1992. In proceedings [1357].

[194] Hans-Georg Beyer. Towards a theory of 'evolution strategies': Some asymptotical results from the $(1, +\lambda)$-theory. Technical Report SYS–5/92, Fachbereich Informatik, Universität Dortmund, P.O. Box 500500, 44227 Dortmund, Germany, December 1, 1992. See also [195].

[195] Hans-Georg Beyer. Toward a theory of evolution strategies: Some asymptotical results from the $(1, +\lambda)$-theory. *Evolutionary Computation*, 1(2):165–188, Summer 1993. ISSN: 1063-6560, 1530-9304. Online available at http://citeseer.ist.psu.edu/beyer93towards.html and http://ls11-www.cs.uni-dortmund.de/~beyer/coll/Bey93a/Bey93a.ps [accessed 2008-07-19]. See also [194].

[196] Hans-Georg Beyer. Toward a theory of evolution strategies: The (μ, λ)-theory. *Evolutionary Computation*, 2(4):381–407, 1994. Online available at http://ls11-www.cs.uni-dortmund.de/people/beyer/coll/Bey95a/Bey95a.ps and http://citeseer.ist.psu.edu/249681.html [accessed 2007-08-27].

[197] Hans-Georg Beyer. Design optimization of a linear accelerator using evolution strategy: solving a TSP-like optimization problem. In *Handbook of Evolutionary Computation*, pages G4.2:1–8. Institute of Physics Publishing and Oxford University Press, 1997. In collection [104].

[198] Hans-Georg Beyer. *The theory of evolution strategies*. Natural Computing Series. Springer-Verlag New York, Inc., New York, NY, USA, March 27, 2001. ISBN: 978-3-54067-297-5, 3-5406-7297-4. Partly online available at http://books.google.de/books?id=8tbInLufkTMC [accessed 2008-06-07]. Series editors: G. Rozenberg, Thomas Bäck, Ágoston E. Eiben, J.N. Kok, and H.P. Spaink.

[199] Hans-Georg Beyer and Una-May O'Reilly, editors. *Workshop Proceedings of Genetic and Evolutionary Computation Conference, GECCO 2005*, June 25–29, 2005, Loews L'Enfant Plaza Hotel, 480 L'enfant Plaza Sw, Washington, D.C. 20024, USA. ACM. ACM Order Number 910052. See also [202, 1764, 1766].

[200] Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies – a comprehensive introduction. *Natural Computing: an international journal*, 1(1):3–52, March 2002. ISSN: 1567-7818 (Print) 1572-9796 (Online). doi:10.1023/A:1015059928466. Online available at http://dx.doi.org/10.1023/A:1015059928466 and http://www.springerlink.com/content/2311qapbrwgrcyey/ [accessed 2007-08-27].

[201] Hans-Georg Beyer, Eva Brucherseifer, Wilfried Jakob, Hartmut Pohlheim, Bernhard Sendhoff, and Thanh Binh To. *Evolutionary Algorithms – Terms and Definitions*. Universität Dortmund, Informatik XI (Systemanalyse), D-44221 Dortmund, Germany, version e-1.2 edition, February 25, 2002. Online available at http://ls11-www.cs.uni-dortmund.de/people/beyer/EA-glossary/ [accessed 2008-04-10].

[202] Hans-Georg Beyer, Una-May O'Reilly, Dirk V. Arnold, Wolfgang Banzhaf, Christian Blum, Eric W. Bonabeau, Erick Cantú-Paz, Dipankar Dasgupta, Kalyanmoy Deb, James A. Foster, Edwin D. de Jong, Hod Lipson, Xavier Llorà, Spiros Mancoridis, Martin Pelikan, Guenther R. Raidl, Terence Soule, Andy M. Tyrrell, Jean-Paul Watson, and Eckart Zitzler, editors. *Proceedings of Genetic and Evolutionary Computation Conference, GECCO 2005*, June 25–29, 2005, Loews L'Enfant Plaza Hotel, 480 L'enfant Plaza Sw, Washington, D.C. 20024, USA. ACM Press, Washington DC, USA. ISBN: 1-5959-3010-8. Order Number: 910050. GECCO-2005 A joint meeting of the fourteenth international conference on genetic algorithms (ICGA-2005) and

the tenth annual genetic programming conference (GP-2005). ACM Order Number 910052. See also [199, 1764, 1766].

[203] James C. Bezdek, James Keller, Raghu Krisnapuram, and Nikhil R. Pal. *Fuzzy Models and Algorithms for Pattern Recognition and Image Processing*. The Handbooks of Fuzzy Sets. Springer, March 2005. ISBN: 978-0-38724-515-7.

[204] V. Bhaskar, Santosh K. Gupta, and A.K. Ray. Applications of multiobjective optimization in chemical engineering. *Reviews in Chemical Engineering*, 16(1):1–54, 2000.

[205] Gouri K. Bhattacharyya and Richard A. Johnson. *Statistical Concepts and Methods*. John Wiley & Sons, Chichester, UK, March 8, 1977. ISBN: 0-4710-7204-4, 978-0-47107-204-1, 978-0-47103-532-9, 0-4710-3532-7.

[206] Arthur S. Bickel and Riva W. Bickel. Tree structured rules in genetic algorithms. In *Proceedings of the Second International Conference on Genetic algorithms and their application*, pages 77–81, 1987. In proceedings [857].

[207] Joseph P. Bigus. *Data Mining With Neural Networks: Solving Business Problems from Application Development to Decision Support*. Mcgraw-Hill (Tx), May 20, 1996. ISBN: 978-0-07005-779-1.

[208] K. Binder and A. P. Young. Spin glasses: Experimental facts, theoretical concepts, and open questions. *Reviews of Modern Physics*, 58(4):801–976, October 1986. doi:10.1103/RevModPhys.58.801. Online available at `http://link.aps.org/abstract/RMP/v58/p801` [accessed 2008-07-02].

[209] Lothar Birk, Günther F. Clauss, and June Y. Lee. Practical application of global optimization to the design of offshore structures. In *Proceedings of OMAE04, 23rd International Conference on Offshore Mechanics and Arctic Engineering*, June 20–25, 2004, Vancouver, British Columbia, Canada. OM AE2004-51225. Online available at `http://130.149.35.79/downloads/publikationen/2004/OMAE2004-51225.pdf` [accessed 2007-09-01].

[210] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, USA, November 3, 1995. ISBN: 0-1985-3864-2, 978-0-19853-864-6. Partly online available at `http://books.google.de/books?id=-aAwQO_-rXwC` [accessed 2009-05-12].

[211] Tim Blackwell. Particle swarm optimization in dynamic environments. In *Evolutionary Computation in Dynamic and Uncertain Environments*, chapter 2, pages 29–52. Springer, 2007. In collection [2280]. See also [2280]. Online available at `http://igor.gold.ac.uk/~mas01tb/papers/PSOdynenv.pdf` [accessed 2007-08-19].

[212] M. Brian Blake, Kwok Ching Tsui, and Andreas Wombacher. The eee-05 challenge: a new web service discovery and composition competition. In *Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce, and e-Service, EEE'05*, pages 780–783, March 29–April 1, 2005. doi:10.1109/EEE.2005.131. Online available at `http://ws-challenge.georgetown.edu/ws-challenge/The%20EEE.htm` [accessed 2007-09-02].

[213] M. Brian Blake, William K.W. Cheung, Michael C. Jaeger, and Andreas Wombacher. Wsc-06: The web service challenge. In *Proceedings of 2006 IEEE Joint Conference on E-Commerce Technology (CEC'06) and Enterprise Computing, E-Commerce and E-Services (EEE'06)*, pages 505–508, 2006. In proceedings [992].

[214] M. Brian Blake, William K.W. Cheung, Michael C. Jaeger, and Andreas Wombacher. Wsc-07: Evolving the web service challenge. In *Proceedings of IEEE Joint Conference (CEC/EEE 2007) on E-Commerce Technology (9th CEC'07) and Enterprise Computing, E-Commerce and E-Services (4th EEE'07)*, pages 422–423, 2007. In proceedings [994].

[215] Woodrow "Woody" Wilson Bledsoe. The use of biological concepts in the analytical study of systems. Technical Report PRI 2, Panoramic Research, Inc., Palo Alto,

California, USA, 1961. Presented at ORSA-TIMS National Meeting, San Francisco, California, November 10, 1961.

[216] Woodrow "Woody" Wilson Bledsoe. Lethally dependent genes using instant selection. Technical Report PRI 1, Panoramic Research Inc., Palo Alto, California, USA, 1961.

[217] Woodrow "Woody" Wilson Bledsoe. An analysis of genetic populations. Technical Report, Panoramic Research Inc., Palo Alto, California, USA, 1962.

[218] Woodrow "Woody" Wilson Bledsoe. The evolutionary method in hill climbing: Convergence rates. Technical Report, Panoramic Research Inc., Palo Alto, California, USA, 1962.

[219] Woodrow "Woody" Wilson Bledsoe and I. Browning. Pattern recognition and reading by machine. In *Proceedings of the Eastern Joint Computer Conference (EJCC) – Papers and Discussions Presented at the Joint IRE - AIEE - ACM Computer Conference*, pages 225–232. The 1959 Eastern Joint Computer Conference for The National Joint Computer Committee, December 1–3, 1959, Boston, Massachusetts, USA. ASIN: B001BS7LVG.

[220] Maria J. Blesa, Pablo Moscato, and Fatos Xhafa. A memetic algorithm for the minimum weighted k-cardinality tree subgraph problem. In *Proceedings of the 4th Metaheuristics International Conference*, pages 85–90, 2001. In proceedings [1721]. Online available at http://citeseer.ist.psu.edu/458772.html and http://citeseer.ist.psu.edu/649471.html [accessed 2007-09-12].

[221] Steffen Bleul. Ähnlichkeitsmaße und clustering (text-based similarity measures and clustering), 2002. Online available at http://www.vs.uni-kassel.de/~bleul/ [accessed 2007-08-11].

[222] Steffen Bleul and Kurt Geihs. Addo: Automatic service brokering in service oriented architectures, project homepage, 2007. Online available at http://www.vs.uni-kassel.de/ADDO/ [accessed 2007-09-02].

[223] Steffen Bleul and Thomas Weise. An Ontology for Quality-Aware Service Discovery. In C. Zirpins, G. Ortiz, W. Lamerdorf, and W. Emmerich, editors, *Engineering Service Compositions: First International Workshop, WESC05*, volume RC23821 of *IBM Research Report*. Yorktown Heights: IBM Research Devision, December 12, 2005, Vrije Universiteit Amsterdam, The Netherlands. See also [224]. Online available at http://www.it-weise.de/documents/files/BW2005QASD.pdf [accessed 2009-06-26].

[224] Steffen Bleul, Thomas Weise, and Kurt Geihs. An Ontology for Quality-Aware Service Discovery. *Computer Systems Science Engineering*, 21(4)(Special issue on "Engineering Design and Composition of Service-Oriented Applications"), July 2006. See also [223]. Online available at http://www.it-weise.de/documents/files/BWG2006QASD.pdf [accessed 2009-06-26].

[225] Steffen Bleul, Thomas Weise, and Kurt Geihs. Large-Scale Service Composition in Semantic Service Discovery. In Ws-Challenge Part: M. Brian Blake, Andreas Wombacher, Michel C. Jaeger, and William K. Cheung, editors, *Proceedings of 2006 IEEE Joint Conference on E-Commerce Technology (CEC'06) and Enterprise Computing, E-Commerce and E-Services (EEE'06)*, pages 427–429, 2006. In proceedings [992]. 1st place in 2006 WSC. Online available at http://www.it-weise.de/documents/files/BWG2006WSC.pdf [accessed 2009-06-26]. See 2007 WSC [226] and [2179].

[226] Steffen Bleul, Thomas Weise, and Kurt Geihs. Making a Fast Semantic Service Composition System Faster. In *Proceedings of IEEE Joint Conference (CEC/EEE 2007) on E-Commerce Technology (9th CEC'07) and Enterprise Computing, E-Commerce and E-Services (4th EEE'07)*, pages 517–520, 2007. In proceedings [994]. 2nd place in 2007 WSC. Online available at http://www.it-weise.de/documents/files/BWG2007WSC.pdf [accessed 2009-06-26]. See 2006 WSC [225] and [2179].

[227] Stefan Bleuler, Martin Brack, Lothar Thiele, and Eckart Zitzler. Multiobjective genetic programming: Reducing bloat using SPEA2. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, pages 536–543, 2001. In proceed-

ings [1003]. Online available at http://citeseer.ist.psu.edu/443099.html and ftp://ftp.tik.ee.ethz.ch/pub/people/zitzler/BBTZ2001b.ps.gz [accessed 2007-09-07].

[228] Tobias Blickle. *Theory of Evolutionary Algorithms and Application to System Synthesis*. PhD thesis, ETH Zurich, Verlag der Fachvereine Hochschulverlag AG of ETH Zurich, November 1996. ISBN: 978-3-72812-433-3. ETH-Thesis number 11894. Examiner: Lothar Thiele, Hans-Paul Schwefel. Online available at http://www.tik.ee.ethz.ch/~tec/publications/bli97a/diss.ps.gz and http://www.handshake.de/user/blickle/publications/diss.pdf [accessed 2007-09-07].

[229] Tobias Blickle and Lothar Thiele. Genetic programming and redundancy. In J. Hopf, editor, *Genetic Algorithms within the Framework of Evolutionary Computation, Workshop at KI-94*, pages 33–38, 1994, Saarbrücken, Germany. Max-Planck-Institut für Informatik (MPI-I-94-241), Im Stadtwald, Building 44, D-66123 Saarbrücken, Germany. Online available at http://citeseer.ist.psu.edu/44816.html and http://www.tik.ee.ethz.ch/~tec/publications/bt94/GPandRedundancy.ps.gz [accessed 2007-09-07].

[230] Tobias Blickle and Lothar Thiele. A comparison of selection schemes used in genetic algorithms. Technical Report 11, Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH), Gloriastrasse 35, 8092 Zurich, Switzerland, 1995. Online available at http://citeseer.ist.psu.edu/16412.html and http://www.tik.ee.ethz.ch/~tec/publications/bt95a/ [accessed 2007-07-28].

[231] Tobias Blickle and Lothar Thiele. A mathematical analysis of tournament selection. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 9–16, 1995. In proceedings [636]. Online available at http://citeseer.ist.psu.edu/blickle95mathematical.html [accessed 2007-07-29].

[232] Tobias Blickle and Lothar Thiele. A comparison of selection schemes used in evolutionary algorithms. *Evolutionary Computation*, 4(4):361–394, 1996. Online available at http://citeseer.ist.psu.edu/blickle97comparison.html [accessed 2007-08-24].

[233] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003. ISSN: 0360-0300. CODEN: CMSVAN. Online available at http://iridia.ulb.ac.be/~meta/newsite/downloads/ACSUR-blum-roli.pdf [accessed 2008-10-06].

[234] Lenore Blum, Manuel Blum, and Michael Shub. A simple unpredictable pseudo-random number generator. *SIAM Journal on Computing*, 15:364–383, May 1986. ISSN: 0097–5397.

[235] Egbert J. W. Boers, Jens Gottlieb, Pier Luca Lanzi, Robert E. Smith, Stefano Cagnoni, Emma Hart, Günther R. Raidl, and H. Tijink, editors. *Applications of Evolutionary Computing, Proceedings of EvoWorkshops 2001: EvoCOP, EvoFlight, EvoIASP, EvoLearn, and EvoSTIM*, volume 2037/2001 of *Lecture Notes in Computer Science (LNCS)*, April 18–20, 2001, Lake Como, Milan, Italy. Springer, Berlin / Heidelberg. ISBN: 3-5404-1920-9.

[236] Stefan Boettcher. Extremal optimization of graph partitioning at the percolation threshold. *Journal of Physics A: Mathematical and General*, 32(28):5201–5211, July 16, 1999. Online available at http://www.iop.org/EJ/article/0305-4470/32/28/302/a92802.pdf and http://xxx.lanl.gov/abs/cond-mat/9901353 [accessed 2008-08-23].

[237] Stefan Boettcher and Allon G. Percus. Extremal optimization: Methods derived from co-evolution. In *GECCO-99: Proceedings of the Genetic and Evolutionary Computation Conferenc*, 1999. In proceedings [142]. Online available at http://arxiv.org/abs/math.OC/9904056 [accessed 2008-08-23].

[238] Stefan Boettcher and Allon G. Percus. Extremal optimization for graph partitioning. *Physical Review E*, 64(2), July 2001. doi:10.1103/PhysRevE.64.026114. Online

available at http://www.math.ucla.edu/~percus/Publications/eopre.pdf and http://arxiv.org/abs/cond-mat/0104214 [accessed 2008-08-23].

[239] Stefan Boettcher and Allon G. Percus. Optimization with extremal dynamics. *Physical Review Letters*, 86(23):5211–5214, June 4, 2001. doi:10.1103/PhysRevLett.86.521. Online available at http://arxiv.org/abs/cond-mat/0010337 and http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.94.3835 [accessed 2008-08-24].

[240] Stefan Boettcher and Allon G. Percus. Optimization with extremal dynamics. *Complexity*, 8(2):57–62, March 21, 2003. doi:10.1002/cplx.10072. Special Issue: Complex Adaptive Systems: Part I. Online available at http://www3.interscience.wiley.com/cgi-bin/fulltext/104085596/PDFSTART [accessed 2008-08-24].

[241] Walter Bohm and Andreas Geyer-Schulz. Exact uniform initialization for genetic programming. In *Foundations of Genetic Algorithms IV*, pages 379–407, 1996. In proceedings [172]. k-bounded context-free languages.

[242] Celia C. Bojarczuk, Heitor S. Lopes, and Alex A. Freitas. Data mining with constrained-syntax genetic programming: applications to medical data sets. In *Proceedings Intelligent Data Analysis in Medicine and Pharmacology (IDAMAP-2001)*, 2001. Online available at http://citeseer.ist.psu.edu/bojarczuk01data.html and http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/bojarczuk_2001_idamap.html [accessed 2007-09-09].

[243] Celia C. Bojarczuk, Heitor S. Lopes, and Alex A. Freitas. An innovative application of a constrained-syntax genetic programming system to the problem of predicting survival of patients. In *Genetic Programming: Proc. 6th European Conference (EuroGP-2003)*, pages 11–59, 2003. In proceedings [1786]. Online available at http://www.cs.kent.ac.uk/people/staff/aaf/pub_papers.dir/EuroGP-2003-Celia.pdf [accessed 2007-09-09].

[244] Alessandro Bollini and Marco Piastra. Distributed and persistent evolutionary algorithms: a design pattern. In *Genetic Programming, Proceedings of EuroGP'99*, pages 173–183, 1999. In proceedings [1664].

[245] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems*. Oxford University Press USA, Oxford University Press, 198 Madison Avenue, New York, NY 10016, U.S.A., August 1999. ISBN: 0-1951-3159-2, 978-0-19513-159-8. Partly online available at http://books.google.de/books?id=mhLj70l8HfAC [accessed 2008-08-23].

[246] Josh C. Bongard and Chandana Paul. Making evolution an offer it can't refuse: Morphology and the extradimensional bypass. In *Advances in Artificial Life, Proceedings of the 6th European Conference, ECAL 2001*, pages 401–412, 2001. doi:10.1007/3-540-44811-X_43. In proceedings [1118]. Online available at http://www.cs.uvm.edu/~jbongard/papers/bongardPaulEcal2001.ps.gz and http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.27.8240 [accessed 2008-11-02].

[247] John Tyler Bonner. *On Development: The Biology of Form*. Commonwealth Fund Publications. Harvard University Press, Cambridge, MA, USA, new ed edition, December 12, 1974. ISBN: 0-6746-3412-8, 978-0-67463-412-1.

[248] Lashon Bernard Booker. *Intelligent behavior as an adaptation to the task environment*. PhD thesis, University of Michigan, Ann Arbor, MI, USA, 1982. Order No. AAI8214966, Technical Report No. 243. Dissertation Abstracts International, 43(2), 469B. (University Microfilms No. 8214966). Online available at http://hdl.handle.net/2027.42/3746 and http://hdl.handle.net/2027.42/3747 [accessed 2008-03-19].

[249] David Booth and Canyang Kevin Liu. *Web Services Description Language (WSDL) Version 2.0 Part 0: Primer*. World Wide Web Consortium (W3C), June 26, 2007. W3C Recommendation. Online available at http://www.w3.org/TR/2007/REC-wsdl20-primer-20070626 [accessed 2007-09-02].

[250] István Borgulya. A cluster-based evolutionary algorithm for the single machine total weighted tardiness-scheduling problem. *Journal of Computing and Information Technology - CIT*, 10(3):211–217, September 2002. Online available at http://cit.zesoi.fer.hr/downloadPaper.php?paper=384 [accessed 2008-04-07].

[251] Erich Bornberg-Bauer and Hue Sun Chan. Modeling evolutionary landscapes: Mutational stability, topology, and superfunnels in sequence space. *Proceedings of the National Academy of Science of the United States of Americs (PNAS) – Biophysics*, 96 (19):10689–10694, September 14, 1999. Edited by Peter G. Wolynes. Online available at http://www.pnas.org/cgi/content/abstract/96/19/10689 [accessed 2008-07-02].

[252] Jürgen Bortz, Gustav Adolf Lienert, and Klaus Boehnke. *Verteilungsfreie Methoden in der Biostatistik*. Springer-Lehrbuch. Springer-Verlag GmbH, 3., korrigierte auflage edition, 2008. ISBN: 3-5406-7590-6, 978-3-54067-590-7, 978-3-54074-706-2, 978-3-54074-707-9. doi:10.1007/978-3-540-74707-9. Online available at http://www.springerlink.com/content/978-3-540-74706-2 [accessed 2008-12-08]. Partly online available at http://books.google.de/books?id=gWvgo8MWGO8C [accessed 2008-08-16].

[253] Peter A. N. Bosman and Dirk Thierens. Multi-objective optimization with diversity preserving mixture-based iterated density estimation evolutionary algorithms. *International Journal Approximate Reasoning*, 31(3):259–289, 2002. doi:10.1016/S0888-613X(02)00090-7. Received 1 January 2002; accepted 1 July 2002. Online available at http://dx.doi.org/10.1016/S0888-613X(02)00090-7 [accessed 2008-03-15].

[254] Peter A. N. Bosman and Dirk Thierens. A thorough documentation of obtained results on real-valued continuous and combinatorial multi-objective optimization problems using diversity preserving mixture-based iterated density estimation evolutionary algorithms. Technical Report UU-CS-2002-052, Institute of Information and Computing Sciences, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, The Netherlands, December 2002. Online available at http://www.cs.uu.nl/research/techreps/repo/CS-2002/2002-052.pdf [accessed 2007-08-24].

[255] Peter A. N. Bosman and Dirk Thierens. The naive $\mathbb{M}$id$\mathbb{E}$a: A baseline multi-objective ea. In *Proceedings of the Third International Conference on Evolutionary Multi-Criterion Optimization*, pages 428–442, 2005. In proceedings [422]. Online available at http://springerlink.metapress.com/content/m01x9t6l2bwd/?p_o=20 [accessed 2007-08-22].

[256] Dragan Bošnački and Stefan Edelkamp, editors. *Proceedings of SPIN 2007, the 14th International SPIN Workshop on Model Checking Software*, volume 4595/2007 of *Lecture Notes in Computer Science (LNCS)*, July 1–3, 2007, Berlin, Germany. Springer Berlin / Heidelberg. ISBN: 978-3-54073-369-0. doi:10.1007/978-3-540-73370-6. Online available at http://spinroot.com/spin/Workshops/ws07/index.html [accessed 2008-10-01].

[257] Jack Bosworth, Norman Foo, and Bernard P. Zeigler. Comparison of genetic algorithms with conjugate gradient methods. ORA Tech Report 00312-1-T, Computer and Communication Sciences Department, The University of Michigan, Ann Arbor, Michigan, USA, February 1972. Other Identifiers: UMR0554. Online available at http://hdl.handle.net/2027.42/376 [accessed 2008-09-10].

[258] Henry Bottomley. Relationship between the mean, median, mode, and standard deviation in a unimodal distribution, September 2006. Online available at http://www.btinternet.com/~se16/hgb/median.htm [accessed 2007-09-15].

[259] Chris P. Bowers. Simulating evolution with a computational model of embryogeny: Obtaining robustness from evolved individuals. In *Advances in Artificial Life, Proceedings of 8th European Conference*, pages 149–158, 2005. In proceedings [338]. Online available at http://www.cs.bham.ac.uk/~cpb/publications/ecal05_bowers.pdf [accessed 2007-08-17].

[260] George E. P. Box. Evolutionary operation: A method for increasing industrial productivity. *Applied Statistics*, 6(2):81–101, June 1957. ISSN: 00359254.

[261] George E. P. Box and Norman R. Draper. *Evolutionary operation. A statistical method for process improvement.* Wiley Publication in Applied Statistics. John Wiley & Sons, 1969. ISBN: 0-4710-9305-X, 978-0-47109-305-3.

[262] George Edward Pelham Box and Mervin Edgar Muller. A note on the generation of random normal deviates. *Annals Math. Stat*, 29(5):610–611, 1958. Online available at http://projecteuclid.org/euclid.aoms/1177706645 [accessed 2007-09-15].

[263] George Edward Pelham Box, J. Stuart Hunter, and William G. Hunter. *Statistics for Experimenters: Design, Innovation, and Discovery.* John Wiley & Sons, May 1st ed: 1978, 2nd ed: 2005. ISBN: 0-4717-1813-0, 978-0-47171-813-0, 4-7109-3157-, 978-0-47109-315-2.

[264] Anthony Brabazon and Michael O'Neill. Evolving financial models using grammatical evolution. In *Proceedings of The Annual Conference of the South Eastern Accounting Group (SEAG) 2003*, September 8, 2003, London Metropolitan University, London.

[265] Anthony Brabazon and Michael O'Neill. A grammar model for foreign exchange trading. In H. R. Arabnia et al., editor, *Proceedings of the International Conference on Artificial Intelligence*, volume II, pages 492–498. CSREA Press, June 2003. ISBN: 1-9324-1513-0.

[266] Anthony Brabazon, Michael O'Neill, Robin Matthews, and Conor Ryan. Grammatical evolution and corporate failure prediction. In *GECCO'02: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1011–1018, 2002. In proceedings [1245]. Online available at http://business.kingston.ac.uk/research/intbus/paper4.pdf and http://www.cs.bham.ac.uk/~wbl/biblio/gecco2002/RWA145.pdf [accessed 2007-09-09].

[267] Ronald J. Brachman, editor. *Proceedings of the National Conference on Artificial Intelligence, AAAI*, August 6–10, 1984, Austin, TX, USA. AAAI Press. ISBN: 0-2625-1053-7. See http://www.aaai.org/Conferences/AAAI/aaai84.php [accessed 2007-09-06].

[268] R. M. Brady. Optimization strategies gleaned from biological evolution. *Nature (Letters to Nature)*, 317(6040):804–806, October 31, 1985. ISSN: 0028-0836, 1476-4687. doi:10.1038/317804a0. Online available at http://www.nature.com/nature/journal/v317/n6040/pdf/317804a0.pdf [accessed 2008-09-10].

[269] Markus F. Brameier. *On Linear Genetic Programming.* PhD thesis, Fachbereich Informatik, Universität Dortmund, February 2004, Dortmund. Day of Submission: 2003-05-28, Committee: Wolfgang Banzhaf and Martin Riedmiller and Peter Nordin. Online available at https://eldorado.uni-dortmund.de/handle/2003/20098 and http://hdl.handle.net/2003/20098 [accessed 2007-08-17].

[270] Markus F. Brameier and Wolfgang Banzhaf. A comparison of genetic programming and neural networks in medical data analysis. Technical Report, University of Dortmund, 1998. Reihe Computational Intelligence, Sonderforschungsbereich 531. Online available at http://citeseer.ist.psu.edu/324837.html and http://dspace.hrz.uni-dortmund.de:8080/bitstream/2003/5344/2/ci4398_doc.pdf [accessed 2007-09-09].

[271] Markus F. Brameier and Wolfgang Banzhaf. Evolving teams of predictors with linear genetic programming. *Genetic Programming and Evolvable Machines*, 2(4):381–407, December 2001. ISSN: 1389-2576 (Print) 1573-7632 (Online). doi:10.1023/A:1012978805372. Online available at http://dx.doi.org/10.1023/A:1012978805372 [accessed 2007-09-09].

[272] Markus F. Brameier and Wolfgang Banzhaf. A comparison of linear genetic programming and neural networks in medical data mining. *IEEE Transactions on Evolutionary Computation*, 5(1):17–26, 2001. Online available at http://citeseer.ist.psu.

edu/brameier00comparison.html and http://web.cs.mun.ca/~banzhaf/papers/ieee_taec.pdf [accessed 2007-09-09].

[273] Markus F. Brameier and Wolfgang Banzhaf. Explicit control of diversity and effective variation distance in linear genetic programming. In *EuroGP'02: Proceedings of the 5th European Conference on Genetic Programming*, pages 37–49, 2002. In proceedings [737]. Online available at http://citeseer.ist.psu.edu/552561.html [accessed 2007-09-09].

[274] Markus F. Brameier and Wolfgang Banzhaf. Neutral variations cause bloat in linear gp. In *Proceedings of 6th European Conference on Genetic Programming, EuroGP*, pages 997–1039, 2003. In proceedings [1786].

[275] Markus F. Brameier and Wolfgang Banzhaf. *Linear Genetic Programming*, volume 1 of *Genetic and Evolutionary Computation*. Springer, December 11, 2006. ISBN: 0-3873-1029-0, 978-0-38731-029-9, 978-0-38731-030-5. doi:10.1007/978-0-387-31030-5. Series Editor: David L. Goldberg, John R. Koza.

[276] Jürgen Branke. Creating robust solutions by means of evolutionary algorithms. In *Parallel Problem Solving from Nature – PPSN V*, pages 119–128, 1998. doi:10.1007/BFb0056855. In proceedings [624].

[277] Jürgen Branke. The moving peaks benchmark. Technical Report, Institute AIFB, University of Karlsruhe, D-76128 Karlsruhe, Germany, December 16, 1999. Online available at http://www.aifb.uni-karlsruhe.de/~jbr/MovPeaks/ [accessed 2007-08-19]. Presented in [278].

[278] Jürgen Branke. Memory enhanced evolutionary algorithms for changing optimization problems. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC99*, volume 3, pages 1875–1882, 1999. INSPEC Accession Number: 6346978. doi:10.1109/CEC.1999.785502. In proceedings [69]. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.2.8897 and http://www.aifb.uni-karlsruhe.de/~jbr/Papers/memory_final2.ps.gz [accessed 2008-12-07]. See also [277].

[279] Jürgen Branke. *Evolutionary Optimization in Dynamic Environments*. PhD thesis, Universität Karlsruhe (TH), Fakultät für Wirtschaftswissenschaften, Universität Karlsruhe (TH), Institut AIFB, D-76128 Karlsruhe, 2000. Advisors: H. Schmeck and G. Bol and Lothar Thiele. See also [280].

[280] Jürgen Branke. *Evolutionary Optimization in Dynamic Environments*, volume 3 of *Genetic Algorithms and Evolutionary Computation*. Kluwer Academic Publishers, December 2001. ISBN: 978-0-79237-631-6. See also [279].

[281] Jürgen Branke, Kalyanmoy Deb, Kaisa Miettinen, and Ralph E. Steuer, editors. *Practical Approaches to Multi-Objective Optimization*, number 04461 in Dagstuhl Seminar Proceedings, November 7–12, 2004, Dagstuhl, Germany. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany IBFI. Published in 2005. Online available at http://drops.dagstuhl.de/portals/index.php?semnr=04461 and http://www.dagstuhl.de/de/programm/kalender/semhp/?semnr=04461 [accessed 2007-09-19].

[282] Jürgen Branke, Erdem Salihoğlu, and Şima Uyar. Towards an analysis of dynamic environments. In *GECCO'05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1433–1440, 2005. In proceedings [202]. Online available at http://doi.acm.org/10.1145/1068009.1068237 and http://www.citeulike.org/user/denizinho/article/995596 [accessed 2007-08-19].

[283] Jürgen Branke, Kalyanmoy Deb, Kaisa Miettinen, and Roman Slowinski, editors. *Practical Approaches to Multi-Objective Optimization*, number 06501 in Dagstuhl Seminar Proceedings, December 10–15, 2006, Dagstuhl, Germany. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany IBFI. Published in 2007. Online available at http://drops.dagstuhl.de/

portals/index.php?semnr=06501 and http://www.dagstuhl.de/de/programm/kalender/semhp/?semnr=06501 [accessed 2007-09-19].

[284] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and François Yergeau. *Extensible Markup Language (XML) 1.0 (Fourth Edition)*. World Wide Web Consortium (W3C), September 29, 2007. W3C Recommendation. Online available at http://www.w3.org/TR/2006/REC-xml-20060816 [accessed 2007-09-02].

[285] Olli Bräysy and Michel Gendreau. Tabu search heuristics for the vehicle routing problem with time windows. *TOP: An Official Journal of the Spanish Society of Statistics and Operations Research*, 10(2):211–237, December 2002. ISSN: 1134-5764 (Print) 1863-8279 (Online). doi:10.1007/BF02579017. Online available at http://www.springerlink.com/content/854188435g26v0h5/fulltext.pdf [accessed 2008-10-27].

[286] Alex Van Breedam. *An analysis of the behavior of heuristics for the vehicle routing problem for a selection of problems with vehicle-related, customer-related, and time-related constraints*. PhD thesis, University of Antwerp, Antwerp, Belgium, 1994. en analyse van het gedrag van rittenplanningsheuristieken voor een selektie van planningsproblemen met vrachtwagengebonden, klantgebonden en tijdgebonden randvoorwaarden.

[287] Hans J. Bremermann. Optimization through evolution and recombination. In Marshall C. Yovitts, George T. Jacobi, and Gordon D. Goldstein, editors, *Self-Organizing Systems (Proceedings of the conference sponsored by the Information Systems Branch of the Office of Naval Research and the Armour Research Foundation of the Illinois Institute of Technology.)*, pages 93–10, May 22–24, 1962, Chicago, USA. Spartan Books, Washington, D.C., USA. ASIN: B000GXZFFG. Online available at http://holtz.org/Library/Natural%20Science/Physics/ [accessed 2007-10-31].

[288] Janez Brest, Viljem Žumer, and Mirjam Sepesy Maučec. Control parameters in self-adaptive differential evolution. In *Proceedings of the Second International Conference on Bioinspired Optimization Methods and their Application, BIOMA 2006*, pages 35–44, 2006. In proceedings [669]. Online available at http://labraj.uni-mb.si/index.php/Bibliografija [accessed 2007-08-13].

[289] A. Brindle. *Genetic algorithms for function optimization*. PhD thesis, University of Alberta, Department of Computer Science, Edmonton, 1981. Technical Report TR81-2.

[290] David Brittain, Jon Sims Williams, and Chris McMahon. A genetic algorithm approach to planning the telecommunications access network. In *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA97)*, pages 623–628, 1997. In proceedings [98]. Online available at http://citeseer.ist.psu.edu/brittain97genetic.html [accessed 2008-08-01].

[291] *Proceedings of the 14th International Conference on Soft Computing, MENDEL'08*, June 18–20, 2009, Brno, Czech Republic.

[292] *Proceedings of the 15th International Conference on Soft Computing, MENDEL'09*, June 17–26, 2009, Brno, Czech Republic.

[293] *Proceedings of the 12th International Conference on Soft Computing, MENDEL'06*, May 31–June 2, 2006, Brno University of Technology, Brno, Czech Republic. Brno University of Technology, Faculty of Mechanical Engineering. ISBN: 8-0214-3195-4.

[294] Donald E. Brown, Christopher L. Huntley, and Andrew R. Spillane. A parallel genetic heuristic for the quadratic assignment problem. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 406–415, 1989. In proceedings [1820].

[295] Will N. Browne and Charalambos Ioannides. Investigating scaling of an abstracted lcs utilising ternary and s-expression alphabets. In *GECCO'07: Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation*, pages 2759–2764, 2007. doi:10.1145/1274000.1274067. In proceedings [2037]. Online available at http://portal.acm.org/citation.cfm?id=1274000.1274067 [accessed 2007-08-01].

[296] Jason Brownlee. Learning classifier systems. Technical Report 070514A, Complex Intelligent Systems Laboratory, Centre for Information Technology Research, Faculty of Information and Communication Technologies, Swinburne University of Technology Melbourne, Australia, May 2007. Online available at http://www.ict.swin.edu.au/personal/jbrownlee/2007/TR24-2007.pdf [accessed 2008-04-03].

[297] Axel T. Brünger, Paul D. Adams, and Luke M. Rice. New applications of simulated annealing in x-ray crystallography and solution nmr. *Structure*, 15:325–336, 1997. Online available at http://atb.slac.stanford.edu/public/papers.php?sendfile=44 [accessed 2007-08-25].

[298] Bradley J. Buckham and Casey Lambert. Simulated annealing applications, November 1999. Seminar presentation: MECH620 Quantitative Analysis, Reasoning and Optimization Methods in CAD/CAM and Concurrent Engineering, Department of Mechanical Engineering, University of Victoria, Course Homepage: http://www.me.uvic.ca/~zdong/courses/mech620/ [accessed 2007-08-25], instructor: Dr. Zuomin Dong, Online available at http://www.me.uvic.ca/~zdong/courses/mech620/SA_App.PDF [accessed 2007-08-25].

[299] Lam Thu Bui and Sameer Alam, editors. *Multi-Objective Optimization in Computational Intelligence: Theory and Practice*. Premier Reference Source. Idea Group Publishing, May 30, 2008. ISBN: 1-5990-4498-6, 978-1-59904-498-9.

[300] Larry Bull. On using zcs in a simulated continuous double-auction market. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 1999)*, pages 83–90, 1999. In proceedings [142]. Online available at http://www.cs.bham.ac.uk/~wbl/biblio/gecco1999/GA-806.pdf [accessed 2007-09-12].

[301] Larry Bull, editor. *Applications of Learning Classifier Systems*. Studies in Fuzziness and Soft Computing. Springer, Berlin, Germany, April 2004. ISBN: 3-5402-1109-8, 978-3-54021-109-9. Partly online available at http://books.google.de/books?id=aBIjqGag5-kC [accessed 2008-04-01].

[302] Larry Bull and Jacob Hurst. Zcs redux. *Evolutionary Computation*, 10(2):185–205, 2002. ISSN: 1063-6560. Online available at http://www.cems.uwe.ac.uk/lcsg/papers/zcsredux.ps [accessed 2007-09-12].

[303] Larry Bull and Tim Kovacs, editors. *Foundations of Learning Classifier Systems*. Studies in Fuzziness and Soft Computing. Springer, September 1, 2005. ISBN: 3-5402-5073-5, 978-3-54025-073-9.

[304] Bernd Bullnheimer, Richard F. Hartl, and Christine Strauss. Applying the ant system to the vehicle routing problem. In *2nd International Conference On Metaheuristics (MIC'97)*. INRIA, 1997, Sophia-Antipolis, France. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.7946 [accessed 2008-10-27].

[305] Bernd Bullnheimer, Richard F. Hartl, and Christine Strauss. An improved ant system algorithm for the vehicle routing problem. *Annals of Operations Research*, 89(0):319–328, January 1999. ISSN: 0254-5330 (print), 1572-9338 (electronic). doi:10.1023/A:1018940026670. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.49.1415 and http://www.macs.hw.ac.uk/~dwcorne/Teaching/bullnheimer-vr.pdf [accessed 2008-10-27].

[306] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998. Online available at http://citeseer.ist.psu.edu/burges98tutorial.html and http://research.microsoft.com/~cburges/papers/SVMTutorial.pdf [accessed 2007-08-08].

[307] Luciana Buriol, Paulo M. França, and Pablo Moscato. A new memetic algorithm for the asymmetric traveling salesman problem. *Journal of Heuristics*, 10(5):483–506, September 2004. ISSN: 1381-1231 (Print) 1572-9397 (Online). doi:10.1023/B:HEUR.0000045321.59202.52. Online available at http://www.springerlink.com/content/w617486q60mphg88/fulltext.pdf and http://citeseer.ist.psu.edu/544761.html [accessed 2007-09-12].

[308] Edmund Burke, Steven Matt Gustafson, Graham Kendall, and Natalio Krasnogor. Advanced population diversity measures in genetic programming. In *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature - PPSN VII*, page 341 ff., 2002. In proceedings [867]. Online available at http://citeseer.ist.psu.edu/529057.html [accessed 2007-09-07].

[309] Edmund K. Burke, Steven Matt Gustafson, and Graham Kendall. Survey and analysis of diversity measures in genetic programming. In *Genetic and Evolutionary Computation Conference*, pages 716–723, 2002. In proceedings [1245]. Online available at http://www.gustafsonresearch.com/research/publications/gecco-diversity-2002.pdf and http://citeseer.ist.psu.edu/505881.html [accessed 2008-07-22].

[310] Edmund K. Burke, Steven Matt Gustafson, Graham Kendall, and Natalio Krasnogor. Is increasing diversity in genetic programming beneficial? an analysis of the effects on fitness. In *Congress on Evolutionary Computation*, pages 1398–1405, 2003. In proceedings [1803]. Online available at http://www.gustafsonresearch.com/research/publications/cec-2003.pdf and http://citeseer.ist.psu.edu/700607.html [accessed 2008-07-22].

[311] Edmund K. Burke, Steven Matt Gustafson, and Graham Kendall. Diversity in genetic programming: An analysis of measures and correlation with fitness. *IEEE Transactions on Evolutionary Computation*, 8(1):47–62, February 2004. ISSN: 1089-778X. doi:10.1109/TEVC.2003.819263. Online available at http://www.gustafsonresearch.com/research/publications/gustafson-ieee2004.pdf and http://citeseer.ist.psu.edu/burke04diversity.html [accessed 2008-07-22].

[312] Arthur W. Burks. From eniac to the stored-program computer: Two revolutions in computers. In N. Metropolis, J. Howlett, and Gian-Carlo Rota, editors, *A History of Computing in the Twentieth Century: A Collection of Essays with Introductory Essay and Indexes*, pages 311–344. Academic Press, New York, November 1980. ISBN: 978-0-12491-650-0. Papers from the Los Alamos International Research Conference on the History of Computing, 1976.

[313] William J. Butera. *Programming a Paintable Computer*. PhD thesis, MIT Media Lab, Massachusetts Institute of Technology, MA, USA, February 2002. Online available at http://www-swiss.ai.mit.edu/projects/amorphous/papers/butera-phd.pdf [accessed 2008-07-27].

[314] Martin V. Butz. *Anticipatory Learning Classifier Systems*. Genetic Algorithms and Evolutionary Computation. Springer, January 25 2002. ISBN: 0-7923-7630-7, 978-0-79237-630-9.

[315] Martin V. Butz. *Rule-Based Evolutionary Online Learning Systems: A Principled Approach to LCS Analysis and Design*. Studies in Fuzziness and Soft Computing. Springer, December 22 2005. ISBN: 3-5402-5379-3, 978-3-54025-379-2.

[316] Martin V. Butz, Kumara Sastry, and Davi E. Goldberg. Tournament selection in xcs. IlliGAL Report 2002020, Illinois Genetic Algorithms Laboratory (IlliGAL), Department of Computer Science, Department of General Engineering, University of Illinois at Urbana-Champaign, Urbana, Illinois, USA, July 2002. Online available at http://citeseer.ist.psu.edu/534844.html and http://www.illigal.uiuc.edu/pub/papers/IlliGALs/2002020.ps.Z [accessed 2008-03-20].

[317] Jon Byous. Java technology: The early years. Technical Report, Sun Microsystems, Inc., ca. 1998. Online available at http://java.sun.com/features/1998/05/birthday.html [accessed 2007-07-03].

[318] Elizabeth Byrnes and Jan Aikins, editors. *Proceedings of the The Sixth Conference on Innovative Applications of Artificial Intelligence (IAAI 1994)*, July 13–August 4, 1994, Convention Center, Seattle, Washington, USA. ISBN: 978-0-92928-062-2. See http://www.aaai.org/Conferences/IAAI/iaai94.php [accessed 2007-09-06].

C

[319] Stefano Cagnoni, A. B. Dobrzeniecki, Ricardo Poli, and J. C. Yanch. Genetic algorithm-based interactive segmentation of 3D medical images. *Image and Vision Computing*, 17(12):881–895, October 1999. Online available at http://citeseer.ist.psu.edu/cagnoni99genetic.html [accessed 2007-07-29].

[320] Stefano Cagnoni, Riccardo Poli, Yun Li, George D. Smith, David Corne, Martin J. Oates, Emma Hart, Pier Luca Lanzi, Egbert J. W. Boers, Ben Paechter, and Terence C. Fogarty, editors. *Real-World Applications of Evolutionary Computing, Proceedings of EvoWorkshops 2000: EvoIASP, EvoSCONDI, EvoTel, EvoSTIM, EvoROB, and EvoFlight*, volume 1803/200 of *Lecture Notes in Computer Science (LNCS)*, April 17, 2000, Edinburgh, Scotland, UK. Springer, Berlin / Heidelberg. ISBN: 3-5406-7353-9, 978-3-54067-353-8. doi:10.1007/3-540-45561-2.

[321] Stefano Cagnoni, Jens Gottlieb, Emma Hart, Martin Middendorf, and Günther R. Raidl, editors. *Applications of Evolutionary Computing, Proceedings of EvoWorkshops 2002: EvoCOP, EvoIASP, EvoSTIM/EvoPLAN*, volume 2279/-1/2002 of *Lecture Notes in Computer Science (LNCS)*, April 3–4, 2002, Kinsale, Ireland. Springer, Berlin / Heidelberg. ISBN: 3-5404-3432-1, 978-3-54043-432-0. doi:10.1007/3-540-46004-7.

[322] Stefano Cagnoni, Evelyne Lutton, and Gustavo Olague, editors. *Genetic and Evolutionary Computation for Image Processing and Analysis*, volume 8 of *EURASIP Book Series on Signal Processing and Communications*. Hindawi Publishing Corporation, 410 Park Avenue, 15th Floor, #287 pmb, New York, NY 10022, USA, February 2008. ISBN: 978-9-77454-001-1. Online available at http://www.hindawi.com/books/9789774540011.pdf [accessed 2008-05-17].

[323] Sébastien Cahon, Nordine Melab, and El-Ghazali Talbi. Paradiseo: A framework for the reusable design of parallel and distributed metaheuristics. *Journal of Heuristics*, 10(3):357–380, May 2004. ISSN: 1381-1231 (Print) 1572-9397 (Online). doi:10.1023/B:HEUR.0000026900.92269.ec.

[324] Tao Cai, Feng Pan, and Jie Chen. Adaptive particle swarm optimization algorithm. In *Proceedings of Fifth World Congress on Intelligent Control and Automation, 2004. WCICA 2004*, volume 3, pages 2245–2247, June 15–19, 2004. ISBN: 0-7803-8273-0. doi:10.1109/WCICA.2004.134198.

[325] Osvaldo Cairó, Luis Enrique Sucar, and Francisco J. Cantu, editors. *Proceedings of the MICAI 2000: Advances in Artificial Intelligence, Mexican International Conference on Artificial Intelligence*, volume 1793/2000 of *Lecture Notes in Computer Science (LNCS), Subseries Lecture Notes in Artificial Intelligence (LNAI)*, April 11–14, 2000, Acapulco, México. Springer Berlin / Heidelberg. ISBN: 3-5406-7354-7, 978-3-54067-354-5. doi:10.1007/10720076.

[326] Edgar H. Callaway, Jr. *Wireless Sensor Networks: Architectures and Protocols*. AUERBACH, August 26, 2003. ISBN: 978-0-84931-823-8.

[327] Valérie Camps, Marie-Pierre Gelizes, and Pierre Glize. Une théorie des phénomènes globaux fondée sur des interactions locales. In Jean-Paul Barthès, Vincent Chevrier, and Christian Brassac, editors, *JFIADSMA'98: Actes des Sixième journées francophones pour l'Intelligence Artificielle Distribuée et les Systèmes Multi-Agents (IAD&SMA)*, November 18–20, 1998, Pont-è-Mousson, Nance, France. Editions Hermès, Paris, France. ISBN: 2-8660-1733-1, 978-2-86601-733-0. Online available at ftp://ftp.irit.fr/pub/IRIT/SMAC/DOCUMENTS/PUBLIS/JFIADSMA98.pdf [accessed 2008-10-08].

[328] Erick Cant'u-Paz. A summary of research on parallel genetic algorithms. IlliGAL report 95007, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, 1995. Online available at http://citeseer.ist.psu.edu/27505.html and ftp://ftp-illigal.ge.uiuc.edu/pub/papers/IlliGALs/95007.ps.Z [accessed 2007-08-13].

[329] Erick Cant'u-Paz. *Designing efficient and accurate parallel genetic algorithms*. PhD thesis, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, July 1999. Also IlliGAL report 99017. Online available at http://citeseer.ist.psu.edu/cantu-paz99designing.html [accessed 2008-04-06].

[330] Erick Cant'u-Paz. *Efficient and Accurate Parallel Genetic Algorithms*, volume 1 of *Genetic Algorithms and Evolutionary Computation*. Springer, December 15, 2000. ISBN: 978-0-79237-221-9, 0-7923-7221-2.

[331] Erick Cantú-Paz, editor. *Late Breaking papers at the Genetic and Evolutionary Computation Conference (GECCO-2002)*, July 9–13, 2002, The Roosevelt Hotel, 45th and Madison Avenue, New York, NY, USA. AAAI, 445 Burgess Drive, Menlo Park, CA 94025. See also [1245, 1572, 154].

[332] Erick Cantú-Paz and Chandrika Kamath. Using evolutionary algorithms to induce oblique decision trees. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'00)*, pages 1053–1060, 2000. In proceedings [2216]. Online available at https://computation.llnl.gov/casc/sapphire/dtrees/oc1.html and http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.28.7300 [accessed 2008-12-27].

[333] Erick Cantú-Paz, Martin Pelikan, and David E. Goldberg. Linkage problem, distribution estimation, and bayesian networks. *Evolutionary Computation*, 8(3):311–340, Fall 2000. ISSN: 1063-6560, 1530-9304. doi:10.1162/106365600750078808. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.16.236 [accessed 2008-10-18]. Also: IlliGAL Report 98013, November 1998, Illinois Genetic Algorithms Laboratory (IlliGAL), Department of General Engineering, University of Illinois at Urbana-Champaign, 117 Transportation Building, 104 S. Mathews Avenue, Urbana-Champaign, Illinois, 61801-2996, USA, online available at http://www.illigal.uiuc.edu/pub/papers/IlliGALs/98013.ps.Z and http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.45.213 [accessed 2008-10-18].

[334] Erick Cantú-Paz, James A. Foster, Kalyanmoy Deb, Lawrence Davis, Rajkumar Roy, Una-May O'Reilly, Hans-Georg Beyer, Russell K. Standish, Graham Kendall, Stewart W. Wilson, Mark Harman, Joachim Wegener, Dipankar Dasgupta, Mitchell A. Potter, Alan C. Schultz, Kathryn A. Dowsland, Natasa Jonoska, and Julian Francis Miller, editors. *Proceedings of Genetic and Evolutionary Computation - GECCO 2003, Genetic and Evolutionary Computation Conference, Part I*, volume 2723/2003 of *Lecture Notes in Computer Science (LNCS)*, July 12–16, 2003, The Holiday Inn Chicago – Mart Plaza, 350 N. Orleans St., Chicago, IL 60654, USA. Springer Berlin/Heidelberg. ISBN: 978-3-54040-602-0. doi:10.1007/3-540-45105-6. See also [335, 1573].

[335] Erick Cantú-Paz, James A. Foster, Kalyanmoy Deb, Lawrence Davis, Rajkumar Roy, Una-May O'Reilly, Hans-Georg Beyer, Russell K. Standish, Graham Kendall, Stewart W. Wilson, Mark Harman, Joachim Wegener, Dipankar Dasgupta, Mitchell A. Potter, Alan C. Schultz, Kathryn A. Dowsland, Natasa Jonoska, and Julian Francis Miller, editors. *Proceedings of Genetic and Evolutionary Computation - GECCO 2003, Genetic and Evolutionary Computation Conference, Part II*, volume 2724/2003 of *Lecture Notes in Computer Science (LNCS)*, July 12–16, 2003, The Holiday Inn Chicago – Mart Plaza, 350 N. Orleans St., Chicago, IL 60654, USA. Springer Berlin/Heidelberg. ISBN: 978-3-54040-603-7. doi:10.1007/3-540-45110-2. See also [334, 1573].

[336] Buyang Cao and Fred Glover. Tabu search and ejection chains-application to a node weighted version of the cardinality-constrained tsp. *Management Science*, 43(7):908–921, July 1997. ISSN: 0025-1909.

[337] Hongqing Cao, Jingxian Yu, and Lishan Kang. An evolutionary approach for modeling the equivalent circuit for electrochemical impedance spectroscopy. In *Proceedings*

of the 2003 Congress on Evolutionary Computation CEC2003, pages 1819–1825, 2003. In proceedings [1803].

[338] Mathieu S. Capcarrère, Alex Alves Freitas, Peter J. Bentley, Colin G. Johnson, and Jon Timmis, editors. *Advances in Artificial Life, Proceedings of the 8th European Conference, ECAL 2005*, volume 3630 of *Lecture Notes in Computer Science (LNCS), subseries Lecture Notes in Artificial Intelligence (LNAI)*, September 5–9, 2005, University of Kent, Canterbury, Kent (UK). Springer Verlag. ISBN: 3-5402-8848-1.

[339] Alain Cardon, Theirry Galinho, and Jean-Philippe Vacher. An agent based architecture for job-shop scheduling problem using the spirit of genetic algorithm. In *Proceedings of EUROGEN'99*, pages 12–19, 1999. In proceedings [1413]. Online available at http://www.mit.jyu.fi/eurogen99/papers/vacher12.ps [accessed 2008-06-07].

[340] Alain Cardon, Theirry Galinho, and Jean-Philippe Vacher. A multi-objective genetic algorithm in job shop scheduling problem to refine an agents' architecture. In *Proceedings of EUROGEN'99*, 1999. In proceedings [1413]. Online available at http://www.jeo.org/emo/cardon99.ps.gz [accessed http://citeseer.ist.psu.edu/244765.html]2008-04-05.

[341] Alain Cardon, Theirry Galinho, and Jean-Philippe Vacher. Using genetic algorithm in job-shop scheduling problem to constraints negociators' agents. In *Proceedings of EUROGEN'99*, pages 20–27, 1999. In proceedings [1413].

[342] Peter A. Cariani. Extradimensional bypass. *Biosystems*, 64:47–53, January 2002. doi:10.1016/S0303-2647(01)00174-5. Online available at http://dx.doi.org/10.1016/S0303-2647(01)00174-5 and 2008-11-01 [accessed .]

[343] Anthony Jack Carlisle. *Applying the Particle Swarm Optimizer to Non-Stationary Environments*. PhD thesis, Graduate Faculty of Auburn University, December 2002. Advisors: Gerry V. Dozier. Online available at http://antho.huntingdon.edu/publications/default.html [accessed 2007-08-19].

[344] Anthony Jack Carlisle and Gerry V. Dozier. Tracking changing extrema with adaptive particle swarm optimizer. In *Proceedings of the 5th Biannual World Automation Congress, WAC 2002*, volume 13, pages 265–270, June 9–13, 2002, Orlando, Florida, USA. doi:10.1109/WAC.2002.1049555. Online available at http://antho.huntingdon.edu/publications/default.html [accessed 2007-08-19].

[345] Charles W. Carroll. *An Operations Research Approach to the Economic Optimization of a Kraft Pulping Process*. PhD thesis, Institute of Paper Chemistry, Appleton, Wisconsin, USA, Georgia Institute of Technology, June 1959. Online available at http://hdl.handle.net/1853/5853 [accessed 2008-11-15].

[346] Charles W. Carroll. The created response surface technique for optimizing nonlinear, restrained systems. *Operations Research*, 9(2):169–184, March–April 1961. ISSN: 0030-364X. doi:10.1287/opre.9.2.169.

[347] Ted Carson and Russell Impagliazzo. Hill-climbing finds random planted bisections. In *Symposium on Discrete Algorithms, Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, volume 12, pages 903–909. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001, Washington, D.C., United States. ISBN: 0-8987-1490-7. Online available at http://portal.acm.org/citation.cfm?id=365411.365805 and http://citeseer.ist.psu.edu/carson01hillclimbing.html [accessed 2007-09-11].

[348] E. F. Carter. The generation and application of random numbers. *Forth Dimensions*, XVI(1 and 2), 1994, Oakland California.

[349] Richard A. Caruana and J. David Schaffer. Representation and hidden bias: Gray vs. binary coding for genetic algorithms. In John E. Laird, editor, *Machine Learning, Proceedings of 5th International Conference on Machine Learning*, pages 153–161, June 1988, Ann Arbor, Michigan, USA. Morgan Kaufmann, San Mateo, California. ISBN: 0-9346-1364-8.

[350] George Casella and Roger L. Berger. *Statistical Inference*. Duxbury Advanced Series. Dubury Thomson Learning / Duxbury Press, 511 Forest Lodge Road, Pa-

cific Grove, CA 93950, USA, second edition, June 18, 2001. ISBN: 0-5342-4312-6, 978-0-53424-312-8.

[351] George Casella and Roger L. Berger. *Statistical Inference*. Duxbury Thomson Learning, Pacific Grove, CA, USA, second edition, 2002. ISBN: 0-5342-4312-6.

[352] Mike Cattolico, editor. *GECCO'06: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, July 8–12, 2006, Renaissance Seattle Hotel, 515 Madison Street, Seattle, Washington 98104, USA. ACM Press, New York, NY, USA. ISBN: 1-5959-3186-4. ACM order number 910060.

[353] H. John Caulfield, Shu-Heng Chen, Heng-Da Cheng, Richard J. Duro, Vasant Honavar, Etienne E. Kerre, Mi Lu, Manuel Grana Romay, Timothy K. Shih, Dan Ventura, Paul P. Wang, and Yuanyuan Yang, editors. *Proceedings of the Sixth Joint Conference on Information Science (JCIS 2002), Section: The Fourth International Workshop on Frontiers in Evolutionary Algorithms (FEA 2002)*, March 8–13, 2002, Research Triangle Park, North Carolina, USA. JCIS / Association for Intelligent Machinery, Inc. ISBN: 0-9707-8901-7. Workshop held in conjunction with Sixth Joint Conference on Information Sciences.

[354] Daniel Joseph Cavicchio, Jr. *Adaptive Search using Simulated Evolution*. PhD thesis, The University of Michigan, College of Literature, Science, and the Arts, Computer and Communication Sciences Department, Ann Arbor, Michigan, USA, August 1970. Published as Technical Report. Chairman: John Henry Holland. Online available at http://hdl.handle.net/2027.42/4042 [accessed 2007-10-31], ID: bab9712.0001.001.

[355] Daniel Joseph Cavicchio, Jr. Reproductive adaptive plans. In *ACM'72: Proceedings of the ACM annual conference*, pages 60–70, 1972, Boston, Massachusetts, United States. ACM, New York, NY, USA. doi:10.1145/800193.805822. Online available at http://doi.acm.org/10.1145/800193.805822 [accessed 2008-09-25].

[356] Walter Cedeño and Dimitris K. Agrafiotis. Using particle swarms for the development of qsar models based on k-nearest neighbor and kernel regression. *Journal of Computer-Aided Molecular Design*, 17(2–4):255–263, February 2003. ISSN: 0920-654X (Print) 1573-4951 (Online). doi:10.1023/A:1025338411016. Online available at http://www.springerlink.com/content/j523757110202636/ and http://www.dimitris-agrafiotis.com/ [accessed 2007-08-21].

[357] Carlos Artemio Ceollo Coello. A short tutorial on evolutionary multiobjective optimization. In *First International Conference on Evolutionary Multi-Criterion Optimization*, pages 21–40, 2001. In proceedings [2331]. Online available at http://citeseer.ist.psu.edu/coellocoello01short.html [accessed 2007-07-29].

[358] Carlos Artemio Ceollo Coello. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art. *Computer Methods in Applied Mechanics and Engineering*, 191(11–12):1245–1287, January 4, 2002. ISSN: 0045-7825. doi:10.1016/S0045-7825(01)00323-1. Online available at http://dx.doi.org/10.1016/S0045-7825(01)00323-1 [accessed 2009-02-28].

[359] Carlos Artemio Ceollo Coello. A comprehensive survey of evolutionary-based multiobjective optimization techniques. *Knowledge and Information Systems*, 1(3):269–308, August 1999. Online available at http://www.lania.mx/~ccoello/EMOO/informationfinal.ps.gz and http://citeseer.ist.psu.edu/coello98comprehensive.html [accessed 2007-08-25].

[360] Carlos Artemio Ceollo Coello. An updated survey of evolutionary multiobjective optimization techniques: State of the art and future trends. In *1999 Congress on Evolutionary Computation*, pages 3–13, 1999. In proceedings [69]. Online available at http://citeseer.ist.psu.edu/coellocoello99updated.html [accessed 2007-08-25].

[361] Carlos Artemio Ceollo Coello, Gary B. Lamont, and David A. van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems*, volume 5 of *Genetic and Evolutionary Computation*. Kluwer Academic Publishers / Springer, second edition, 1st ed: 2002, 2nd ed: 2007. ISBN: 0-3064-6762-3, 978-0-30646-762-2, 0-3873-3254-5,

978-0-38733-254-3, 978-0-38736-797-2. doi:10.1007/978-0-387-36797-2. Series Editor: David E. Goldberg and John R. Koza. Partly online available at http://books.google.de/books?id=sgX_Cst_yTsC [accessed 2008-10-20].

[362] Vinton Cerf, Yogen Dalal, and Carl Sunshine. Specification of internet transmission control program (december 1974 version). Request for Comments (RFC) 675, Network Working Group, December 1974. Online available at http://tools.ietf.org/html/rfc675 [accessed 2008-06-13].

[363] V. Černý. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45(1):41–51, January 1985. ISSN: 0022-3239 (Print) 1573-2878 (Online). doi:10.1007/BF00940812. Communicated by S. E. Dreyfus. Online available at http://www.springerlink.com/content/gt0743622913gg33/fulltext.pdf [accessed 2008-03-26]. Also: Technical Report, Comenius University, Bratislava, Czechoslovakia, 1982.

[364] Abdulkadir Çevik. A new formulation for web crippling strength of cold-formed steel sheeting using genetic programming. *Journal of Constructional Steel Research*, 63 (7):867–883, July 2007. doi:10.1016/j.jcsr.2006.08.012.

[365] Deepti Chafekar, Jiang Xuan, and Khaled Rasheed. Constrained multi-objective optimization using steady state genetic algorithms. In *Proceedings of Genetic and Evolutionary Computation GECCO 2003*, pages 813–824, 2003. In proceedings [334]. Online available at http://citeseer.ist.psu.edu/591279.html and http://www.cs.uga.edu/~khaled/papers/385.pdf [accessed 2007-07-28].

[366] Jinxiang Chai and Songde Ma. Robust epipolar geometry estimation using genetic algorithm. *Pattern Recognition Letters*, 19(9):829–838, July 1998. ISSN: 0167-8655. doi:10.1016/S0167-8655(98)00032-4. See also [367]. Online available at http://dx.doi.org/10.1016/S0167-8655(98)00032-4 [accessed 2008-03-22].

[367] Jinxiang Chai and Songde Ma. Robust epipolar geometry estimation using genetic algorithm. In Roland T. Chin and Ting-Chuen Pong, editors, *ACCV, Proceedings of Computer Vision - ACCV'98, Third Asian Conference on Computer Vision, Volume I*, volume 1351 of *Lecture Notes in Computer Science (LNCS)*, pages 272–279. Springer, January 1998, Hong Kong, China. ISBN: 3-5406-3930-6. doi:10.1016/S0167-8655(98)00032-4. See also [366].

[368] Lance D. Chambers, editor. *Practical Handbook of Genetic Algorithms: Applications*, volume I of *Practical Handbook of Genetic Algorithms*. Chapman & Hall/CRC Press, Inc., Boca Raton, FL, USA, 2nd ed: 2000 edition, 1995. ISBN: 0-8493-2519-6, 1-5848-8240-9, 978-1-58488-240-4.

[369] Lance D. Chambers, editor. *Practical Handbook of Genetic Algorithms: New Frontiers*, volume II of *Practical Handbook of Genetic Algorithms*. CRC Press, Inc., Boca Raton, FL, USA, 1995. ISBN: 0-8493-2529-3. Partly online available at http://books.google.de/books?id=9RCE3pgj9K4C [accessed 2008-07-19].

[370] Lance D. Chambers, editor. *Practical Handbook of Genetic Algorithms: Complex Coding Systems*, volume III of *Practical Handbook of Genetic Algorithms*. CRC Press, Inc., Boca Raton, FL, USA, December 23, 1998. ISBN: 0-8493-2539-0, 978-0-84932-539-7.

[371] Edmon Hok-Man Chan. Design and implementation of a high speed cable-based planar parallel manipulator. Master's thesis, University of Waterloo, Ontario, Canada, 2005. Advisor: Dr. Amir Khajepour. Online available at http://etd.uwaterloo.ca/etd/hmechan2005.pdf [accessed 2008-06-14].

[372] Felix T.S. Chan and Manoj Kumar Tiwari, editors. *Swarm Intelligence – Focus on Ant and Particle Swarm Optimization*. I-Tech Education and Publishing, Vienna, Austria, December 2007. ISBN: 978-3-90261-309-7. Online available at http://s.i-techonline.com/Book/Swarm-Intelligence/Swarm-Intelligence.zip [accessed 2008-08-22].

[373] Sandeep Chandran and R. Russell Rhinehart. Heuristic random optimizer-version ii. In *Proceedings of the American Control Conference*, volume 4, pages 2589–2594. IEEE, Piscataway NJ, US, 2002. doi:10.1109/ACC.2002.1025175.

[374] C. S. Chang and Chung Min Kwan. Evaluation of evolutionary algorithms for multi-objective train schedule optimization. In *AI 2004: Advances in Artificial Intelligence*, volume 3339/2004 of *Lecture Notes in Artificial Intelligence, subseries of Lecture Notes in Computer Science (LNCS)*, pages 803–815. Springer-Verlag, 2004. ISBN: 978-3-54024-059-4. doi:10.1007/b104336.

[375] Vira Chankong and Yacov Y. Haimes. *Multiobjective Decision Making Theory and Methodology*. North-Holland, Elsevier, Dover Publications, New York, January 1983. ISBN: 978-0-44400-710-0, 978-0-48646-289-9.

[376] Abraham Charnes and William Wager Cooper. *Management Models and Industrial Applications of Linear Programming*. John Wiley & Sons Inc, New York, December 1961. ISBN: 0-4711-4850-4, 978-0-47114-850-0.

[377] Abraham Charnes, William Wager Cooper, and R. O. Ferguson. Optimal estimation of executive compensation by linear programming. *Management Science*, 1(2):138–151, January 1955. doi:10.1287/mnsc.1.2.138.

[378] Neela Chattoraj and Jibendu Sekhar Roy. Application of genetic algorithm to the optimization of microstrip antennas with and without superstrate. *Mikrotalasna Revija (Microwave Review)*, 2(6), November 2006. ISSN: 1450-5835. Online available at http://www.mwr.medianis.net/pdf/Vol12No2-06-NChattoraj.pdf [accessed 2008-09-02].

[379] Pravir K. Chawdry, Rajkumar Roy, and Raj K. Pant, editors. *Soft Computing in Engineering Design and Manufacturing*. Springer-Verlag, August 1998. ISBN: 978-3-54076-214-0. Partly online available at http://books.google.de/books?id=mxcP1mSjOlsC [accessed 2008-06-07].

[380] Sin Man Cheang, Kwong Sak Leung, and Kin Hong Lee. Genetic parallel programming: design and implementation. *Evolutionary Computation*, 14(2):129–156, Summer 2006. ISSN: 1063-6560, 1530-9304. doi:10.1162/evco.2006.14.2.129.

[381] Artem V. Chebotko. *Programming Languages (Course Material CSC3200)*. Wayne State University (WSU), Detroit, USA, 2006. Online available at http://www.cs.wayne.edu/~artem/ [accessed 2007-07-03].

[382] Guillaume Chelius, Eric Fleury, and Thierry Mignon. Lower and upper bounds for minimum energy broadcast and sensing problems in sensor networks. *International Journal of Parallel, Emergent and Distributed Systems*, to be published, 2005. Also: technical report, January 2004: Rapport de recherche de l'INRIA - Rhone-Alpes, Equipe: ARES. Online available at http://www.inria.fr/rrrt/rr-5072.html [accessed 2007-08-01]. See also [383].

[383] Guillaume Chelius, Eric Fleury, and Thierry Mignon. Lower and upper bounds for minimum energy broadcast and sensing problems in sensor networks. In *ICPADS'05: Proceedings of the 11th International Conference on Parallel and Distributed Systems - Workshops (ICPADS'05)*, pages 88–92, 2005. IEEE Computer Society, Washington, DC, USA. ISBN: 0-7695-2281-5. doi:10.1109/ICPADS.2005.181. See also [382].

[384] Kumar Chellapilla. Evolutionary programming with tree mutations: Evolving computer programs without crossover. In *Genetic Programming 1997: Proceedings of the Second Annual Conference GP-97*, pages 432–438, 1997. In proceedings [1208].

[385] Arbee L. P. Chen, Jui-Shang Chiu, and Frank S.C. Tseng. Evaluating aggregate operations over imprecise data. *IEEE Transactions on Knowledge and Data Engineering*, 8(2):273–284, April 1996. ISSN: 1041-4347. Online available at http://make.cs.nthu.edu.tw/alp/alp_paper/Evaluating%20aggregate%20operations%20over%20imprecise%20data.pdf [accessed 2007-09-12].

[386] Jian Chen, Yong Guan, and Udo Pooch. Customizing a geographical routing protocol for wireless sensor networks. In *ITCC'05: Proceedings of the International Conference*

on Information Technology: Coding and Computing (ITCC'05) - Volume II, volume 2, pages 586–591, April 4–6, 2005. IEEE Computer Society, Washington, DC, USA. ISBN: 0-7695-2315-3. doi:10.1109/ITCC.2005.118.

[387] Jong-Chen Chen and Michael Conrad. A multilevel neuromolecular architecture that uses the extradimensional bypass principle to facilitate evolutionary learning. *Physica D: Nonlinear Phenomena*, 75(1–3):417–437, August 1, 1994. doi:10.1016/0167-2789(94)90295-X. Also: proceedings of the NATO advanced research workshop and EGS topical workshop on Chaotic advection, tracer dynamics and turbulent dispersion, Sereno di Gavi, Italy, editors: Armando Babiano, Antonello Provenzale, Angelo Vulpiani, H. Flaschka, and F. H. Busse. Online available at http://dx.doi.org/10.1016/0167-2789(94)90295-X [accessed 2008-11-01].

[388] Shu-Heng Chen, editor. *Evolutionary Computation in Economics and Finance*, volume 100 of *Studies in Fuzziness and Soft Computing*. Physica-Verlag Heidelberg, August 5, 2002. ISBN: 3-7908-1476-8.

[389] Bastien Chevreux. Genetische algorithmen zur molekülstrukturoptimierung. Master's thesis, Universität Heidelberg/Fachhochschule Heilbronn, Deutsches Krebsforschungszentrum Heidelberg, July 2001. Online available at http://chevreux.org/diplom/diplom.html [accessed 2007-07-29].

[390] Altannar Chinchuluun, Panos M. Pardalos, Athanasios Migdalas, and Leonidas Pitsoulis, editors. *Pareto Optimality, Game Theory and Equilibria*, volume 17 of *Springer Optimization and Its Applications*. Springer, New York, USA, 2008. ISBN: 0-3877-7246-4, 978-0-38777-246-2, 978-0-38777-247-9. doi:10.1007/978-0-387-77247-9. Partly online available at http://books.google.de/books?id=kNqHxU3Tc2YC [accessed 2009-02-20].

[391] Raymond Chiong, editor. *Nature-Inspired Algorithms for Optimisation*, volume 193 of *Studies in Computational Intelligence*. Springer, April 30 2009. ISBN: 978-3-64200-266-3. doi:10.1007/978-3-642-00267-0.

[392] Sung-Soon Choi, Kyomin Jung, and Jeong Han Kim. Phase transition in a random nk landscape model. In *Proceedings of Genetic and Evolutionary Computation Conference, GECCO 2005*, pages 1241–1248, 2005. doi:10.1145/1068009.1068212. In proceedings [202] Session: Genetic Algorithms. Online available at http://web.mit.edu/kmjung/Public/NK%20gecco%20final.pdf and http://doi.acm.org/10.1145/1068009.1068212 [accessed 2009-02-26]. See also [393].

[393] Sung-Soon Choi, Kyomin Jung, and Jeong Han Kim. Phase transition in a random nk landscape model. *Artificial Intelligence*, 172(2–3):179–203, February 2008. ISSN: 0004-3702. doi:10.1016/j.artint.2007.06.002. Online available at http://dx.doi.org/10.1016/j.artint.2007.06.002 [accessed 2009-02-26]. See also [392].

[394] Noam Chomsky. Three models for the description of language. *IEEE Transactions on Information Theory*, 2(3):113–124, September 1956. Online available at http://www.chomsky.info/articles/195609--.pdf [accessed 2007-09-14].

[395] Noam Chomsky. *Syntactic structures*. 's-Gravenhage: Mouton & Co., 1957. Online available at http://books.google.de/books?id=a6a_b-CXYAkC and http://web.uni-marburg.de/dsa//Direktor/Rabanus/pdf/Syntactic_Structures.pdf [accessed 2007-09-14].

[396] Noam Chomsky and Marcel P. Schützenberger. The algebraic theory of context-free languages. In P. Braffort and D. Hirschberg, editors, *Computer Programming and Formal Systems*, pages 118–161. North-Holland, Amsterdam, 1963.

[397] Chee-Yee Chong and S.P. Kumar. Sensor networks: evolution, opportunities, and challenges. *Proceedings of the IEEE*, 91(8):1247–1256, August 2003. doi:10.1109/JPROC.2003.814918.

[398] Fuey Sian Chong and William B. Langdon. Java based distributed genetic programming on the internet. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, page 1229, 1999. In proceedings [142]. Online available

at http://libra.msra.cn/paperDetail.aspx?id=589775 and http://citeseer.ist.psu.edu/chong99java.html [accessed 2007-08-14].

[399] Samantha Y. Chong and Maryjane Tremayne. Combined optimization using cultural and differential evolution: application to crystal structure solution from powder diffraction data. *Chemical Communication*, 39:4078–4080, October 2006. ISSN: 1359-7345. Online available at http://www.rsc.org/publishing/journals/CC/article.asp?doi=b609138e [accessed 2007-08-13].

[400] Hosung Choo, Adrian Hutani, Luiz Cezar Trintinalia, and Hao Ling. Shape optimisation of broadband microstrip antennas using genetic algorithm. *Electronics Letters*, 36(25):2057–2058, December 7, 2000. ISSN: 0013-5194. CODEN: ELLEAK. INSPEC Accession Number: 6798475. doi:10.1049/el:20001452.

[401] Heather Christensen, Roger L. Wainwright, and Dale A. Schoenefeld. A hybrid algorithm for the point to multipoint routing problem. In *Proceedings of the 1997 ACM/SIGAPP Symposium on Applied Computing (SAC'97)*, pages 263–268, February 28–March 2, 1997, Hyatt St. Claire, 302 S. Market St., San Jose, CA 95113, USA. ACM Press, New York, NY, USA. ISBN: 0-8979-1850-9. doi:10.1145/331697.331751. Online available at http://euler.mcs.utulsa.edu/~rogerw/papers/Heather-PMP.pdf and http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.22.8559 [accessed 2008-08-28].

[402] Henning Christiansen. Syntax, semantics, and implementation strategies for programming languages with powerful abstraction mechanisms. In *Proceedings of 18th Hawaii International Conference on System Sciences*, volume 2, pages 57–66, 1985. Also Datalogiske skrifter 1, Department of Computer Science, Roskilde University, 1985.

[403] Henning Christiansen. Parsing and compilation of generative languages. Technical Report 3, Department of Computer Science, Roskilde University, 1986. abridged version: [404].

[404] Henning Christiansen. Recognition of generative languages. In *Proceedings of Programs as Data Objects Workshop*, volume 217 of *Lecture Notes in Computer Science (LNCS)*, pages 63–81. Springer-Verlag, October 1985, Copenhagen, Denmark. ISBN: 0-3871-6446-4. Abridged version of [403].

[405] Henning Christiansen. Programming as language development. Technical Report 15, Department of Computer Science, Roskilde University, 1988. Ph.D. thesis (summary).

[406] Henning Christiansen. The syntax and semantics of extensible languages. Technical Report 14, Department of Computer Science, Roskilde University, 1988.

[407] Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58(2):345–363, April 1936.

[408] Alonzo Church. A note on the Entscheidungsproblem. *Journal of Symbolic Logic*, 1 (1):40–41, March 1936.

[409] Christopher D. Clack, editor. *Advanced Research Challenges in Financial Evolutionary Computing (ARC-FEC) Workshop 2008*, July 12, 2008, Renaissance Atlanta Hotel Downtown, 590 West Peachtree Street NW, Atlanta, Georgia 30308 USA. ACM Press, New York, NY, USA. ISBN: 978-1-60558-131-6. Part of GECCO 2008, see also [1117] and http://www.cs.ucl.ac.uk/financialcomputing/gecco_arcfec.htm [accessed 2008-07-21].

[410] Bill Clancey and Dan Weld, editors. *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96*, August 4–8, 1996, Portland, Oregon, USA. AAAI Press/The MIT Press. ISBN: 0-2625-1091-X. 2 volumes, see http://www.aaai.org/Conferences/AAAI/aaai96.php [accessed 2007-09-06], [2], and http://www.aaai.org/Conferences/IAAI/iaai96.php [accessed 2007-09-06].

[411] David E. Clark, editor. *Evolutionary Algorithms in Molecular Design*, volume 8 of *Methods and Principles in Medicinal Chemistry*. Wiley-VCH, Weinheim and New

York, September 11 2000. ISBN: 3-5273-0155-0, 978-3-52730-155-3. Series editors: Raimund Mannhold, Hugo Kubinyi, and Hendrik Timmerman.

[412] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In Dexter Kozen, editor, *Logics of Programs, Workshop*, volume 131/1982 of *Lecture Notes in Computer Science (LNCS)*, pages 52–71, May 1981, Yorktown Heights, New York, USA. Springer-Verlag, London, UK. ISBN: 3-5401-1212-X, 978-3-54011-212-9. doi:10.1007/BFb0025774. Published in 1982.

[413] Edmund M. Clarke, Jr., Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, January 2000. ISBN: 0-2620-3270-8, 978-0-26203-270-4. Partly online available at http://books.google.de/books?id=Nmc4wEaLXFEC [accessed 2008-10-02].

[414] Janet Clegg, James Alfred Walker, and Julian Frances Miller. A new crossover technique for cartesian genetic programming. In *GECCO'07: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation*, pages 1580–1587, 2007. In proceedings [2038]. Online available at http://doi.acm.org/10.1145/1276958.1277276 and http://www.cs.bham.ac.uk/~wbl/biblio/gecco2007/docs/p1580.pdf [accessed 2007-11-01].

[415] Maurice Clerc. *Particle Swarm Optimization*. ISTE Publishing Company, February 24, 2006. ISBN: 1-9052-0904-5, 978-1-90520-904-0.

[416] Manuel Clergue, Philippe Collard, Marco Tomassini, and Leonardo Vanneschi. Fitness distance correlation and problem difficulty for genetic programming. In *GECCO'02: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 724–732, 2002. In proceedings [1245]. Online available at http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/gecco2002/gecco-2002-14.pdf and http://www.cs.bham.ac.uk/~wbl/biblio/gecco2002/GP072.pdf [accessed 2008-07-20]. See also [2103].

[417] Gary A. Cleveland and Stephen F. Smith. Using genetic algorithms to schedule flow shop releases. In *ICGA, Proceedings of the third International Conference on Genetic Algorithms*, pages 160–169, 1989. In proceedings [1820].

[418] Dave Cliff and Susi Ross. Adding temporary memory to zcs. *Adaptive Behavior*, 3(2): 101–150, Fall 1994. ISSN: 1059-7123. Online available at ftp://ftp.informatics.sussex.ac.uk/pub/reports/csrp/csrp347.ps.Z [accessed 2007-09-12].

[419] João Climaco, editor. *Proceedings of the 11th International Conference on Multiple Criteria Decision Making: Multicriteria Analysis (MCDM'1994)*, August 1–6, 1994, Coimbra, Portugal. Springer. ISBN: 978-3-54062-074-7. Published in May 1997.

[420] *The gnutella protocol specification v0.4*. Clip2 Distributed Search Services, August 10, 2000. Online available at http://www.stanford.edu/class/cs244b/gnutella_protocol_0.4.pdf [accessed 2008-06-24]. Document Revision 1.2, June 15, 2001 online available at http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf [accessed 2008-06-24].

[421] P. D. Coddington and A. J. Newell. Japara – a java parallel random number generator library for high-performance computing. In *Proceedings of the 6th International Workshop on Java for Parallel and Distributed Computing IPDPS 2004*, page 156a, April 26, 2004, Santa Fe, New Mexico. IEEE Computer Society, Los Alamitos, CA, USA. ISBN: 0-7695-2132-0. doi:10.1109/IPDPS.2004.1303143.

[422] Carlos A Coello Coello, Arturo Hernández Aguirre, and Eckart Zitzler, editors. *Proceedings of the Third International Conference on Evolutionary Multi-Criterion Optimization (EMO'05)*, volume 3410 of *Lecture Notes in Computer Science (LNCS)*, March 9–11, 2005, Guanajuato, México. Springer-Verlag, Berlin. ISBN: 978-3-54024-983-2.

[423] Carlos Artemio Coello Coello. An updated survey of ga-based multiobjective optimization techniques. *ACM Computing Surveys*, 32(2):109–143, 2000. ISSN: 0360-0300. doi:10.1145/358923.358929. Online available at

http://citeseer.ist.psu.edu/coello98updated.html and http://portal.acm.org/citation.cfm?id=358923.358929 [accessed 2007-07-28].

[424] Carlos Artemio Coello Coello and Gary B. Lamont, editors. *Applications of Multi-Objective Evolutionary Algorithms*, volume 1 of *Advances in Natural Computation*. World Scientific Publishing Co., December 2004. ISBN: 978-9-81256-106-0, 9-8125-6106-4. Partly online available at http://books.google.de/books?id=viWm0k9meOcC [accessed 2008-07-20].

[425] Carlos Artemio Coello Coello, Alvaro de Albornoz, Luis Enrique Sucar, and Osvaldo Cairó Battistutti, editors. *Proceedings of the MICAI 2002: Advances in Artificial Intelligence, Second Mexican International Conference on Artificial Intelligence*, volume 2313/2002 of *Lecture Notes in Computer Science (LNCS), Subseries Lecture Notes in Artificial Intelligence (LNAI)*, April 22–26, 2002, Mérida, Yucatán, México. Springer Berlin / Heidelberg. ISBN: 3-5404-3475-5, 978-3-54043-475-7. doi:10.1007/3-540-46016-0.

[426] James P. Cohoon, S. U. Hegde, Worthy Neil Martin, and D. Richards. Punctuated equilibria: a parallel genetic algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, pages 148–154, 1987. In proceedings [857].

[427] David W. Coit and Fatema Baheranwala. Solution of stochastic multi-objective system reliability design problems using genetic algorithms. In *Proceedings of the European safety and reliability conference (ESREL)*, pages 391–398, June 2005, Gdansk, Poland. Online available at http://www.engr.rutgers.edu/ie/research/working_paper.html [accessed 2007-07-29].

[428] Pierre Collet, Cyril Fonlupt, Jin-Kao Hao, Evelyne Lutton, and Marc Schoenauer, editors. *Selected Papers of the 5th International Conference on Artificial Evolution, Evolution Artificielle, EA 2001*, volume 2310 of *Lecture Notes in Computer Science (LNCS)*, October 29–31, 2001, Le Creusot, France. Springer Berlin/Heidelberg. ISBN: 3-5404-3544-1. doi:10.1007/3-540-46033-0. Published in 2002.

[429] Pierre Collet, Marco Tomassini, Marc Ebner, Steven Matt Gustafson, and Anikó Ekárt, editors. *Proceedings of the 9th European Conference Genetic Programming, EuroGP 2006*, volume 3905/2006 of *Lecture Notes in Computer Science (LNCS)*, April 10-12, 2006, Budapest, Hungary. Springer Berlin/Heidelberg. ISBN: 3-5403-3143-3.

[430] J. J. Collins and Malachy Eaton. Genocodes for genetic algorithms. In *Proceedings of the 2nd International Mendel Conference on Genetic Algorithms, MENDEL 1997*, 1997. In proceedings [2078]. Online available at http://citeseer.ist.psu.edu/116539.html and http://www.csis.ul.ie/staff/jjcollins/mendel97.ps.gz [accessed 2008-03-14].

[431] Robert J. Collins and David R. Jefferson. Representations for artificial organisms. In *From animals to animats: Proceedings of the first international conference on simulation of adaptive behavior*, pages 382–390, 1990. In proceedings [1397].

[432] Robert J. Collins and David R. Jefferson. Selection in massively parallel genetic algorithms. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 249–256, 1991. In proceedings [170]. Online available at http://citeseer.ist.psu.edu/349839.html [accessed 2007-11-30].

[433] Robert J. Collins and David R. Jefferson. Antfarm: Towards simulated evolution. In *Artificial Life II*, pages 579–601, 1992. In proceedings [1248]. Online available at http://citeseer.ist.psu.edu/collins91antfarm.html and http://en.scientificcommons.org/108082 [accessed 2007-08-05].

[434] Francesc Comellas and G. Giménez. Genetic programming to design communication algorithms for parallel architectures. *Parallel Processing Letters (PPL)*, 8(4):549–560, December 1998. ISSN: 0129-6264, 1793-642X. doi:10.1142/S0129626498000547. Online available at http://www.cs.bham.ac.uk/

~wbl/biblio/gp-html/Comellas_1998_GPD.html and http://citeseer.ist.psu.edu/comellas98genetic.html [accessed 2007-09-14].

[435] Brian Connolly. Genetic algorithms – survival of the fittest: Natural selection with windows forms. *MSDN Magazin*, August 2004. Online available at http://msdn.microsoft.com/de-de/magazine/cc163934(en-us).aspx [accessed 2008-06-24].

[436] Michael Conrad. Bootstrapping on the adaptive landscape. *Biosystems*, 11:81–84, August 1979. doi:10.1016/0303-2647(79)90009-1. Online available at http://hdl.handle.net/2027.42/23514 and http://dx.doi.org/10.1016/0303-2647(79)90009-1 [accessed 2008-11-02].

[437] Michael Conrad. *Adaptability: The Significance of Variability from Molecule to Ecosystem*. Plenum Press, New York, USA, March 31, 1983. ISBN: 0-3064-1223-3, 978-0-30641-223-3.

[438] Michael Conrad. Molecular computing. In Marshall C. Yovits, editor, *Advances in Computers*, volume 31, pages 235–324. Academic Press Professional, Inc., San Diego, CA, USA, October 1990. ISBN: 0-1201-2131-X, 978-0-12012-131-1.

[439] Michael Conrad. The geometry of evolution. *Biosystems*, 24(1):61–81, 1990. doi:10.1016/0303-2647(90)90030-5. Online available at http://dx.doi.org/10.1016/0303-2647(90)90030-5 [accessed 2008-11-02].

[440] Michael Conrad. Towards high evolvability dynamics. In Gertrudis Van de Vijver, Stanley N. Salthe, and Manuela Delpos, editors, *Evolutionary Systems – Biological and Epistemological Perspectives on Selection and Self-Organization*, pages 33–43. Kluwer, 1998. ISBN: 0-7923-5260-2, 978-0-79235-260-0.

[441] Susan Coombs and Lawrence Davis. Genetic algorithms and communication link speed design: constraints and operators. In *Proceedings of the Second International Conference on Genetic Algorithms and their application*, pages 257–260, 1987. In proceedings [857]. Partly online available at http://www.questia.com/PM.qst?a=o&d=97597557 [accessed 2008-08-29].

[442] Emilio Corchado, Juan M. Corchado, and Ajith Abraham, editors. *Innovations in Hybrid Intelligent Systems – Proceedings of the 2nd International Workshop on Hybrid Artificial Intelligence Systems (HAIS 2007)*, volume 44/2008 of *Advances in Soft Computing*, November 2007, University of Salamanca, Salamanca, Spain. Springer Berlin/Heidelberg. ISBN: 3-5407-4971-3, 978-3-54074-971-4. Library of Congress Control Number: 2007935489. doi:10.1007/978-3-540-74972-1.

[443] Emilio Corchado, Ajith Abraham, and Witold Pedrycz, editors. *Proceedings of the Third International Workshop on Hybrid Artificial Intelligence Systems (HAIS 2008)*, volume 5271/2008 of *Lecture Notes in Artificial Intelligence (LNAI), subseries of Lecture Notes in Computer Science (LNCS)*, September 24–26, 2008, Burgos, Spain. Springer Berlin/Heidelberg. ISBN: 3-5408-7655-3, 978-3-54087-655-7. Library of Congress Control Number: 2008935394. doi:10.1007/978-3-540-87656-4.

[444] Arthur L. Corcoran and Sandip Sen. Using real-valued genetic algorithms to evolve rule sets for classification. In *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, volume 1, pages 120–124, 1994. doi:10.1109/ICEC.1994.350030. In proceedings [1411]. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.55.1864 [accessed 2008-12-27].

[445] Oscar Cordón, Francisco Herrera, and Luciano Sánchez. Evolutionary learning processes for data analysis in electrical engineering applications. In D. Quagliarella, J. Périaux, C. Poloni, and G. Winter, editors, *Genetic Algorithms and Evolution Strategy in Engineering and Computer Science*, chapter 10, pages 205–224. John Wiley and Sons, Chichester, 1998. In collection [1413]. Online available at http://citeseer.ist.psu.edu/64737.html and ftp://decsai.ugr.es/pub/arai/tech_rep/ga-fl/eurogen97.ps.Z [accessed 2007-08-25].

[446] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms*. MIT Electrical Engineering and Computer Science. The MIT Press and McGraw-Hill, second edition, August 2001. ISBN: 0-2625-3196-8, 978-0-26203-141-7. First edition June 1990.

[447] David Corne and Jonathan L. Shapiro, editors. *Proceedings of the Workshop on Artificial Intelligence and Simulation of Behaviour (AISB), International Workshop on Evolutionary Computing, Selected Papers*, volume 1305/1997 of *Lecture Notes in Computer Science (LNCS)*, April 7–8 1997, Manchester, UK. The Society for the Study of Artificial Intelligence and the Simulation of Behaviour (SSAISB), Springer Heidelberg/Berlin. ISBN: 3-5406-3476-2, 978-3-54063-476-8. doi:10.1007/BFb0027161.

[448] David Corne, Marco Dorigo, Fred Glover, Dipankar Dasgupta, Pablo Moscato, Riccardo Poli, and Kenneth V. Price, editors. *New Ideas in Optimization*. McGraw-Hill's Advanced Topics In Computer Science Series. McGraw-Hill Ltd., Maidenhead, UK, England, May 1999. ISBN: 0-0770-9506-5, 978-0-07709-506-2.

[449] David Corne, Zbigniew Michalewicz, Bob McKay, Gusz Eiben, David Fogel, Carlos Fonseca, Garrison Greenwood, Gunther Raidl, Kay Chen Tan, and Ali Zalzala, editors. *Proceedings of the IEEE Congress on Evolutionary Computation, CEC2005*, September 2–5, 2005, Edinburgh, Scotland, UK. IEEE Press, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA. ISBN: 0-7803-9363-5. Gusz Eiben ≡ Ágoston E. Eiben.

[450] David Wolfe Corne, Martin J. Oates, and George D. Smith, editors. *Telecommunications Optimization: Heuristic and Adaptive Techniques*. John Wiley & Sons, Ltd., September 2000. ISBN: 0-4719-8855-3, 978-0-47198-855-7, 978-0-47084-163-1. doi:10.1002/047084163X.

[451] F. Corno, G. Cumani, M. Sonza Reorda, and Giovanni Squillero. Efficent machine-code test-program induction. In *Proceedings of the 2002 Congress on Evolutionary Computation, CEC2002*, pages 1486–1491, 2002. In proceedings [703]. Online available at http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/corno_2002_emctpi.html and http://www.cad.polito.it/pap/db/cec2002.pdf [accessed 2008-09-17].

[452] Nelson Correa. An extension of earley's algorithm for s-attributed grammars. In *Proceedings of the fifth conference on European chapter of the Association for Computational Linguistics*, pages 299–302, 1991, Berlin, Germany. Association for Computational Linguistics, Morristown, NJ, USA. doi:10.3115/977180.977232. Online available at http://dx.doi.org/10.3115/977180.977232 and http://www.aclweb.org/anthology-new/E/E91/E91-1052.pdf [accessed 2007-09-115].

[453] Pablo Cortés Achedad, Luis Onieva Giménez, Jesús Muñuzuri Sanz, and José Guadix Martín. A revision of evolutionary computation techniques in telecommunications and an application for the network global planning problem. In *Success in Evolutionary Computation*, pages 239–262. Springer Berlin / Heidelberg, 2008. doi:10.1007/978-3-540-76286-7_11. In collection [2279].

[454] Pascal Côté, Tony Wong, and Robert Sabourin. Application of a hybrid multi-objective evolutionary algorithm to the uncapacitated exam proximity problem. In Edmund K. Burke and Michael Trick, editors, *Proceedings of the 5th International Conference on Practice and Theory of Automated Timetabling (PATAT 2004)*, pages 151–168, 2004. Online available at http://citeseer.ist.psu.edu/653221.html and http://www.asap.cs.nott.ac.uk/patat/patat04/151.pdf [accessed 2007-08-27].

[455] Carlos Cotta and Peter Cowling, editors. *Evolutionary Computation in Combinatorial Optimization, Proceedings of the 9th European Conference, EvoCOP 2009*, volume 5482/2009 of *Lecture Notes in Computer Science (LNCS), Subseries: SL 1 – Theoretical Computer Science and General Issues*, April 15–17, 2009, Tübingen, Germany. Springer, Berlin / Heidelberg. ISBN: 3-6420-1008-3, 978-3-64201-008-8. doi:10.1007/978-3-642-01009-5.

[456] Carlos Cotta and Jano I. van Hemert, editors. *Proceedings of the 7th European Conference on Evolutionary Computation in Combinatorial Optimization, EvoCOP 2007*, volume 4446/2007 of *Lecture Notes in Computer Science (LNCS)*, April 11–13, 2007, Valencia, Spain. Springer. ISBN: 978-3-54071-614-3.

[457] George F. Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems: Concepts and Design*. Pearson Education / Addison-Wesley Longman, fourth rev. edition, June 2005. ISBN: 0-3212-6354-5, 978-0-32126-354-4. Partly online available at http://books.google.de/books?id=d63sQPvBezgC [accessed 2008-11-25].

[458] Richard Courant. Variational methods for the solution of problems of equilibrium and vibrations. *Bulletin of the American Mathematical Society*, 49(1):1–23, 1943. ISSN: 1936-881X, 0002-9904. doi:10.1090/S0002-9904-1943-07818-4. Mathematical Reviews number (MathSciNet): MR0007838, Zentralblatt MATH identifier: 0063.00985 Online available at http://www.ams.org/bull/1943-49-01/S0002-9904-1943-07818-4/home.html and http://projecteuclid.org/euclid.bams/1183504922 [accessed 2008-11-15].

[459] Steven H. Cousins. Species diversity measurement: Choosing the right index. *Trends in Ecology and Evolution (TREE)*, 6(6):190–192, June 1991. ISSN: 0169-5347. doi:10.1016/0169-5347(91)90212-G. Online available at http://dx.doi.org/10.1016/0169-5347(91)90212-G [accessed 2008-11-10].

[460] David Roxbee Cox and Nancy Reid. *The Theory of the Design of Experiments*, volume 86 of *Monographs on Statistics and Applied Probability*. CRC Press / Taylor and Francis LLC, June 6, 2000. ISBN: 978-1-58488-195-7, 1-5848-8195-X. Catalogue no. C195X. Partly online available at http://books.google.com/books/crc_press?id=rCgKVbhJ0BYC [accessed 2008-08-18].

[461] Louis Anthony Cox, Jr., Lawrence Davis, and Yuping Qiu. Dynamic anticipatory routing in circuit-switched telecommunications networks. In *Handbook of Genetic Algorithms*, pages 124–143. Van Nostrand Reinhold, 1991. In collection [495].

[462] Nichael Lynn Cramer. A representation for the adaptive generation of simple sequential programs. In *Proceedings of the 1st International Conference on Genetic Algorithms and their Applications*, pages 183–187, 1985. In proceedings [856]. Online available at http://www.sover.net/~nichael/nlc-publications/icga85/index.html [accessed 2007-09-06].

[463] Raphael Crawford-Marks and Lee Spector. Size control via size fair genetic operators in the PushGP genetic programming system. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 733–739, 2002. In proceedings [1245]. Online available at http://citeseer.ist.psu.edu/608051.html and http://alum.hampshire.edu/~rpc01/gp234.pdf [accessed 2007-12-25].

[464] Ronald L. Crepeau. Genetic evolution of machine language software. In *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 121–134, 1995. In proceedings [1757]. Online available at http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/crepeau_1995_GEMS.html and http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.61.7001 [accessed 2008-09-17].

[465] Matej Črepinček, Marjan Mernik, Faizan Javed, Barrett R. Bryant, and Alan Sprague. Extracting grammar from programs: evolutionary approach. *SIGPLAN Notices*, 40(4):39–46, 2005. ISSN: 0362-1340. doi:10.1145/1064165.1064172. Online available at http://doi.acm.org/10.1145/1064165.1064174 [accessed 2007-09-09].

[466] H. Brown Cribbs, III and Robert Elliott Smith. What can i do with a learning classifier system? In *Industrial Applications of Genetic Algorithms*, pages 299–319. CRC Press, 1999. In collection [1093].

[467] Claudia Crosio, Francesco Cecconi, Paolo Mariottini, Gianni Cesareni, Sydney Brenner, and Francesco Amaldi. Fugu intron oversize reveals the presence of u15 snorna coding sequences in some introns of the ribosomal protein s3 gene. *Genome Research*,

6(12):1227–1231, December 1996. ISSN: 1088-9051. Online available at http://www.genome.org/cgi/content/abstract/6/12/1227 [accessed 2008-03-23].

[468] David Culler and Dr. David Tennenhouse. Largest tiny network yet – large-scale demonstration of self-organizing wireless sensor networks, August 2001, San Jose, California, USA. See http://webs.cs.berkeley.edu/800demo/ [accessed 2007-07-03].

[469] David Culler, Deborah Estrin, and Mani Srivastava. Guest editors' introduction: Overview of sensor networks. *Computer*, 37(8):41–49, August 2004. ISSN: 0018-9162. Online available at http://www.archrock.com/downloads/resources/IEEE-overview-2004.pdf [accessed 2007-09-15].

[470] Vincenzo Cutello, Giuseppe Narzisi, and Giuseppe Nicosia. A class of pareto archived evolution strategy algorithms using immune inspired operators for ab-initio protein structure prediction. In *Applications on Evolutionary Computing, Proceedings of EvoWorkkshops 2005*, pages 54–63, 2005. doi:10.1007/b106856. In proceedings [1767].

[471] Djurdje Cvijović and Jacek Klinowski. Taboo search: an approach to the multiple minima problem. *Science*, 267(5198):664–666, February 3, 1995. ISSN: 0036-8075. doi:10.1126/science.267.5198.664.

[472] Walling Cyre. Learning grammars with a modified classifier system. In *CEC'02: Proceedings of the Congress on Evolutionary Computation*, pages 1366–1371, 2002. In proceedings [703].

[473] Zbigniew J. Czech and Piotr Czarnas. Parallel simulated annealing for the vehicle routing problem with time windows. In *10th Euromicro Workshop on Parallel, Distributed and Network-based Processing (PDP'02)*, pages 376–383. IEEE Computer Society, January 9–11, 2002, Canary Islands, Spain. ISBN: 0-7695-1444-8. doi:10.1109/EMPDP.2002.994313. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.16.5766 [accessed http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.100.8805]2008-10-27.

D

[474] António Gaspar Lopes da Cunha. A multi-objective evolutionary algorithm for solving traveling salesman problems: Application to the design of polymer extruders. In *Proceedings of the 7th International Conference on Adaptive and Natural Computing Algorithms, ICANNGA 2005, Part II*, pages 189–193, 2005. In proceedings [1725].

[475] António Gaspar Lopes da Cunha and José António Colaço Gomes Covas. RPSGAe – reduced pareto set genetic algorithm: Application to polymer extrusion. In *Metaheuristics for Multiobjective Optimisation*, pages 221–249. Springer, 2004. In collection [766].

[476] António Gaspar Lopes da Cunha, José António Colaço Gomes Covas, and Pedro Oliveira. Optimization of polymer extrusion with genetic algorithms. *IMA Journal of Mathematics Applied in Business & Industry*, (9):267–277, 1998. Online available at http://imaman.oxfordjournals.org/cgi/reprint/9/3/267.pdf [accessed 2007-07-28].

[477] F. A. da Veiga. Structure discovery in medical databases: a conceptual clustering approach. *Artificial Intelligence in Medicine*, 8(5):473–491, 1996. Online available at http://dx.doi.org/10.1016/S0933-3657(96)00353-3 [accessed 2007-08-11].

[478] Gerard E. Dallal. *The Little Handbook of Statistical Practice*. Gerard E. Dallal, July 16, 2008. Online available at http://www.statisticalpracting.com/ [accessed 2008-08-15].

[479] Hai H. Dam, Hussein A. Abbass, and Chris Lokan. Dxcs: an xcs system for distributed data mining. In *Proceedings of Genetic and Evolutionary Computation Conference GECCO 2005*, pages 1883–1890, 2005. In proceedings [202] and also [480]. Online available at http://doi.acm.org/10.1145/1068009.1068326 [accessed 2007-09-12].

[480] Hai H. Dam, Hussein A. Abbass, and Chris Lokan. Dxcs: an xcs system for distributed data mining. Technical Report TR-ALAR-200504002, The Artificial Life and Adaptive Robotics Laboratory, School of Information Technology and Electrical Engineering, University of New South Wales, Northcott Drive, Campbell, Canberra, ACT 2600 Australia, 2005. ALAR Technical Report Series. Online available at http://www.itee.adfa.edu.au/~alar/techreps/200504002.pdf [accessed 2007-09-12]. See also [479].

[481] E. Damiani, Lakhmi C. Jain, and Robert J. Howlett, editors. *Proceedings of the Sixth International Conference on Knowledge-Based Intelligent Information & Engineering Systems, KES 2002*, September 16–18, 2002, Podere d'Ombriano, Crema, Italy. See http://www.bton.ac.uk/kes/kes2002/kes2002.html/ [accessed 2008-07-28].

[482] Martin Damsbo, Brian S. Kinnear, Matthew R. Hartings, Peder T. Ruhoff, Martin F. Jarrold, and Mark A. Ratner. Application of evolutionary algorithm methods to polypeptide folding: Comparison with experimental results for unsolvated ac-(ala-gly-gly)$_5$-lysh$^+$. *Proceedings of the National Academy of Science of the United States of America*, 101(19):7215–7222, May 2004. ISSN: 1091-6490. Online available at http://www.pnas.org/cgi/reprint/101/19/7215.pdf?ck=nck and http://www.pnas.org/content/vol101/issue19/ [accessed 2007-08-27].

[483] Ludwig Danzera and Victor Klee. Lengths of snakes in boxes. *Journal of Combinatorial Theory*, 2(3):258–265, May 1967. doi:10.1016/S0021-9800(67)80026-7.

[484] Richard B. Darlington. The wilcoxon signed-ranks test, September 1996. Online available at http://comp9.psych.cornell.edu/Darlington/wilcoxon/wilcox0.htm [accessed 2008-08-18].

[485] Charles Darwin. *On the Origin of Species*. John Murray, sixth edition, November 1859. Online available at http://www.gutenberg.org/etext/1228 [accessed 2007-08-05].

[486] Erasmus Darwin. *Zoönomia; or, the Organic Laws of Life*, volume I. J. Johnson, St. Paul's Church-Yard, London, UK, second corrected edition, 1796. Entered at Stationers' Hall. Online available at http://www.gutenberg.org/etext/15707 [accessed 2008-09-10].

[487] Indraneel Das and John E. Dennis. A closer look at drawbacks of minimizing weighted sums of objectives for pareto set generation in multicriteria optimization problems. *Structural optimization*, 14(1):63–69, August 1997. ISSN: 1615-147X (Print) 1615-1488 (Online). doi:10.1007/BF01197559. Online available at http://www.springerlink.com/content/q35140txv121u116/fulltext.pdf [accessed 2008-10-10].

[488] Swagatam Das, Amit Konar, and Uday K. Chakraborty. An efficient evolutionary algorithm applied to the design of two-dimensional iir filters. In *GECCO'05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 2157–2163, 2005. doi:10.1145/1068009.1068364. In proceedings [202]. Online available at http://doi.acm.org/10.1145/1068009.1068364 [accessed 2007-08-27].

[489] Yuval Davidor. Epistasis variance: Suitability of a representation to genetic algorithms. *Complex Systems*, 4(4):369–383, 1990. ISSN: 0891-2513.

[490] Yuval Davidor. A naturally occurring niche-species phenomenon – the model and first results. In *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 257–263, 1989. In proceedings [170].

[491] Yuval Davidor. Epistasis variance: A viewpoint on GA-hardness. In *Proceedings of the First Workshop on Foundations of Genetic Algorithms*, pages 23–35, 1990. In proceedings [1924].

[492] Yuval Davidor, Hans-Paul Schwefel, and Reinhard Männer, editors. *Parallel Problem Solving from Nature – PPSN III, International Conference on Evolutionary Computation. Proceedings of the Third Conference on Parallel Problem Solving from Nature*, volume 866/1994 of *Lecture Notes in Computer Science (LNCS)*, October 9–14, 1994, Jerusalem, Israel. Springer. ISBN: 3-5405-8484-6, 978-3-54058-484-1.

doi:10.1007/3-540-58484-6. See http://ls11-www.informatik.uni-dortmund.de/PPSN/ppsn3/ppsn3.html [accessed 2007-09-05].

[493] Rogelio Dávila, Mauricio Osorio, and Claudia Zepeda, editors. *Proceedings of the LoLaCOM06 Workshop on Logic, Language and Computation*, volume 220 of *CEUR Workshop Proceedings*, November 13–14, 2006, Instituto Tecnológico de Apizaco, Apizaco, Tlaxcala, México. CEUR-WS.org. Online available at http://CEUR-WS.org/Vol-220/ and ftp://sunsite.informatik.rwth-aachen.de/pub/publications/CEUR-WS/Vol-220.zip [accessed 2008-06-29]. See also [781].

[494] Lawrence Davis, editor. *Genetic Algorithms and Simulated Annealing*. Research Notes in Artificial Intelligence. Morgan Kaufmann Publishers Inc., Los Altos/San Francisco, CA, USA, October 1987. ISBN: 0-9346-1344-3, 978-0-93461-344-6.

[495] Lawrence Davis, editor. *Handbook of Genetic Algorithms*. Thomson Publishing Group / Van Nostrand Reinhold Company, New York, USA, January 1991. ISBN: 0-4420-0173-8, 978-0-44200-173-5.

[496] Martin Davis. *Engines of Logic*. W.W. Norton & Company, London, UK, 2000. ISBN: 0-3933-2229-7.

[497] Randall Davis and Jonathan King. An overview of production systems. Technical Report STAN-CS-75-524, Stanford Computer Science Department, Stanford University, Stanford Computer Science Department, October 1975. Online available at http://handle.dtic.mil/100.2/ADA019702 [accessed 2007-07-18].

[498] Randall Davis and Jonathan King. An overview of production systems. *Machine Intelligence*, 8, 1977. ISSN: 300332.

[499] Brian D. Davison and Khaled Rasheed. Effect of global parallelism on a steady state ga. In Erick Cantu-Paz and Bill Punch, editors, *Evolutionary Computation and Parallel Processing*, pages 167–170, 1999. Orlando, Florida, USA. Workshop part of GECCO 1999, see proceedings [142]. Online available at http://citeseer.ist.psu.edu/davison99effect.html [accessed http://www.cse.lehigh.edu/ brian/pubs/1999/cec99/pgado.pdf]2007-08-28.

[500] Richard Dawkins. The evolution of evolvability. In *ALIFE – Artificial Life: Proceedings of the Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems*, pages 201–220, 1987. In proceedings [1237]. Published in January 1989.

[501] Richard Dawkins. *The Selfish Gene*. Oxford University Press, first: 1976, second edition, October 1989. ISBN: 0-1928-6092-5, 978-0-19286-092-7, 0-1928-6092-5. Partly online available at http://books.google.de/books?id=WkHO9HI7koEC [accessed 2008-03-28].

[502] Richard Dawkins. *Climbing Mount Improbable*. Penguin Books (UK) and W.W. Norton & Company (USA), London, 1 edition, 1996. ISBN: 0-6708-5018-7, 0-1401-7918-6. ASIN: 0393039307, 0393316823.

[503] Charles D. Day. Application of an evolutionary algorithm to multivariate optimal allocation in stratified sample designs. In *SOI Tax Stats – Papers – 2006 American Statistical Association Conference, Joint Statistical Meeting 2006*. Tax Stats. Internal Revenue Service. United States Department of the Treasury, August 2006, Seattle, Washington. An Application of Genetic Algorithms to Multivariate Optimal Allocation in Stratified Sample Designs. Online available at http://www.irs.gov/pub/irs-soi/06asaday.pdf [accessed 2007-08-27].

[504] Sérgio Granato de Araújo, Aloysio de Castro Pinto Pedroza, and Antônio Carneiro de Mesquita Filho. Evolutionary synthesis of communication protocols. In *Proceedings of the 10th International Conference on Telecommunications (ICT'03)*, volume 2, pages 986–993. IEEE, February 23–March 1, 2003, Tahiti, French Polynesia. doi:10.1109/ICTEL.2003.1191573. Online available at http://www.gta.ufrj.br/ftp/gta/TechReports/AMP03a.pdf [accessed 2008-06-21]. See also [505, 506] for Portuguese versions.

[505] Sérgio Granato de Araújo, Aloysio de Castro Pinto Pedroza, and Antônio Carneiro de Mesquita Filho. Uma metodologia de projeto de protocolos de comunicação baseada em técnicas evolutivas. In *XX Simpósio Brasileiro de Telecomunicações (SBrT'03)*, October 5–8, 2003, Hotel Glória, Rio de Janeiro, Brasil. Online available at http://www.gta.ufrj.br/ftp/gta/TechReports/AMP03e.pdf [accessed 2008-06-21]. See also [506] and the English version [504].

[506] Sérgio Granato de Araújo, Antônio Carneiro de Mesquita Filho, and Aloysio de Castro Pinto Pedroza. Síntese de circuitos digitais otimizados via programação genética. In *XXX Seminário Integrado de Software e Hardware (SEMISH'03) in Proceedings of XXIII Congresso da Sociedade Brasileira de Computação*, volume III, pages 273–285, August 4–5, 2003, Campinas, SP, Brasil. Online available at http://www.gta.ufrj.br/ftp/gta/TechReports/AMP03d.pdf [accessed 2008-06-21]. See also [505] and the English version [504].

[507] Bart de Boer. Classifier systems: a useful approach to machine learning? Master's thesis, Leiden University, Rijksuniversiteit Leiden, Netherlands, August 1994. Internal Report Number IR94-02. Supervisors Ida Sprinkhuizen-Kuyper and Egbert Boers. Online available at citeseer.ist.psu.edu/deboer94classifier.html [accessed 2007-08-08].

[508] Hugo de Garis. Artificial embryology: The genetic programming of an artificial embryo. In *Dynamic, Genetic, and Chaotic Programming*, pages 373–393. John Wiley, 1992. In collection [1921]. Online available at http://citeseer.ist.psu.edu/99683.html [accessed 2007-08-01].

[509] Hugo de Garis. Evolving a replicator: The genetic programming of self reproduction in cellular automata. In *ECAL-93 Self organisation and life: from simple rules to global complexity*, pages 274–284, 1993. In proceedings [552]. Online available at http://citeseer.ist.psu.edu/degaris93evolving.html [accessed 2007-08-01].

[510] Edwin D. de Jong, Richard A. Watson, and Jordan B. Pollack. Reducing bloat and promoting diversity using multi-objective methods. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 11–18, 2001. In proceedings [1937]. Online available at http://citeseer.ist.psu.edu/440305.html and http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/jong_2001_gecco.html [accessed 2008-07-20].

[511] Gerdien de Jong. Evolution of phenotypic plasticity: patterns of plasticity and the emergence of ecotypes. *New Phytologist*, 166(1):101–118, April 2005. doi:10.1111/j.1469-8137.2005.01322.x. Online available at http://dx.doi.org/10.111109-11 [accessed .]

[512] Kenneth Alan De Jong. *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, Computer and Communication Sciences, University of Michigan, August 1975. Order number AAI7609381. Doctoral Committee: John Henry Holland, Larry K. Flanigan, Richard A. Volz, Berhard P. Zeigler. Online available at http://cs.gmu.edu/~eclab/kdj_thesis.html [accessed 2007-08-17].

[513] Kenneth Alan De Jong. On using genetic algorithms to search program spaces. In *Proceedings of the Second International Conference on Genetic algorithms and their application*, pages 210–216, 1987. In proceedings [857].

[514] Kenneth Alan De Jong. Learning with genetic algorithms: An overview. *Machine Learning*, 3(2-3):121–138, October 1988. ISSN: 0885-6125 (Print) 1573-0565 (Online). doi:10.1007/BF00113894. Online available at http://www.springerlink.com/content/mg877066r521w781/fulltext.pdf [accessed 2008-04-04].

[515] Kenneth Alan De Jong. *Evolutionary Computation: A Unified Approach*, volume 4 of *Complex Adaptive Systems: A Bradford Book*. MIT Press, Cambridge, Massachusetts, USA, February 2006. ISBN: 978-0-26204-194-2, 0-2620-4194-4, 8-1203-3002-1.

[516] Kenneth Alan De Jong and William M. Spears. Learning concept classification rules using genetic algorithms. In *Proceedings of the Twelth International Conference on*

Artificial Intelligence IJCAI-91, volume 2, 1991. In proceedings [1496]. Online available at http://citeseer.ist.psu.edu/dejong91learning.html and http://www.cs.uwyo.edu/~wspears/papers/ijcai91.pdf [accessed 2007-09-12].

[517] Kenneth Alan De Jong, William M. Spears, and Diana F. Gordon. Using genetic algorithms for concept learning. *Machine Learning*, 13:161–188, 1993. Online available at http://citeseer.ist.psu.edu/dejong93using.html and http://www.cs.uwyo.edu/~dspears/papers/mlj93.pdf [accessed 2007-09-12].

[518] Kenneth Alan De Jong, David B. Fogel, and Hans-Paul Schwefel. A history of evolutionary computation. In *Evolutionary Computation 1: Basic Algorithms and Operators*, chapter 6, pages 40–. Institute of Physics Publishing (IOP), 2000. Online available at http://books.google.de/books?id=4HMYCq9US78C [accessed 2008-06-11].

[519] Kenneth Alan De Jong, Riccardo Poli, and Jonathan E. Rowe, editors. *Foundations of Genetic Algorithms 7 (FOGA)*, September 4–6, 2002, Torremolinos, Spain. Morgan Kaufmann, San Mateo, CA, USA. ISBN: 0-1220-8155-2. Published 2003.

[520] Marina de la Cruz Echeandía, Alfonso Ortega de la Puente, and Manuel Alfonseca. Attribute grammar evolution. In José Mira and José R. Álvarez, editors, *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach, Part II*, volume 3562/2005 of *Lecture Notes in Computer Science (LNCS)*, pages 182–191. Springer Berlin / Heidelberg, August 2005. ISBN: 978-3-54026-319-7. doi:10.1007/b137296. Online available at http://arantxa.ii.uam.es/~alfonsec/docs/confint/iwinac05.pdf [accessed 2007-09-09].

[521] Alfonso Ortega de la Puente, Marina de la Cruz Echeandía, and Manuel Alfonseca. Christiansen grammar evolution: Grammatical evolution with semantics. *IEEE Transactions on Evolutionary Computation*, 11(1):77–90, February 2007. ISSN: 1089-778X. Online available at http://arantxa.ii.uam.es/~alfonsec/artint/ieeetec.pdf [accessed 2007-09-09].

[522] Jean-Baptiste Pierre Antoine de Monet, Chevalier de Lamarck. *Philosophie zoologique – ou Exposition des considérations relatives à l'histoire naturelle des Animaux; à la diversité de leur organisation et des facultés qu'ils en obtiennent; aux causes physiques qui maintiennent en eux la vie et donnent lieu aux mouvements qu'ils exécutant; enfin, à celles qui produisent les unes le sentiment, et les autres l'intelligence de ceux qui en sont doués.* Dentu / J. B. Baillière Liberaire, De L'Académie Royale de Médecine, Rue de l'École de Médecine 13, Paris, France / Harvard University, 1809, 1830. ISBN: 1-4121-1646-5, 978-1-41211-646-6. Online available at http://www.lamarck.cnrs.fr/ice/ice_book_detail.php?lang=en&type=text&bdd=lamarck&table=ouvrages_lamarck&bookId=29 and http://books.google.fr/books?id=L6qAG6ZPgj4C [accessed 2008-09-10] and http://books.google.fr/books?id=rlAKHKY8RboC [accessed 2008-09-10].

[523] Pierre-Simon Marquis de Laplace. *Théorie Analytique des Probabilités (Analytical Theory of Probability)*. Courcier, Imprimeur-Libraire pour les Mathématiques, quai des Augustins, no. 57, 1812. Première Partie. Online available at http://books.google.de/books?id=nQwAAAAAMAAJ [accessed 2007-08-27].

[524] Fabiano Luis de Sousa and Fernando Manuel Ramos. Function optimization using extremal dynamics. In *4th International Conference on Inverse Problems in Engineering: Theory and Practice*, May 26–31, 2002, Angra dos Reis, Rio de Janeiro, Brazil.

[525] Fabiano Luis de Sousa, Fernando Manuel Ramos, Pedro Pagione, and Roberto M. Girardi. New stochastic algorithm for design optimization. *AIAA journal*, 41(9): 1808–1818, 2003. ISSN: 0001-1452. CODEN: AIAJAH.

[526] Fabiano Luis de Sousa, Valeri Vlassov, and Fernando Manuel Ramos. Generalized extremal optimization: An application in heat pipe design. *Applied Mathematical Modelling*, 28(10):911–931, October 2004. ISSN: 0307-904X.

doi:doi:10.1016/j.apm.2004.04.004. Online available at http://dx.doi.org/10.1016/j.apm.2004.04.004 [accessed 2008-08-24].

[527] Francisco Fernandez de Vega. *Modelos de Programacion Genetica Paralela y Distribuida con aplicaciones a la Sintesis Logica en FPGAs*. PhD thesis, University of Extremadura, 2001. Version español. For english version see [528]. Online available at http://cum.unex.es/profes/profes/fcofdez/escritorio/investigacion/pgp/thesis/phd.html [accessed 2007-09-09].

[528] Francisco Fernandez de Vega. *Distributed Genetic Programming Models with Application to Logic Synthesis on FPGAs*. PhD thesis, University of Extremadura, 2001. Online available at http://cum.unex.es/profes/profes/fcofdez/escritorio/investigacion/pgp/thesis/phd.html [accessed 2007-09-09]. Spanish version: [527].

[529] Dominique de Werra and Alain Hertz. Tabu search techniques: A tutorial and an application to neural networks. *OR Spectrum*, 11:131–141, 1989. ISSN: 0171-6468 (Print) 1436-6304 (Online). Online available at http://www.springerlink.de/content/x25k97k0qx237553/fulltext.pdf [accessed 2008-03-27].

[530] Thomas L. Dean and Kathleen McKeown, editors. *Proceedings of the 9th National Conference on Artificial Intelligence, AAAI*, July 14–19, 1991, Anaheim, California, USA. AAAI Press/The MIT Press. ISBN: 0-2625-1059-6. 2 volumes, see http://www.aaai.org/Conferences/AAAI/aaai91.php [accessed 2007-09-06].

[531] D. M. Deaven and Kai-Ming Ho. Molecular geometry optimization with a genetic algorithm. *Physical Review Letters*, 75:288–291, July 1995. doi:10.1103/PhysRevLett.75.288. eprint arXiv:mtrl-th/9506004. Online available at http://adsabs.harvard.edu/abs/1995mtrl.th...6004D and http://prola.aps.org/abstract/PRL/v75/i2/p288_1 [accessed 2007-09-05].

[532] Kalyanmoy Deb. Genetic algorithms for multimodal function optimization. Master's thesis, The Clearinghouse for Genetic Algorithms, University of Alabama, Tuscaloosa, 1989. TCGA Report No. 89002.

[533] Kalyanmoy Deb. Solving goal programming problems using multi-objective genetic algorithms. In *Proceedings of Congress on Evolutionary Computation*, volume 1, pages 77–84, 1999. doi:10.1109/CEC.1999.781910. In proceedings [69].

[534] Kalyanmoy Deb. Evolutionary algorithms for multi-criterion optimization in engineering design. In *Evolutionary Algorithms in Engineering and Computer Science*, pages 135–161. John Wiley & Sons, Ltd, 1999. In collection [1412]. Online available at http://citeseer.ist.psu.edu/deb99evolutionary.html and http://www.lania.mx/~ccoello/EMOO/deb99.ps.gz [accessed 2007-08-25].

[535] Kalyanmoy Deb. An introduction to genetic algorithms. *Sadhana*, 24(4-5):293–315, 1999. Online available at http://www.iitk.ac.in/kangal/papers/sadhana.ps.gz [accessed 2007-07-28].

[536] Kalyanmoy Deb. Nonlinear goal programming using multi-objective genetic algorithms. *Journal of the Operational Research Society*, 52(3):291–302, March 2001. Online available at http://www.lania.mx/~ccoello/EMOO/deb01d.ps.gz and http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.6.6775 [accessed 2008-11-14].

[537] Kalyanmoy Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley Interscience Series in Systems and Optimization. John Wiley & Sons, Inc., New York, NY, USA, May 2001. ISBN: 978-0-47187-339-6. Partly online available at http://books.google.de/books?id=ll_Blt03u5IC [accessed 2009-03-05].

[538] Kalyanmoy Deb. Genetic algorithms for optimization. KanGAL Report 2001002, Kanpur Genetic Algorithms Laboratory (KanGAL), Kanpur, PIN 208 016, India, 2001. Online available at http://citeseer.ist.psu.edu/604908.html and http://www.iitk.ac.in/kangal/papers/isna.ps.gz [accessed 2007-07-28].

[539] Kalyanmoy Deb and David E. Goldberg. An investigation of niche and species formation in genetic function optimization. In *Proceedings of the third international conference on Genetic algorithms*, pages 42–50, 1989. In proceedings [1820].

[540] Kalyanmoy Deb and David E. Goldberg. Analyzing deception in trap functions. In *Foundations of Genetic Algorithms 2*, pages 93–108, 1993. In proceedings [2209].

[541] Kalyanmoy Deb and David E. Goldberg. Sufficient conditions for deceptive and easy binary functions. *Annals of Mathematics and Artificial Intelligence*, 10(4):385–408, December 1994. ISSN: 1012-2443 (Print) 1573-7470 (Online). doi:10.1007/BF01531277. Online available at http://www.springerlink.com/content/q180816712nl7283/fulltext.pdf [accessed 2008-10-17]. Also: IlliGAL report 92001, Illinois Genetic Algorithms Laboratory, University of Illinois, Urbana-Champaign.

[542] Kalyanmoy Deb, Samir Agrawal, Amrit Pratab, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In *Proceedings of the Parallel Problem Solving from Nature VI Conference*, pages 849–858, 2000. In proceedings [1830]. Online available at https://eprints.kfupm.edu.sa/17643/1/17643.pdf and http://citeseer.ist.psu.edu/deb00fast.html [accessed 2008-10-20]. See also [543]. Also: KanGAL Report No. 200001, online available at http://www.iitk.ac.in/kangal/papers/tech2000001.ps.gz [accessed 2008-10-20].

[543] Kalyanmoy Deb, Amrit Pratab, Samir Agrawal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, April 2002. doi:10.1109/4235.996017. Online available at http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=00996017 [accessed 2008-10-20]. See also [542].

[544] Kalyanmoy Deb, Riccardo Poli, Wolfgang Banzhaf, Hans-Georg Beyer, Edmund K. Burke, Paul J. Darwen, Dipankar Dasgupta, Dario Floreano, James A. Foster, Mark Harman, Owen Holland, Pier Luca Lanzi, Lee Spector, Andrea Tettamanzi, Dirk Thierens, and Andrew M. Tyrrell, editors. *Proceedings of Genetic and Evolutionary Computation - GECCO 2004, Genetic and Evolutionary Computation Conference, Part I*, volume 3102/2004 of *Lecture Notes in Computer Science (LNCS)*, June 26–30, 2004, Red Lion Hotel, 1415 5th Avenue, Seattle, WA 98101, USA. Springer Berlin/Heidelberg. ISBN: 978-3-54022-344-3. See also [545, 1113, 1574].

[545] Kalyanmoy Deb, Riccardo Poli, Wolfgang Banzhaf, Hans-Georg Beyer, Edmund K. Burke, Paul J. Darwen, Dipankar Dasgupta, Dario Floreano, James A. Foster, Mark Harman, Owen Holland, Pier Luca Lanzi, Lee Spector, Andrea Tettamanzi, Dirk Thierens, and Andrew M. Tyrrell, editors. *Proceedings of Genetic and Evolutionary Computation - GECCO 2004, Genetic and Evolutionary Computation Conference, Part II*, volume 3103/2004 of *Lecture Notes in Computer Science (LNCS)*, June 26–30, 2004, Red Lion Hotel, 1415 5th Avenue, Seattle, WA 98101, USA. Springer Berlin/Heidelberg. ISBN: 978-3-54022-343-6. See also [544, 1113, 1574].

[546] Kalyanmoy Deb, Ankur Sinha, and Saku Kukkonen. Multi-objective test problems, linkages, and evolutionary methodologies. In *GECCO'06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1141–1148, 2006, Seattle, Washington, USA. ACM, New York, NY, USA. ISBN: 1-5959-3186-4. doi:10.1145/1143997.1144179. In proceedings [352]. Online available at http://www.iitk.ac.in/kangal/papers/k2006001.pdf and http://doi.acm.org/10.1145/1143997.1144179 [accessed 2008-10-17]. Also: KanGAL Report Number 2006001.

[547] Rina Dechter and Richard Sutton, editors. *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI) and Fourteenth Conference on Innovative Applications of Artificial Intelligence (IAAI)*, July 28–August 1, 2002, Edmonton, Alberta, Canada. AAAI Press. See http://www.aaai.org/Conferences/AAAI/aaai02.php [accessed 2007-09-06] and http://www.aaai.org/Conferences/IAAI/iaai02.php [accessed 2007-09-06].

[548] Michael Defoin Platel, Manuel Clergue, and Philippe Collard. Maximum homologous crossover for linear genetic programming. In *Genetic Programming: 6th European*

Conference, pages 29–48, 2003. In proceedings [1786]. Online available at http://www.i3s.unice.fr/~clergue/siteCV/publi/eurogp_03.pdf [accessed 2007-12-01].

[549] Michael Defoin Platel, Sébastien Vérel, Manuel Clergue, and Philippe Collard. From royal road to epistatic road for variable length evolution algorithm. In *Evolution Artificielle, 6th International Conference*, pages 3–14, 2003. doi:10.1007/b96080. In proceedings [1283]. Online available at http://arxiv.org/abs/0707.0548 and http://www.i3s.unice.fr/~verel/publi/ea03-fromRRtoER.pdf [accessed 2007-12-01].

[550] Marcel Dekker. *Introduction to Set Theory*. CRC, revised, and expanded, third edition, June 22, 1999. ISBN: 0-8247-7915-0.

[551] Tina Dell'Armi, Wolfgang Faber, Giuseppe Ielpa, Nicola Leone, and Gerald Pfeifer. Aggregate functions in disjunctive logic programming semantics, complexity, and implementation in dlv. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI) 2003*, pages 847–852, 2003. Online available at http://citeseer.ist.psu.edu/643941.html and http://www.icons.rodan.pl/publications/%5BDellArmi2003%5D.pdf [accessed 2007-09-12]. In proceedings [844].

[552] J. L. Denenbourg, S. Goss, G. Nicolis, H. Bersini, and R. Dagonnier, editors. *ECAL-93 Self organisation and life: from simple rules to global complexity, Proceedings of the Second European Conference on Artificial Life*, May 24–26, 1993, Bruxelles, Belgium. Free University of Brussels.

[553] Jean-Louis Deneubourg and Simon Goss. Collective patterns and decision-making. *Ethology, Ecology & Evolution*, 1(4):295–311, December 1989.

[554] Jean-Louis Deneubourg, Jacques M. Pasteels, and J. C. Verhaeghe. Probabilistic behaviour in ants: a strategy of errors? *Journal of Theoretical Biology*, 105(2):259–271, 1983. doi:10.1016/S0022-5193(83)80007-1. Online available at http://dx.doi.org/10.1016/S0022-5193(83)80007-1 [accessed 2008-03-27].

[555] Berna Dengiz, Fulya Altiparmak, and Alice E. Smith. Local search genetic algorithm for optimal design of reliable networks. *IEEE Transactions on Evolutionary Computation*, 1(3):179–188, September 1997. Online available at http://citeseer.ist.psu.edu/dengiz97local.html and http://www.eng.auburn.edu/~aesmith/publications/journal/ieeeec.pdf [accessed 2008-08-01].

[556] Luc Devroye. *Non-Uniform Random Variate Generation*. Springer-Verlag, New York, 1986. ISBN: 0-3879-6305-7, 3-5409-6305-7. Online available at http://cg.scs.carleton.ca/~luc/rnbookindex.html [accessed 2007-07-05].

[557] Patrik D'Haeseleer. Context preserving crossover in genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, volume 1, pages 256–261, 1994. In proceedings [1411]. Online available at http://citeseer.ist.psu.edu/dhaeseleer94context.html [accessed 2008-06-15].

[558] Sanjeev Dhir, K. John Morrow Jr, R. Russell Rhinehart, and Theodore Wiesner. Dynamic optimization of hybridoma growth in a fed-batch bioreactor. *Biotechnology and Bioengineering*, 67(2):197–205, January 2000. ISSN: 1097-0290. doi:10.1002/(SICI)1097-0290(20000120)67:2<197::AID-BIT9>3.0.CO;2-W. Online available at http://www3.interscience.wiley.com/cgi-bin/abstract/71003651/ [accessed 2007-08-26].

[559] Gianni Di Caro. *Ant Colony Optimization and its application to adaptive routing in telecommunication networks*. PhD thesis, Applied Sciences, Polytechnic School, Université Libre de Bruxelles, Brussels, Belgium, September 2004. Advisor: Marco Dorigo. Partly online available at http://www.idsia.ch/~gianni/Papers/thesis-abstract.pdf [accessed 2008-07-26] and http://www.idsia.ch/~gianni/Papers/routing-chapters.pdf [accessed 2008-07-26].

[560] Gianni Di Caro and Marco Dorigo. Antnet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research (JAIR)*, 9:317–365, December 1998. ISSN: 11076-9757. Submitted 5/98; published 12/98. Online available

at http://citeseer.ist.psu.edu/dicaro98antnet.html and http://www.jair.org/media/544/live-544-1748-jair.pdf [accessed 2008-07-26].

[561] Gianni Di Caro and Marco Dorigo. Two ant colony algorithms for best-effort routing in datagram networks. In *Proceedings of the Tenth IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS'98)*, pages 541–546. IASTED/ACTA Press, October 1–4, 1998, Las Vegas, USA. Online available at http://citeseer.ist.psu.edu/dicaro98two.html and http://www.idsia.ch/~gianni/Papers/PDCS98.ps.gz [accessed 2008-07-26].

[562] Dirk Dickmanns, Jürgen Schmidhuber, and Andreas Winklhofer. Der genetische algorithmus: Eine implementierung in prolog. Fortgeschrittenenpraktikum, Institut für Informatik, Lehrstuhl Professor Radig, Technische Universität München, Munich, Germany, 1987. Online available at http://www.idsia.ch/~juergen/geneticprogramming.html [accessed 2007-11-01].

[563] Thomas Dietterich and William Swartout, editors. *Proceedings of the 8th National Conference on Artificial Intelligence, AAAI*, July 29–August 3, 1990, Boston, Massachusetts, USA. AAAI Press/The MIT Press. ISBN: 0-2625-1057-X. 2 volumes, see http://www.aaai.org/Conferences/AAAI/aaai90.php [accessed 2007-09-06].

[564] Tom Dietterich. Overfitting and undercomputing in machine learning. *ACM Computing Surveys (CSUR)*, 27(3):326–327, 1995. ISSN: 0360-0300. doi:10.1145/212094.212114. Online available at http://doi.acm.org/10.1145/212094.212114 and http://citeseer.ist.psu.edu/dietterich95overfitting.html [accessed 2007-09-13].

[565] Jason Digalakis and Konstantinos Margaritis. A parallel memetic algorithm for solving optimization problems. In *Proceedings of the 4th Metaheuristics International Conference*, pages 121–125, 2001. In proceedings [1721]. Online available at http://citeseer.ist.psu.edu/digalakis01parallel.html and http://www.it.uom.gr/people/digalakis/digiasfull.pdf [accessed 2007-09-12].

[566] Jason Digalakis and Konstantinos Margaritis. Performance comparison of memetic algorithms. *Journal of Applied Mathematics and Computation*, 158:237–252, October 2004. Online available at http://citeseer.ist.psu.edu/458892.html and http://www.complexity.org.au/ci/draft/draft/digala02/digala02s.pdf [accessed 2007-09-12].

[567] Edsger Wybe Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959. Online available at http://www-m3.ma.tum.de/twiki/pub/MN0506/WebHome/dijkstra.pdf [accessed 2008-06-24].

[568] Edsger Wybe Dijkstra. Guarded commands, nondeterminacy and formal derivation of programs. *Communications of the ACM*, 18(8):453–458, August 1975. ISSN: 0001-0782. doi:10.1145/360933.360975. Online available at http://www.cs.utexas.edu/users/EWD/ewd04xx/EWD472.PDF and http://doi.acm.org/10.1145/360933.360975 [accessed 2008-07-23].

[569] Werner Dilger. *Einführung in die Künstliche Intelligenz*. Chemnitz University of Technology, Faculty of Computer Science, Chair of Artificial Intelligence (Künstliche Intelligenz), April 2006. Lecture notes for the lectures artificial intelligence. Online available at http://www.tu-chemnitz.de/informatik/KI/skripte.php [accessed 2007-08-06].

[570] Yiping Ding, Ethan D. Bolker, and Arjun Kumar. Performance implications of hyperthreading. In *Proceedings of 29th International Computer Measurement Group Conference*, pages 21–29. Computer Measurement Group, December 7–12, 2003, Dallas, Texas, USA.

[571] Paolo Dini. Section 1: Science, new paradigms. subsection 1: A scientific foundation for digital ecosystems. In Francesco Nachira, Paolo Dini, Andrea Nicolai, Marion Le Louarn, and Lorena Rivera Lèon, editors, *Digital Business Ecosystems*. The European Commission in association and with the support of the FP6 projects DBE and

OPAALS, Luxembourg: Office for Official Publications of the European Communities, 2007. ISBN: 9-2790-1817-5. Online available at http://www.digital-ecosystems.org/book/Section1.pdf [accessed 2008-11-09]. See also: Keynote Talk "Structure and Outlook of Digital Ecosystems Research", IEEE Digital Ecosystems Technologies Conference, DEST07, Cairns, Australia, February 20–23, 2007, online available at http://www.ieee-dest.curtin.edu.au/2007/PDini.pdf [accessed 2008-11-09] and [572].

[572] Paolo Dini. *Digital Business Ecosystems (DBE), Contract nr. 507953 – Workpackage 18: Socio-economic perspectives of sustainability and dynamic specification of behaviour in Digital Business Ecosystems – Deliverable 18.4: Report on self-organisation from a dynamical systems and computer science viewpoint*, Information Society Technologies, March 5, 2007. Online available at http://files.opaals.org/DBE/deliverables/Del_18.4_DBE_Report_on_self-organisation_from_a_dynamical_systems_and_computer_viewpoint.pdf [accessed 2008-11-09].

[573] L. C. Dinneen and B. C. Blakesley. Algorithm as 62: A generator for the sampling distribution of the mann-whitney u statistic. *Applied Statistics*, 22(2):269–273, 1973. ISSN: 0035-9254.

[574] Fred M. Discenzo, Dukki Chung, and Kenneth A. Loparo. Power scavenging enables maintenance-free wireless sensor nodes. In *Proceedings of NECSI International Conference on Complex Systems*, June 25–30, 2006, New England Complex Systems Institute, Marriott Boston Quincy, Boston, MA, USA. Online available at http://necsi.org/events/iccs6/viewpaper.php?id=188 [accessed 2007-08-01].

[575] Peter W. G. Dittrich, Jens Ziegler, and Wolfgang Banzhaf. Artificial chemistries – a review. *Artificial Life*, 7(3):225–275, 2001. Online available at http://citeseer.ist.psu.edu/509901.html and http://www.cs.mun.ca/~banzhaf/papers/alchemistry_review_MIT.pdf [accessed 2008-05-01].

[576] Andrej Dobnikar, Nigel C. Steele, David W. Pearson, and Rudolf F. Albrecht, editors. *Proceedings of the 4^{th} International Conference on Artificial Neural Nets and Genetic Algorithms, ICANNGA*, 1999, Portoroz, Slovenia. Springer. ISBN: 978-3-21183-364-3, 3-2118-3364-1.

[577] Karl Doerner, Manfred Gronalt, Richard F. Hartl, Marc Reimann, Christine Strauss, and Michael Stummer. Savings ants for the vehicle routing problem. In *Proceedings of the Applications of Evolutionary Computing on EvoWorkshops 2002*, pages 11–20, 2002. doi:10.1007/3-540-46004-7_2. In proceedings [321]. Also: Report Series SFB "Adaptive Information Systems and Modelling in Economics and Management Science", Nr. 63, December 2001, Vienna University of Economics and Business Administration, Augasse 2–6, 1090 Wien, Austria, Dokument ID: oai:epub.wu-wien.ac.at:epub-wu-01_1f1, online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.6.8882 and http://epub.wu-wien.ac.at/dyn/virlib/wp/mediate/epub-wu-01_1f1.pdf?ID=epub-wu-01_1f1 [accessed 2008-10-27].

[578] Pedro Domingos and Michael Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29(2-3):103–130, November 1997. ISSN: 0885-6125 (Print) 1573-0565 (Online). Online available at http://citeseer.ist.psu.edu/domingos97optimality.html and http://www.ics.uci.edu/~pazzani/Publications/mlj97-pedro.pdf [accessed 2007-08-11].

[579] Marco Dorigo. Message-based bucket brigade: An algorithm for the apportionment of credit problem. In Yves Kodratoff, editor, *EWSL-91: Proceedings of the European working session on learning on Machine learning*, pages 235–244, 1991, Porto, Portugal. Springer-Verlag New York, Inc., New York, NY, USA. ISBN: 0-3875-3816-X. Number 482. Online available at http://citeseer.ist.psu.edu/dorigo91messagebased.html [accessed 2008-04-04].

[580] Marco Dorigo and Hugues Bersini. A comparison of Q-learning and classifier systems. In *Proceedings of From Animals to Animats, Third International Conference on Simulation of Adaptive Behavior*, pages 248–255, 1994, Brighton, United Kingdom. ISBN: 0-2625-3122-4. Online available at http://citeseer.ist.psu.edu/dorigo94comparison.html [accessed 2008-04-03].

[581] Marco Dorigo and Christian Blum. Ant colony optimization theory: a survey. *Theoretical Computer Science*, 344(2-3):243–278, 2005. ISSN: 0304-3975. Online available at http://code.ulb.ac.be/dbfiles/DorBlu2005tcs.pdf [accessed 2007-08-05].

[582] Marco Dorigo and Luca Maria Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, April 1997. Online available at http://citeseer.ist.psu.edu/153.html and http://www.idsia.ch/~luca/acs-ec97.pdf [accessed 2007-08-19].

[583] Marco Dorigo and Thomas Stützle. *Ant Colony Optimization*. Bradford Books. The MIT Press, July 1, 2004. ISBN: 0-2620-4219-3, 978-0-26204-219-2.

[584] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics*, 26(1):29–41, 1996. Online available at ftp://iridia.ulb.ac.be/pub/mdorigo/journals/IJ.10-SMC96.pdf and http://citeseer.ist.psu.edu/dorigo96ant.html [accessed 2007-08-05].

[585] Marco Dorigo, Gianni Di Caro, and Luca Maria Gambardella. Ant algorithms for discrete optimization. Technical Report IRIDIA/98-10, Universite Libre de Bruxelles, Belgium, 1998. Online available at http://citeseer.ist.psu.edu/dorigo98ant.html [accessed 2007-08-05]. See [586].

[586] Marco Dorigo, Gianni Di Caro, and Luca Maria Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5(2):137–172, 1999. Online available at http://www.idsia.ch/~luca/ij_23-alife99.pdf [accessed 2007-09-15]. See [585].

[587] Marco Dorigo, Gianni Di Caro, and Thomas Stützle. Special issue on "ant algorithms". *Future Generation Computer Systems*, 16(8):851–955, June 2000. ISSN: 0167-739X.

[588] Marco Dorigo, Gianni Di Caro, and Thomas Stützle, editors. *From Ant Colonies to Artificial Ants – First International Workshop on Ant Colony Optimization and Swarm Intelligence – ANTS 98*, October 15–16, 2000, Brussels, Belgium. No proceedings, but best papers appear in [587].

[589] Marco Dorigo, Luca Maria Gambardella, Martin Middendorf, and Thomas Stützle, editors. *From Ant Colonies to Artificial Ants – Second International Workshop on Ant Colony Optimization and Swarm Intelligence – ANTS 2000*, September 8–9, 2000, Brussels, Belgium. See also [591].

[590] Marco Dorigo, Gianni Di Caro, and Michael Samples, editors. *From Ant Colonies to Artificial Ants – Third International Workshop on Ant Colony Optimization and Swarm Intelligence – ANTS 2002*, volume 2463/2002 of *Lecture Notes in Computer Science (LNCS)*, September 12–14, 2002, Brussels, Belgium. Springer Verlag, Berlin, Germany. ISBN: 978-3-54044-146-5. doi:10.1007/3-540-45724-0.

[591] Marco Dorigo, Luca Maria Gambardella, Martin Middendorf, and Thomas Stützle. Special section on "ant colony Optimization". *IEEE Transactions on Evolutionary Computation*, 6(4):317–365, August 2002. doi:10.1109/TEVC.2002.802446.

[592] Marco Dorigo, Mauro Birattari, Christian Blum, Luca Maria Gambardella, Francesco Mondada, and Thomas Stützle, editors. *Fourth International Workshop on Ant Colony Optimization and Swarm Intelligence – ANTS 2004*, volume 3172/2004 of *Lecture Notes in Computer Science (LNCS)*, September 5–8, 2004, Brussels, Belgium. Springer Verlag, Berlin, Germany. ISBN: 978-3-54022-672-7. doi:10.1007/b99492.

[593] Marco Dorigo, Mauro Birattari, and Thomas Stützle. Ant colony optimization. *Computational Intelligence Magazine*, 1(4):28–39, November 2006. ISSN: 1556-603X. INSPEC Accession Number: 9184238. doi:10.1109/MCI.2006.329691.

[594] Marco Dorigo, Luca Maria Gambardella, Mauro Birattari, A. Martinoli, Ricardo Poli, and Thomas Stützle, editors. *Fifth International Workshop on Ant Colony Optimization and Swarm Intelligence – ANTS 2006*, volume 4150/2006 of *Lecture Notes in Computer Science (LNCS)*, September 4–7, 2006, Brussels, Belgium. Springer Verlag, Berlin, Germany. ISBN: 978-3-54038-482-3. doi:10.1007/11839088.

[595] Norman Richard Draper and H. Smith. *Applied regression analysis*. Wiley, New York, 1966. Reviewed in [631].

[596] Dimiter Driankov, Peter W. Eklund, and Anca L. Ralescu, editors. *Fuzzy Logic and Fuzzy Control, IJCAI'91, Proceedings of Workshops on Fuzzy Logic and Fuzzy Control*, volume 833 of *Lecture Notes in Computer Science (LNCS)*, 1994, Sydney, Australia. Springer. ISBN: 3-5405-8279-7. See also [1495, 1496, 1608].

[597] Stefan Droste and Dirk Wiesmann. On representation and genetic operators in evolutionary algorithms. Technical Report CI–41/98, Fachbereich Informatik, Universität Dortmund, 44221 Dortmund, June 1998. Online available at http://citeseer.ist.psu.edu/323494.html and http://en.scientificcommons.org/336032 [accessed 2007-08-12].

[598] Stefan Droste, Thomas Jansen, and Ingo Wegener. On the optimization of unimodal functions with the (1+1) evolutionary algorithm. In *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature – PPSN V*, pages 13–22, 1998. doi:10.1007/BFb0056845. In proceedings [624].

[599] Stefan Droste, Thomas Jansen, and Ingo Wegener. Perhaps not a free lunch but at least a free appetizer. Reihe Computational Intelligence: Design and Management of Complex Technical Processes and Systems by means of Computational Intelligence Methods CI-45/98, University of Dortmund, Collaborative Research Center 531, September 1998. See also [600]. Online available at http://hdl.handle.net/2003/5339 [accessed 2009-03-01].

[600] Stefan Droste, Thomas Jansen, and Ingo Wegener. Perhaps not a free lunch but at least a free appetizer. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 1999. See also [599]. In proceedings [142].

[601] Stefan Droste, Thomas Jansen, and Ingo Wegener. Optimization with randomized search heuristics – the (a)nfl theorem, realistic scenarios, and difficult functions. Reihe Computational Intelligence: Design and Management of Complex Technical Processes and Systems by means of Computational Intelligence Methods CI-91/00, University of Dortmund, Collaborative Research Center 531, August 2000. See also [602]. Online available at http://hdl.handle.net/2003/5394 [accessed 2009-03-01].

[602] Stefan Droste, Thomas Jansen, and Ingo Wegener. Optimization with randomized search heuristics – the (a)nfl theorem, realistic scenarios, and difficult functions. *Theoretical Computer Science*, 287(1):131–144, September 25, 2002. ISSN: 0304-3975. doi:10.1016/S0304-3975(02)00094-4. See also [601]. Online available at http://dx.doi.org/10.1016/S0304-3975(02)00094-4 and http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.35.5850 [accessed 2009-03-01].

[603] Miloš Drutarovský, Viktor Fischer, Martin Šimka, and Frédéric Celle. A simple pll-based true random number generator for embedded digital systems. *Computing and Informatics*, 23(5):501–515, 2004.

[604] Marc Dubreuil, Christian Gagné, and Marc Parizeau. Analysis of a master-slave architecture for distributed evolutionary computations. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 36(1):229–235, February 2006. doi:10.1109/TSMCB.2005.856724. Online available at http://vision.gel.ulaval.ca/~parizeau/publications/smc06.pdf [accessed 2008-04-06].

[605] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. Estimation, Simulation, and Control – Wiley-Interscience Series in Discrete Mathematics and Optimization. Wiley Interscience, second edition, November 2000. ISBN: 0-4710-5669-3.

[606] John Duffy and Jim Engle-Warnick. Using symbolic regression to infer strategies from experimental data. In David A. Belsley and Christopher F. Baum, editors, *Fifth International Conference: Computing in Economics and Finance*, page 150, June 24–26, 1999. Boston College, MA, USA. Online available at http://www.pitt.edu/~jduffy/papers/Usr.pdf and http://citeseer.ist.psu.edu/304022.html [accessed 2007-09-09]. Later published as bookchapter: [607].

[607] John Duffy and Jim Engle-Warnick. Using symbolic regression to infer strategies from experimental data. In *Evolutionary Computation in Economics and Finance*, chapter 4, pages 61–84. Physica-Verlag Heidelberg, 2002. In collection [388]. Presented at CEF'99 (see [606]). Online available at http://www.pitt.edu/~jduffy/docs/Usr.ps [accessed 2007-09-09].

[608] Dumitru Dumitrescu, Beatrice Lazzerini, Lakhmi C. Jain, and A. Dumitrescu. *Evolutionary Computation*, volume 18 of *International Series on Computational Intelligence*. CRC Press, June 2000. ISBN: 978-0-84930-588-7, 0-8493-0588-8. Partly online available at http://books.google.de/books?hl=de&id=MSU9ep79JvUC [accessed 2008-06-11].

[609] Bruce S. Duncan and Arthur J. Olson. Applications of evolutionary programming for the prediction of protein-protein interactions. In *Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming*, pages 411–417, 1996. In proceedings [709].

[610] Margaret H. Dunham. *Data Mining: Introductory and Advanced Topics*. Prentice Hall PTR, Upper Saddle River, NJ, USA, August 2002. ISBN: 0-1308-8892-3.

[611] H. Durrant-Whyte. Data fusion in sensor networks. In *Proceedings of IEEE International Conference on Video and Signal Based Surveillance AVSS'06*, pages 39–39, November 2006, Sydney, Australia. ISBN: 0-7803-9202-7. Online available at http://www.ee.ucla.edu/~mbs/ipsn05/keynote_abstracts/hdurrantwhyte.pdf and http://portal.acm.org/citation.cfm?id=1147687 [accessed 2007-08-01].

[612] R. Kent Dybvig. *The SCHEME programming language*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1987. ISBN: 0-1379-1864-X, 978-0-26254-148-0. With Illustrations by Jean-Pierre Hébert. Third edition (October 2003) Online available at http://www.scheme.com/tspl3/ [accessed 2007-09-15].

[613] G. Dzemyda, V. Saltenis, and A. Zilinskas, editors. *Stochastic and Global Optimization*. Nonconvex Optimization and Its Applications. Springer-Verlag GmbH, March 1, 2002. ISBN: 978-1-40200-484-1.

E

[614] Fernando Almeida e Costa, Luis Mateus Rocha, Ernesto Costa, Inman Harvey, and António Coutinho, editors. *Advances in Artificial Life, Proceedings of the 9th European Conference, ECAL 2007*, volume 4648 of *Lecture Notes in Computer Science (LNCS), subseries Lecture Notes in Artificial Intelligence (LNAI)*, September 10–14, 2007, Lisbon, Portugal. Springer, Berlin / Heidelberg. ISBN: 978-3-54074-912-7. doi:10.1007/978-3-540-74913-4.

[615] Russel C. Eberhart and James Kennedy. A new optimizer using particle swarm theory. In *Proceedings of the Sixth International Symposium on Micro Machine and Human Science MHS'95*, pages 39–43. IEEE Press, October 1995. ISBN: 0-7803-2676-8.

[616] Russel C. Eberhart and Yuhui Shi. Comparison between genetic algorithms and particle swarm optimization. In *Proceedings of the 7th International Conference on Evolutionary Programming*, pages 611–616, 1998. In proceedings [1670].

[617] Marc Ebner, Michael O'Neill, Anikó Ekárt, Leonardo Vanneschi, and Anna Isabel Esparcia-Alcázar, editors. *Proceedings of the 10th European Conference on Genetic Programming, EuroGP 2007*, volume 4445/2007 of *Lecture Notes in Computer Science (LNCS), LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues*, April 11-13, 2007, Valencia, Spain. Springer Berlin/Heidelberg. ISBN: 3-5407-1602-5, 978-3-54071-602-0. Library of Congress Control Number: 2007923720. doi:10.1007/978-3-540-71605-1.

[618] Bruce Eckel. *Thinking in Java*. Prentice Hall PTR, 4th edition, February 2006. ISBN: 978-0-13187-248-6. 3rd edition online available at http://www.mindview.net/Books/TIJ/ [accessed 2007-07-03].

[619] Eugene S. Edgington. *Randomization tests*. CRC Press / Marcel Dekker Ltd, third revised and expanded edition, July 1995. ISBN: 0-8247-9669-1, 978-0-82479-669-3. Partly online available at http://books.google.de/books?id=UxGqdcmL5gMC [accessed 2008-08-16].

[620] Matthias Ehrgott, editor. *Proceedings of the 19th International Conference on Multiple Criteria Decision Making: MCDM for Sustainable Energy and Transportation Systems (MCDM'2008)*, June 7–12, 2008, Auckland, New Zealand. See https://secure.orsnz.org.nz/mcdm2008/ [accessed 2007-09-10].

[621] Ágoston E. Eiben and A. Michalewicz, editors. *Evolutionary Computation*. Ohmsha, Tokyo, Japan, 1999. ISBN: 4-2749-0269-2. This is the book edition of the journal, Fundamenta Informaticae, Volume 35, Nos. 1-4, 1998.

[622] Ágoston E. Eiben and C. A. Schippers. On evolutionary exploration and exploitation. *Fundamenta Informaticae*, 35(1-4):35–50, 1998. Online available at http://www.cs.vu.nl/~gusz/papers/FunInf98-Eiben-Schippers.ps and http://citeseer.ist.psu.edu/eiben98evolutionary.html [accessed 2007-08-22].

[623] Ágoston E. Eiben and James E. Smith. *Introduction to Evolutionary Computing*. Natural Computing Series. Springer, first edition, November 2003. ISBN: 978-3-54040-184-1. Partly online available at http://books.google.de/books?id=7IOE5VIpFpwC [accessed 2008-05-18].

[624] Ágoston E. Eiben, Thomas Bäck, Marc Schoenauer, and Hans-Paul Schwefel, editors. *Proceedings of the 5th International Conference on Parallel Problem Solving from Nature – PPSN V*, volume 1498/1998 of *Lecture Notes in Computer Science (LNCS)*, September 27–30, 1998, Amsterdam, The Netherlands. Springer. ISBN: 3-5406-5078-4, 978-3-54065-078-2. doi:10.1007/BFb0056843. Partly online available at http://books.google.de/books?id=HLeU_1TkwTsC [accessed 2008-08-13].

[625] Manfred Eigen and Peter Schuster. *The Hypercycle: A Principle of Natural Self Organization*. Springer-Verlag, June 1979. ISBN: 0-3870-9293-5, 978-0-38709-293-5.

[626] Anikó Ekárt and Sándor Zoltán Németh. Selection based on the pareto nondomination criterion for controlling code growth in genetic programming. *Genetic Programming and Evolvable Machines*, 2(1):61–73, March 2001. ISSN: 1389-2576 (Print) 1573-7632 (Online). doi:10.1023/A:1010070616149. Online available at http://www.springerlink.com/content/j70v0303523v6x65/fulltext.pdf [accessed 2008-05-04].

[627] Sven E. Eklund. A massively parallel GP engine in VLSI. In *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, pages 629–633, 2002. doi:10.1109/CEC.2002.1006999. In proceedings [703].

[628] Khaled El-Fakihy, Hirozumi Yamaguchi, and Gregor von Bochmann. A method and a genetic algorithm for deriving protocols for distributed applications with minimum communication cost. In *Proceedings of Eleventh IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS'99)*, pages 863–868. ACTA Press, November 3–6, 1999, Cambridge, Boston, MA, USA. Online available at http://citeseer.ist.psu.edu/430349.html and http://www-higashi.ist.osaka-u.ac.jp/~h-yamagu/resource/pdcs99.pdf [accessed 2007-09-14].

[629] Niles Eldredge and Stephen Jay Gould. Punctuated equilibria: The tempo and mode of evolution reconsidered. *Paleobiology*, 3(2):115–151, April 1, 1977. Online available at http://www.nileseldredge.com/pdf_files/Punctuated_Equilibria_Gould_Eldredge_1977.pdf [accessed 2008-11-09].

[630] Nils Eldredge and Stephen Jay Gould. Punctuated equilibria: an alternative to phyletic gradualism. In Thomas J. M. Schopf, editor, *Models in Paleobiology*, chapter 5, pages 82–115. Freeman, Cooper & Co / DoubleDay, San Francisco, USA, January 1972. ISBN: 0-3850-3024-X, 978-0-38503-024-3. Online available at http://www.blackwellpublishing.com/ridley/classictexts/eldredge.pdf [accessed 2008-07-01].

[631] D. M. Ellis. Book reviews: "applied regression analysis" by N. P. Draper and H. Smith. *Applied Statistics*, 17(1):83–84, 1968. ISSN: 0035-9254. Reviews [595].

[632] Michael Emmerich, Monika Grötzner, Berd Groß, and Martin Schütz. Mixed-integer evolution strategy for chemical plant optimization. In I.C. Parmee, editor, *Evolutionary Design and Manufacture – Selected papers from ACDM'00*, pages 55–67, April 2000, Plymouth, UK. Springer, London, New York. Online available at http://citeseer.ist.psu.edu/359756.html and http://www.liacs.nl/~emmerich/pdf/EGG+00.pdf [accessed 2007-08-27].

[633] Andries P. Engelbrecht. *Fundamentals of Computational Swarm Intelligence*. Wiley and Sons, Ltd., England, January 3, 2006. ISBN: 0-4700-9191-6, 978-0-47009-191-3.

[634] Karl Entacher. Bad subsequences of well-known linear congruential pseudorandom number generators. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):61–70, 1998. ISSN: 1049-3301. doi:10.1145/272991.273009. Online available at http://doi.acm.org/10.1145/272991.273009 [accessed 2007-09-15].

[635] Thomas Erl. *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. The Prentice Hall Service-Oriented Computing Series from Thomas Erl. Prentice Hall PTR, August 2, 2005. ISBN: 978-0-13185-858-9.

[636] Larry J. Eshelman, editor. *Proceedings of the Sixth International Conference on Genetic Algorithms*, July 15–19, 1995, Pittsburgh, PA, USA. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN: 1-5586-0370-0.

[637] Larry J. Eshelman and J. David Schaffer. Preventing premature convergence in genetic algorithms by preventing incest. In *ICGA, Proceedings of the 4th International Conference on Genetic Algorithms*, pages 115–122, 1991. In proceedings [170].

[638] Larry J. Eshelman, Richard A. Caruana, and J. David Schaffer. Biases in the crossover landscape. In *Proceedings of the third international conference on Genetic algorithms*, pages 10–19, 1989. In proceedings [1820].

[639] Ken Chen et al., editor. *Proceedings of the Seventh Joint Conference on Information Science (JCIS 2003), Section: The Fifth International Workshop on Frontiers in Evolutionary Algorithms (FEA 2003)*, September 26–30, 2003, Embassy Suites Hotel and Conference Center, Cary, North Carolina, USA. Workshop held in conjunction with Seventh Joint Conference on Information Sciences.

F

[640] Marco Aurelio Falcone, Heitor Silverio Lopes, and Leandro Dos Santos Coelho. Supply chain optimisation using evolutionary algorithms. *International Journal of Computer Applications in Technology*, 31(3/4):158–167, 2008. doi:10.1504/IJCAT.2008.018154.

[641] David C. Fallside and Priscilla Walmsley. *XML Schema Part 0: Primer Second Edition*. World Wide Web Consortium (W3C), second edition, October 28, 2004. W3C Recommendation. Online available at http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/ [accessed 2007-09-02].

[642] Shu-Kai S. Fan and Erwie Zaharaa. A hybrid simplex search and particle swarm optimization for unconstrained optimization. *European Journal of Operational Research*,

181(2):527–548, September 1 2007. doi:10.1016/j.ejor.2006.06.034. Continuous Optimization. Online available at http://dx.doi.org/10.1016/j.ejor.2006.06.034 [accessed 2008-06-14].

[643] Shu-Kai S. Fan, Yun-Chia Liang, and Erwie Zahara. A genetic algorithm and a particle swarm optimizer hybridized with nelder-mead simplex search. *Computers and Industrial Engineering*, 50(4):401–425, August 2006. ISSN: 0360-8352. doi:10.1016/j.cie.2005.01.022. Special issue: Sustainability and globalization: Selected papers from the 32nd ICC&IE.

[644] Günter Fandel and Thomás Gál, editors. *Proceedings of the 3rd International Conference on Multiple Criteria Decision Making: Theory and Application (MCDM'1979)*, volume 17 of *Lecture Notes in Economics and Mathematical Systems*, August 20–24, 1979, Hagen/Königswinter, (West) Germany. Springer-Verlag, Berlin, Germany. ISBN: 978-0-38709-963-7. Published in May 1980.

[645] Günter Fandel, Thomás Gál, and Thomas Hanne, editors. *Proceedings of the 12th International Conference on Multiple Criteria Decision Making (MCDM'1995)*, volume 448 of *Lecture Notes in Economics and Mathematical Systems*, 1995, Hagen, Germany. Springer. ISBN: 978-3-54062-097-6. Published on March 21, 1997.

[646] Michael E. Farmer, Shweta Bapna, and Anil K. Jain. Large scale feature selection using modified random mutation hill climbing. In *ICPR 2004. Proceedings of the 17th International Conference on Pattern Recognition*, volume 2, pages 287–290, August 23–26, 2004. IEEE Computer Society, Los Alamitos, CA, USA. doi:10.1109/ICPR.2004.1334169.

[647] Xiang Fei, Junzhou Luo, Jieyi Wu, and Guanqun Gu. Qos routing based on genetic algorithms. *Computer Commnunications*, 22(15-16):1392–1399, September 25, 1999. doi:10.1016/S0140-3664(99)00113-9. Online available at http://neo.lcc.uma.es/opticomm/main.html and http://dx.doi.org/10.1016/S0140-3664(99)00113-9 [accessed 2008-08-01].

[648] Robert Feldt and Peter Nordin. Using factorial experiments to evaluate the effect of genetic programming parameters. In *Genetic Programming, Proceedings of EuroGP'2000*, pages 271–282, 2000. doi:10.1007/b75085. In proceedings [1666]. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.36.4989 [accessed 2008-10-14].

[649] William Feller. *An Introduction to Probability Theory and Its Applications*, volume 1. Wiley, 3 edition, 1968. ISBN: 0-4712-5708-7, 978-0-47125-708-0.

[650] Thomas A. Feo and Jonathan F. Bard. Flight scheduling and maintenance base planning. *Management Science*, 35(12):1415–1432, December 1989. ISSN: 0025-1909. doi:10.1287/mnsc.35.12.1415.

[651] Thomas A. Feo and Mauricio G. C. Resende. A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67–71, April 1989. ISSN: 0167-6377. doi:10.1016/0167-6377(89)90002-3. Online available at http://dx.doi.org/10.1016/0167-6377(89)90002-3 [accessed 2008-10-20].

[652] Thomas A. Feo and Mauricio G. C. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6(2):109–133, March 1995. ISSN: 0925-5001 (print version), 1573-2916 (electronic version). doi:10.1007/BF01096763. Journal no. 10898. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.8667 and http://www.research.att.com/~mgcr/doc/gtut.ps.Z [accessed 2008-10-20], and http://www.springerlink.com/content/m57h74q0805g1143/fulltext.pdf [accessed 2008-10-20].

[653] Vitaliy Feoktistov. *Differential Evolution – In Search of Solutions*, volume 5 of *Springer Optimization and Its Applications*. Springer, December 2006. ISBN: 978-0-38736-895-5.

[654] Cândida Ferreira. Gene expression programming: a new adaptive algorithm for solving problems. *Complex Systems*, 13:87, 2001. Online avail-

able at http://arxiv.org/ftp/cs/papers/0102/0102027.pdf and http://www.gene-expression-programming.com/GEPBiblio.asp [accessed 2007-08-23].

[655] Cândida Ferreira. Mutation, transposition, and recombination: An analysis of the evolutionary dynamics. In H. John Caulfield, Shu-Heng Chen, Heng-Da Cheng, Richard J. Duro, Vasant Honavar, Etienne E. Kerre, Mi Lu, Manuel Grana Romay, Timothy K. Shih, Dan Ventura, Paul P. Wang, and Yuanyuan Yang, editors, *Proceedings of the 6th Joint Conference on Information Science (JCIS)*, pages 614–617. JCIS / Association for Intelligent Machinery, Inc., March 2002, Research Triangle Park, North Carolina, USA. ISBN: 0-9707-8901-7. Online available at http://www.gene-expression-programming.com/GEPBiblio.asp [accessed 2007-08-23].

[656] Cândida Ferreira. Analyzing the founder effect in simulated evolutionary processes using gene expression programming. In *Soft Computing Systems - Design, Management and Applications, HIS 2002*, pages 153–162, 2002. In proceedings [7]. Online available at http://www.gene-expression-programming.com/GEPBiblio.asp [accessed 2007-08-23].

[657] Cândida Ferreira. Genetic representation and genetic neutrality in gene expression programming. *Advances in Complex Systems*, 5(4):389–408, 2002. Online available at http://www.gene-expression-programming.com/GEPBiblio.asp [accessed 2007-08-23].

[658] Cândida Ferreira. Function finding and the creation of numerical constants in gene expression programming. In *7th Online World Conference on Soft Computing in Industrial Applications*, September 2002. Online available at http://www.gene-expression-programming.com/GEPBiblio.asp [accessed 2007-08-23].

[659] Cândida Ferreira. Discovery of the boolean functions to the best density-classification rules using gene expression programming. In James A. Foster, Evelyne Lutton, Julian Francis Miller, Conor Ryan, and Andrea G. B. Tettamanzi, editors, *Proceedings of the 5th European Conference on Genetic Programming, EuroGP 2002*, pages 50–59, 2002. ISBN: 3-5404-3378-3. In proceedings [737]. Online available at http://www.gene-expression-programming.com/webpapers/ferreira-EuroGP02.pdf [accessed 2007-09-09].

[660] Cândida Ferreira. Function finding and the creation of numerical constants in gene expression programming. In *7th Online World Conference on Soft Computing in Industrial Applications*, September 2002. Online available at http://www.sureserv.com/technic/datum_detail.php?id=466 and http://www.gene-expression-programming.com/GEPBiblio.asp [accessed 2007-08-23].

[661] Cândida Ferreira. Designing neural networks using gene expression programming. In Ajith Abraham and Mario Köppen, editors, *9th Online World Conference on Soft Computing in Industrial Applications*, September 2004. Online available at http://www.gene-expression-programming.com/GEPBiblio.asp [accessed 2007-08-23].

[662] Alan M. Ferrenberg, D. P. Landau, and Y. Joanna Wong. Monte carlo simulations: Hidden errors from "good" random number generators. *Physical Review Letters*, 69 (23):3382–3384, December 1992. Online available at http://prola.aps.org/pdf/PRL/v69/i23/p3382_1 [accessed 2007-08-18].

[663] Paola Festa and Mauricio G.C. Resende. An annotated bibliography of grasp. AT&T Labs Research Technical Report TD-5WYSEW, AT&T Labs, February 29, 2004. Online available at http://www.research.att.com/~mgcr/grasp/gannbib/gannbib.html [accessed 2008-10-20].

[664] Mark Fewster and Dorothy Graham. *Software Test Automation*. Addison-Wesley Professional, August 25, 1999. ISBN: 0-2013-3140-3.

[665] Anthony V. Fiacco and Garth P. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques*. John Wiley & Sons Inc., 1968. ISBN: 0-4712-5810-5, 978-0-47125-810-0, 0-8987-1254-8, 978-0-89871-254-4. Reprint: Society for Industrial Mathematics, 1987; SIAM, Philadelphia, PA, USA, 1990 (Classics

in Applied Mathematics Series, Vol. 4). Partly online available at http://books.google.com/books?id=sjD_RJfxvr0C&hl=de [accessed 2008-11-15].

[666] Jonathan E. Fieldsend, Richard M. Everson, and Sameer Singh. Extensions to the strength pareto evolutionary algorithm. *IEEE Transactions on Evolutionary Computation*, 2001. submitted, superseded by [667]. Online available at http://www.dcs.ex.ac.uk/people/reverson/pubs/espea.ps.gz and http://citeseer.ist.psu.edu/fieldsend01extensions.html [accessed 2007-08-25].

[667] Jonathan E. Fieldsend, Richard M. Everson, and Sameer Singh. Using unconstrained elite archives for multi-objective optimisation. *IEEE Transactions on Evolutionary Computing*, 7(3):305–323, 2003. This document supersedes the manuscript [666]. Online available at http://citeseer.ist.psu.edu/543052.html and http://www.lania.mx/~ccoello/EMOO/fieldsend03.pdf.gz [accessed 2007-08-25].

[668] Richard Fikes and Wendy Lehnert, editors. *Proceedings of the 11th National Conference on Artificial Intelligence, AAAI*, July 11–15, 1993, Washington, DC, USA. The AAAI Press/The MIT Press. ISBN: 0-2625-1071-5. See http://www.aaai.org/Conferences/AAAI/aaai93.php [accessed 2007-09-06].

[669] Bogdan Filipič and Jurij Šilc, editors. *Proceedings of the Second International Conference on Bioinspired Optimization Methods and their Applications (BIOMA 2006)*, October 9–10, 2006, Jožef Stefan International Postgraduate School, Ljubljana, Slovenia. Jožef Stefan Institute. ISBN: 9-6163-0381-3, 978-9-61630-381-1. Online available at http://is.ijs.si/is/is2006/Proceedings/C/BIOMA06_Complete.pdf [accessed 2008-11-05]. Part of the Information Society Multiconference (IS 2006).

[670] Bogdan Filipič and Jurij Šilc, editors. *Proceedings of the Third International Conference on Bioinspired Optimization Methods and their Applications (BIOMA 2008)*, October 13–14, 2008, Kolarjeva predavalnica, Jožef Stefan International Postgraduate School, Ljubljana, Slovenia. Jožef Stefan Institute. ISBN: 978-9-61264-002-6. Online available at http://is.ijs.si/is/is2008/zborniki/D%20-%20BIOMA.pdf [accessed 2008-11-05]. Part of the Information Society Multiconference (IS 2008).

[671] Bogdan Filipič and Jurij Šilc, editors. *Proceedings of the International Conference on Bioinspired Optimization Methods and their Applications (BIOMA 2004)*, October 11–12, 2004, Jožef Stefan International Postgraduate School, Ljubljana, Slovenia. Jožef Stefan Institute. Part of the Information Society Multiconference (IS 2004).

[672] Steven R. Finch. *Mathematical Constants (Encyclopedia of Mathematics and its Applications)*. Cambridge University Press, August 18, 2003. ISBN: 978-0-52181-805-6. See http://algo.inria.fr/bsolve/ [accessed 2007-09-15].

[673] Steven R. Finch. Transitive relations, topologies and partial orders, June 5, 2003. Online available at http://citeseer.ist.psu.edu/finch03transitive.html and http://algo.inria.fr/csolve/posets.pdf [accessed 2007-07-28]. See also [672].

[674] Andreas Fink and S. Voß. Generic application of tabu search methods to manufacturing problems. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics (SMC'98)*, volume 3, pages 2385–2390. IEEE, October 1998, Piscataway. doi:10.1109/ICSMC.1998.725013.

[675] Hans Fischer. *The different forms and applications of the central limit theorem in its development from classical to modern probability theory*. Shaker Verlag GmbH, Aachen, Germany, August 2000. Partly online available at http://www.ku-eichstaett.de/Fakultaeten/MGF/Mathematik/Didmath/Didmath.Fischer/HF_sections/content/ [accessed 2008-08-19]. German version: Die verschiedenen Formen und Funktionen des zentralen Grenzwertsatzes in der Entwicklung von der klassischen zur modernen Wahrscheinlichkeitsrechnung, ISBN: 978-3-8265-7767-3.

[676] Viktor Fischer, Miloš Drutarovský, Martin Šimka, and Nathalie Bochard. High performance true random number generator in altera stratix fplds. In Jürgen Becker, Marco Platzner, and Serge Vernalde, editors, *Field-Programmable Logic and Appli-*

cations – FPL 2004, volume 3203 of *Lecture Notes in Computer Science (LNCS)*, pages 555–564, August 2004. Springer-Verlag, Lueven, Belgium. Online available at http://www.best.tuke.sk/simka/pub.html [accessed 2007-09-15].

[677] Sir Ronald Aylmer Fisher. The correlations between relatives on the supposition of mendelian inheritance. *Philosophical Transactions of the Royal Society of Edinburgh*, 52:399–433, 1918. See also http://en.wikipedia.org/wiki/The_Correlation_Between_Relatives_on_the_Supposition_of_Mendelian_Inheritance [accessed 2008-03-02].

[678] Sir Ronald Aylmer Fisher. On the interpretation of χ^2 from contingency tables, and the calculation of p. *Journal of the Royal Statistical Society*, 85:87–94, 1922. Online available at http://hdl.handle.net/2440/15173 [accessed 2008-12-08]. See also [178].

[679] Sir Ronald Aylmer Fisher. Applications of "student's" distribution. *Metron*, 5:90–104, 1925. Online available at http://digital.library.adelaide.edu.au/coll/special/fisher/43.pdf [accessed 2007-09-30]. See also [178].

[680] Sir Ronald Aylmer Fisher. *Statistical Methods for Research Workers*. Oliver and Boyd, Edinburgh, UK, 1925, 1970. ISBN: 0-0500-2170-2, 978-0-05002-170-5. Online available at http://psychclassics.yorku.ca/Fisher/Methods/ [accessed 2008-12-08].

[681] Sir Ronald Aylmer Fisher. The arrangement of field experiments. *Journal of the Ministry of Agriculture of Great Britain*, 33:503–513, 1926. Online available at http://digital.library.adelaide.edu.au/coll/special//fisher/48.pdf [accessed 2008-08-07]. See also [178].

[682] Sir Ronald Aylmer Fisher. *The design of experiments*. Collier Macmillan / Oliver & Boyd, Edinburgh, UK, 1935. ISBN: 0-0284-4690-9, 978-0-02844-690-5. 9th edition, reprint 1971/1972/1974.

[683] Sir Ronald Aylmer Fisher. "the coefficient of racial likeness" and the future of craniometry. *The Journal of the Royal Anthropological Institute of Great Britain and Ireland*, 66:57–63, January–July 1936. Online available at http://digital.library.adelaide.edu.au/dspace/bitstream/2440/15228/1/141.pdf [accessed 2008-08-16]. See also [178].

[684] Sir Ronald Aylmer Fisher. *Statistical methods and scientific inference*. New York, Hafner Press/Macmillan Pub Co, revised (june 1973) edition, 1956. ISBN: 0-0284-4740-9.

[685] J. Michael Fitzpatrick and John J. Grefenstette. Genetic algorithms in noisy environments. *Machine Learning*, 3(2–3):101–120, October 1988. ISSN: 0885-6125 (Print) 1573-0565 (Online). doi:10.1007/BF00113893. Online available at http://www.springerlink.com/content/n6w328441t63q374/fulltext.pdf [accessed 2008-07-19].

[686] David Flanagan. *Java Examples in a Nutshell*. O'Reilly Media, Inc., 3rd, illustrated edition, January 2004. ISBN: 978-0-59600-620-4.

[687] David Flanagan. *Java In A Nutshell*. O'Reilly Media, Inc., 5th edition, March 2005. ISBN: 978-0-59600-773-7.

[688] Eric Fleury, Jean-Loup Guillaume, Céline Robardet, and Antoine Scherrer. Analysis of dynamic sensor networks: Power law then what? In *Second International Conference on COMmunication Systems softWAre and middlewaRE (COMSWARE 2007)*. IEEE, January 7–12, 2007, Bangalore, India. ISBN: 1-4244-0614-5. doi:10.1109/COMSWA.2007.382427. Online available at http://jlguillaume.free.fr/www/publis/Guillaume07analysis.pdf [accessed 2007-08-01].

[689] Dario Floreano, Jean-Daniel Nicoud, and Francesco Mondada, editors. *Advances in Artificial Life, Proceedings of the 5th European Conference, ECAL'99*, volume 1674 of *Lecture Notes in Computer Science (LNCS)*, September 13–17, 1999, Lausanne, Switzerland. Springer. ISBN: 3-5406-6452-1.

[690] Christodoulos A. Floudas, editor. *Deterministic Global Optimization: Theory, Methods and Applications*, volume 37 of *Nonconvex Optimization and Its Applications*.

Springer-Verlag GmbH, 1999. ISBN: 978-0-79236-014-8. Partly online available at http://books.google.de/books?id=qZSpq27TsOcC [accessed 2008-03-10].

[691] Christodoulos A. Floudas and Panos M. Pardalos, editors. *Frontiers in Global Optimization*. Nonconvex Optimization and Its Applications. Springer US, November 30, 2003. ISBN: 978-1-40207-699-2.

[692] Henrick Flyvbjerg and Benny Lautrup. Evolution in a rugged fitness landscape. *Physical Review A*, 46(10):6714–6723, November 15, 1992. doi:10.1103/PhysRevA.46.6714. Online available at http://citeseer.ist.psu.edu/26647.html and http://prola.aps.org/abstract/PRA/v46/i10/p6714_1 [accessed 2007-08-14].

[693] Terence C. Fogarty, editor. *Proceedings of the Workshop on Artificial Intelligence and Simulation of Behaviour (AISB), International Workshop on Evolutionary Computing, Selected Papers*, volume 865/1994 of *Lecture Notes in Computer Science (LNCS)*, April 11–13 1994, Leeds, UK. The Society for the Study of Artificial Intelligence and the Simulation of Behaviour (SSAISB), Springer Heidelberg/Berlin. ISBN: 3-5405-8483-8, 978-3-54058-483-4. doi:10.1007/3-540-58483-8.

[694] Terence C. Fogarty, editor. *Proceedings of the Workshop on Artificial Intelligence and Simulation of Behaviour (AISB), International Workshop on Evolutionary Computing, Selected Papers*, volume 993/1995 of *Lecture Notes in Computer Science (LNCS)*, April3–4 1995, Sheffield, UK. The Society for the Study of Artificial Intelligence and the Simulation of Behaviour (SSAISB), Springer Heidelberg/Berlin. ISBN: 3-5406-0469-3, 978-3-54060-469-3. doi:10.1007/3-540-60469-3.

[695] Terence C. Fogarty, editor. *Proceedings of the Workshop on Artificial Intelligence and Simulation of Behaviour (AISB), International Workshop on Evolutionary Computing, Selected Papers*, volume 1143/1996 of *Lecture Notes in Computer Science (LNCS)*, April 1–2 1996, Brighton, UK. The Society for the Study of Artificial Intelligence and the Simulation of Behaviour (SSAISB), Springer Heidelberg/Berlin. ISBN: 3-5406-1749-3, 978-3-54061-749-5. doi:10.1007/BFb0032768.

[696] David B. Fogel, editor. *Evolutionary Computation: The Fossil Record*. Wiley-IEEE Press, May 1, 1998. ISBN: 978-0-78033-481-6.

[697] David B. Fogel. *System Identification through Simulated Evolution: A Machine Learning Approach to Modeling*. Ginn Press, Needham Heights, MA, June 1991. ISBN: 978-0-53657-943-0.

[698] David B. Fogel. *Evolutionary Computation: Principles and Practice for Signal Processing*. Tutorial Texts in Optical Engineering. Society of Photo-Optical Instrumentation Engineering (SPIE) Press, July 6, 2000. ISBN: 0-8194-3725-5, 978-0-81943-725-9. Partly online available at http://books.google.de/books?id=gEmZ80JVHXcC [accessed 2008-04-03].

[699] David B. Fogel. Evolving a checkers player without relying on human experience. *Intelligence*, 11(2):20–27, 2000. ISSN: 1523-8822. Online available at http://doi.acm.org/10.1145/337897.337996 [accessed 2007-08-27].

[700] David B. Fogel. *Blondie24: playing at the edge of AI*. The Morgan Kaufmann Series in Artificial Intelligence. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, September 2001. ISBN: 978-1-55860-783-5.

[701] David B. Fogel and W. Atmar, editors. *Proceedings of the 1st Annual Conference on Evolutionary Programming*, 1992. Evolutionary Programming Society, 9363 Towne Centre Dr., San Diego, CA 92121, USA.

[702] David B. Fogel and W. Atmar, editors. *Proceedings of the 2nd Annual Conference on Evolutionary Programming*, February 25–26, 1993, La Jolla, CA, USA. Evolutionary Programming Society, 9363 Towne Centre Dr., San Diego, CA 92121, USA.

[703] David B. Fogel, Mohamed A. El-Sharkawi, Xin Yao, Garry Greenwood, Hitoshi Iba, Paul Marrow, and Mark Shackleton, editors. *Proceedings of the IEEE Congress on Evolutionary Computation, CEC2002*, May 12–17, 2002, Honolulu, HI, USA. IEEE

Press, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA. ISBN: 0-7803-7278-6. CEC 2002 – A joint meeting of the IEEE, the Evolutionary Programming Society, and the IEE. Held in connection with the World Congress on Computational Intelligence (WCCI 2002).

[704] Gary B. Fogel and David W. Corne, editors. *Evolutionary Computation in Bioinformatics*. Academic Press. Elsevier LTD, Oxford, September 1, 2002. ISBN: 1-5586-0797-8, 978-1-55860-797-2. Partly online available at http://books.google.de/books?id=Jtj7NpO9F_gC [accessed 2008-04-03].

[705] Lawrence Jerome Fogel. *On the organization of intellect*. PhD thesis, UCLA University of California, Los Angeles, California, USA, 1964.

[706] Lawrence Jerome Fogel. *Intelligence Through Simulated Evolution: Forty Years of Evolutionary Programming*. Wiley Series on Intelligent Systems. John Wiley & Sons, Inc., New York, NY, USA, August 2, 1999. ISBN: 0-4713-3250-X, 978-0-47133-250-3.

[707] Lawrence Jerome Fogel and David B. Fogel. A preliminary investigation on extending evolutionary programming to include self-adaptation on finite state. *Informatica (Slovenia)*, 18(4), 1994.

[708] Lawrence Jerome Fogel, Alvin J. Owens, and Michael J. Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley & Sons, New York, USA, October 1966. ISBN: 978-0-47126-516-0.

[709] Lawrence Jerome Fogel, Peter John Angeline, and Thomas Bäck, editors. *Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming (EP'96)*, February 1996, San Diego, California, USA. MIT Press, Cambridge, Massachusetts. ISBN: 0-2620-6190-2.

[710] Gianluigi Folino, Clara Pizzuti, and Giandomenico Spezzano. Gp ensemble for distributed intrusion detection systems. In *Pattern Recognition and Data Mining, Proceedings of the Third International Conference on Advances in Pattern Recognition, ICAPR 2005*, volume 3686/2005 of *Lecture Notes in Computer Science (LNCS)*, pages 54–62. Springer Berlin / Heidelberg, August 22–25, 2005, Bath, UK. ISBN: 978-3-54028-757-5. doi:10.1007/11551188_6. Online available at http://dns2.icar.cnr.it/pizzuti/icapr05.pdf [accessed 2008-06-23].

[711] Cyril Fonlupt, Jin-Kao Hao, Evelyne Lutton, Edmund M. A. Ronald, and Marc Schoenauer, editors. *Selected Papers of the 4th European Conference on Artificial Evolution, AE'99*, volume 1829 of *Lecture Notes in Computer Science (LNCS)*, November 3–5, 1999, Dunkerque, France. Springer Berlin/Heidelberg. ISBN: 3-5406-7846-8. Published in 2000.

[712] Carlos M. Fonseca. Decision making in evolutionary optimization (abstract of invited talk). In *4th International Conference on Evolutionary Multi-Criterion Optimization*, 2007. In proceedings [1555].

[713] Carlos M. Fonseca. Preference articulation in evolutionary multiobjective optimisation – plenary talk. In *Proceedings of the Seventh International Conference on Hybrid Intelligent Systems, HIS 2007*, 2007. In proceedings [1170].

[714] Carlos M. Fonseca and Peter J. Fleming. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 416–423, 1993. In proceedings [730]. Online available at http://citeseer.ist.psu.edu/fonseca93genetic.html and http://www.lania.mx/~ccoello/EMOO/fonseca93.ps.gz [accessed 2007-08-29].

[715] Carlos M. Fonseca and Peter J. Fleming. An overview of evolutionary algorithms in multiobjective optimization. *Evolutionary Computation*, 3(1):1–16, 1995. Online available at http://citeseer.ist.psu.edu/108172.html [accessed 2007-07-29].

[716] Carlos M. Fonseca and Peter J. Fleming. Multiobjective optimization and multiple constraint handling with evolutionary algorithms – part i: A unified formulation. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and*

Humans, 28(1):26–37, 1998. Online available at http://citeseer.ist.psu.edu/fonseca98multiobjective.html [accessed 2007-07-29]. See also [717].

[717] Carlos M. Fonseca and Peter J. Fleming. Multiobjective optimization and multiple constraint handling with evolutionary algorithms – part ii: Application example. *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, 28(1):38–47, 1998. Online available at http://citeseer.ist.psu.edu/27937.html [accessed 2007-09-19]. See also [716].

[718] Carlos M. Fonseca and Peter J. Fleming. Multiobjective optimization and multiple constraint handling with evolutionary algorithms. In *Practical Approaches to Multi-Objective Optimization*, 2004. In proceedings [281]. Online available at http://drops.dagstuhl.de/opus/volltexte/2005/237/ [accessed 2007-09-19].

[719] Carlos M. Fonseca, Peter J. Fleming Fleming, Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele, editors. *Proceedings of the Second International Conference on Evolutionary Multi-Criterion Optimization (EMO'03)*, volume 2632/2003 of *Lecture Notes in Computer Science (LNCS)*, April 8–11, 2003, Faro, Portugal. Springer-Verlag, Berlin. ISBN: 978-3-54001-869-8.

[720] Walter Fontana. Algorithmic chemistry. In *Artificial Life II*, pages 159–210, 1990. In proceedings [1248].

[721] Walter Fontana, Peter F. Stadler, Erich G. Bornberg-Bauer, Thomas Griesmacher, Ivo L. Hofacker, Manfred Tacker, Pedro Tarazona, Edward D. Weinberger, and Peter Schuster. Rna folding and combinatory landscapes. *Physical Review E*, 47(3):2083–2099, March 1993. doi:10.1103/PhysRevE.47.2083. Online available at http://dx.doi.org/10.1103/PhysRevE.47.2083 and http://citeseer.ist.psu.edu/94823.html [accessed 2007-11-27].

[722] International Organization for Standardization (ISO). *ISO/IEC 14977:1996: Information technology – Syntactic metalanguage – Extended BNF*. International Organization for Standardization (ISO), 1, ch. de la Voie-Creuse, Case postale 56, CH-1211 Geneva 20, Switzerland, 1996. Online available at http://www.cl.cam.ac.uk/~mgk25/iso-14977.pdf [accessed 2007-09-15].

[723] Kenneth S. H. Forbus and Howard Shrobe, editors. *Proceedings of the 6th National Conference on Artificial Intelligence, AAAI*, July 13–17, 1987, Seattle, WA, USA. Morgan Kaufmann, San Francisco, CA, USA. See http://www.aaai.org/Conferences/AAAI/aaai87.php [accessed 2007-09-06].

[724] Agostino Forestiero, Carlo Mastroianni, and Giandomenico Spezzano. Construction of a peer-to-peer information system in grids. In Hans Czap, Rainer Unland, Cherif Branki, and Huaglory Tianfield, editors, *Self-Organization and Autonomic Informatics (I), Proceedings of the International Conference on Self-Organization and Adaptation of Multi-agent and Grid Systems (SOAS 2005)*, volume 135 of *Frontiers in Artificial Intelligence and Applications*, pages 220–236, December 11–13, 2005, Glasgow, UK. IOS Press, Nieuwe Hemweg 6B, 1013 BG Amsterdam, The Netherlands. ISBN: 1-5860-3577-0, 978-1-58603-577-8. Online available at http://grid.deis.unical.it/papers/pdf/SOAS05-ForestieroMastroianniSpezzano.pdf [accessed 2008-09-05].

[725] Agostino Forestiero, Carlo Mastroianni, and Giandomenico Spezzano. Antares: an ant-inspired p2p information system for a self-structured grid. In *Proceedings of BIONETICS 2007*, 2007. doi:10.1109/BIMNICS.2007.4610103. In proceedings [1019]. Online available at http://grid.deis.unical.it/papers/pdf/ForestieroMastroianniSpezzano-Antares.pdf [accessed 2008-06-13].

[726] Agostino Forestiero, Carlo Mastroianni, and Giandomenico Spezzano. Building a peer-to-peer information system in grids via self-organizing agents. *Journal of Grid Computing*, 6(2):125–140, June 2008. ISSN: 1570-7873 (Print) 1572-9814 (Online). doi:10.1007/s10723-007-9062-z. Online available at http://grid.deis.unical.it/papers/pdf/JGC2008-ForestieroMastroianniSpezzano.pdf [accessed 2008-09-05].

[727] Agostino Forestiero, Carlo Mastroianni, and Giandomenico Spezzano. So-grid: A self-organizing grid featuring bio-inspired algorithms. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 3(2):5:1–5:37, May 2008. ISSN: 1556-4665. doi:10.1145/1352789.1352790. Article 5. Online available at http://grid.deis.unical.it/papers/pdf/Article.pdf and http://doi.acm.org/10.1145/1352789.1352790 [accessed 2008-09-05].

[728] Agostino Forestiero, Carlo Mastroianni, and Giandomenico Spezzano. Qos-based dissemination of content in grids. *Future Generation Computer Systems*, 24(3):235–244, March 2008. ISSN: 0167-739X. doi:10.1016/j.future.2007.05.003. Online available at http://grid.deis.unical.it/papers/pdf/ForestieroMastroianniSpezzano-FGCS-QoS.pdf [accessed 2008-09-05].

[729] Agostino Forestiero, Carlo Mastroianni, and Giandomenico Spezzano. Reorganization and discovery of grid information with epidemic tuning. *FGCS – The International Journal of Grid Computing: Theory, Methods and Applications*, 24(8):788–797, October 2008. ISSN: 0167-739X. doi:10.1016/j.future.2008.04.001. Online available at http://grid.deis.unical.it/papers/pdf/FGCS-epidemic.pdf [accessed 2008-09-05].

[730] Stephanie Forrest, editor. *Proceedings of the 5th International Conference on Genetic Algorithms, ICGA*, July 17–21, 1993, University of Illinois at Urbana-Champaign, IL, USA. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN: 1-5586-0299-2.

[731] Stephanie Forrest and Melanie Mitchell. Relative building-block fitness and the building-block hypothesis. In *Foundations of Genetic Algorithms 2*, pages 109–126, 1992. In proceedings [2209]. Online available at http://web.cecs.pdx.edu/~mm/Forrest-Mitchell-FOGA.pdf and http://citeseer.ist.psu.edu/39682.html [accessed 2007-08-13].

[732] Christian V. Forst, Christian Reidys, and Jacqueline Weber. Evolutionary dynamics and optimization: Neutral networks as model-landscapes for RNA secondary-structure folding-landscapes. In *European Conference on Artificial Life*, pages 128–147, 1995. In proceedings [1450]. Online available at http://arxiv.org/abs/adap-org/9505002v1 and http://citeseer.ist.psu.edu/8900.html [accessed 2008-06-01].

[733] Richard Forsyth. BEAGLE a darwinian approach to pattern recognition. *Kybernetes*, 10:159–166, 1981. Received December 17, 1980. Online available at http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/kybernetes_forsyth.pdf [accessed 2007-11-01] (copy from British Library May 1994).

[734] Richard Forsyth. The evolution of intelligence. In Richard Forsyth, editor, *Machine Learning, Priciples and Techniques*, chapter 4, pages 65–82. Chapman and Hall, 1989. ISBN: 0-4123-0570-4, 0-4123-0580-1. Refers also to PC/BEAGLE, see [733].

[735] Richard Forsyth and Roy Rada. *Machine Learning applications in Expert Systems and Information Retrieval*. Ellis Horwood series in Artificial Intelligence. Ellis Horwood, Chichester, UK, 1986. ISBN: 0-7458-0045-9, 0-4702-0309-9. Contains chapters on BEAGLE, see [733]. Also published by John Wiley & Sons Australia (May 28, 1986) and Halsted Press, New York, NY, USA.

[736] James A. Foster. Review: Discipulus: A commercial genetic programming system. *Genetic Programming and Evolvable Machines*, 2(2):201–203, June 2001. ISSN: 1389-2576 (Print) 1573-7632 (Online). doi:10.1023/A:1011516717456. Online available at http://www.springerlink.com/content/l00350408437u708/fulltext.pdf [accessed 2008-09-17].

[737] James A. Foster, Evelyne Lutton, Julian Francis Miller, Conor Ryan, and Andrea Tettamanzi, editors. *Proceedings of the 5th European Conference on Genetic Programming, EuroGP 2002*, volume 2278/2002 of *Lecture Notes in Computer Science (LNCS)*, April 3-5, 2002, Kinsale, Ireland. Springer Berlin/Heidelberg. ISBN: 3-5404-3378-3.

[738] Dieter Fox and Carla P. Gomes, editors. *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI*, June 13–17, 2008, Chicago, Illinois. AAAI Press. ISBN: 978-1-57735-368-3. See http://www.aaai.org/Conferences/AAAI/aaai08.php [accessed 2009-02-27].

[739] John Fox. *Applied Regression Analysis, Linear Models, and Related Methods*. Sage Publications, Inc, New York, subsequent edition, February 1997. ISBN: 978-0-80394-540-1.

[740] Frank D. Francone, Markus Conrads, Wolfgang Banzhaf, and Peter Nordin. Homologous crossover in genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1021–1026, 1999. Online available at http://www.cs.bham.ac.uk/~wbl/biblio/gecco1999/GP-463.pdf [accessed 2008-06-16].

[741] Daniel Raymond Frantz. *Nonlinearities in Genetic Adaptive Search*. PhD thesis, The University of Michigan, Ann Arbor, MI, USA, September 1972. Order No. AAI7311116. Chairman: John Henry Holland. Published as Technical Report Nr. 138 and in Dissertations International 33(11), 5240B-5241B, University Microfilms No. 73-11116. Online available at http://hdl.handle.net/2027.42/4950 [accessed 2007-10-31].

[742] Alex S. Fraser. Simulation of genetic systems by automatic digital computers. *Australian Journal of Biological Science*, 10:484–491, 1957. ISSN: 0004-9417.

[743] William J. Frawley, Gregory Piatetsky-Shapiro, and Christopher J. Matheus. Knowledge discovery in databases: An overview. *AI Magazine*, pages 213–228, Fall 1992. ISSN: 0738-4602. Online available at http://citeseer.ist.psu.edu/524488.html and http://www.kdnuggets.com/gpspubs/aimag-kdd-overview-1992.pdf [accessed 2007-08-11].

[744] Alex A. Freitas. A genetic programming framework for two data mining tasks: Classification and generalized rule induction. In *Genetic Programming 1997: Proceedings of the Second Annual Conference GP-97*, pages 96–101, 1997. In proceedings [1208]. Online available at http://citeseer.ist.psu.edu/43454.html [accessed 2007-09-09].

[745] Yoav Freund. Boosting a weak learning algorithm by majority. In *COLT: Proceedings of the third annual Workshop on Computational Learning Theory*. Morgan Kaufmann Publishers, August 8–9, 1990, University of Rochester, Rochester, New York, USA. ISBN: 1-5586-0146-5. Online available at http://citeseer.ist.psu.edu/freund95boosting.html [accessed 2007-09-15].

[746] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *European Conference on Computational Learning Theory*, volume 904 of *Lecture Notes In Computer Science (LNCS)*, pages 23–37. Springer, 1995. ISBN: 3-5405-9119-2. Online available at http://citeseer.ist.psu.edu/freund95decisiontheoretic.html and http://www.cs.princeton.edu/~schapire/uncompress-papers.cgi/FreundSc95.ps [accessed 2007-09-15]. See also [748].

[747] Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning, ICML'96*, pages 148–156. Morgan Kaufmann, July 3–6, 1996, Bari, Italy. ISBN: 1-5586-0419-7. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?cid=45368 and http://www.cs.ucsb.edu/~yfwang/courses/cs290i/papers/freund-adaboost96.pdf [accessed 2008-06-23].

[748] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55:119–139, 1997. Article no. SS971504. Online available at http://www.face-rec.org/algorithms/Boosting-Ensemble/decision-theoretic_generalization.pdf [accessed 2007-09-15]. See also [746].

[749] Clemens Frey. *Virtual Ecosystems – Evolutionary and Genetic Programming from the perspective of modern means of ecosystem-modelling*, volume 93 of *Bayreuth Forum*

Ecology. Institute for Terrestrial Ecosystems, Bayreuth, Bayreuth, Germany, 2002.

[750] Richard M. Friedberg. A learning machine: Part i. *IBM Journal of Research and Development*, 2:2–13, November 1958. Online available at http://www.research.ibm.com/journal/rd/021/ibmrd0201B.pdf [accessed 2007-09-06]. See also [751].

[751] Richard M. Friedberg, B. Dunham, and J. H. North. A learning machine: Part ii. *IBM Journal of Research and Development*, 3(3):282–287, March 1959. Online available at http://www.research.ibm.com/journal/rd/033/ibmrd0303H.pdf [accessed 2007-09-06]. See also [750].

[752] Drew Fudenberg and Jean Tirole. *Game Theory*. The MIT Press, August 1991. ISBN: 0-2620-6141-4, 978-0-26206-141-4.

[753] Cory Fujiko and John Dickinson. Using the genetic algorithm to generate lisp source code to solve the prisoner's dilemma. In *Proceedings of the Second International Conference on Genetic algorithms and their application*, pages 236–240, 1987. In proceedings [857].

[754] Cory Fujuki. An evaluation of hollands genetic algorithm applied to a program generator. Master's thesis, Department of Computer Science, University of Idaho, Moscow, ID, USA, 1986.

[755] Alex S. Fukunaga and Andre D. Stechert. An evolutionary optimization system for spacecraft design. In *IEA/AIE'1997: Proceedings of the 10th international conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, pages 1–6. Goose Pond Press, 1997, Atlanta, Georgia. ISBN: 9-0569-9615-0. Online available at http://alexf04.maclisp.org/Fukunaga-Stechert-IEA-1997.pdf and http://de.scientificcommons.org/17874369 [accessed 2007-08-24].

G

[756] Bogdan Gabrys, Robert J. Howlett, and Lakhmi C. Jain, editors. *Proceedings of the 10th International Conference on Knowledge-Based Intelligent Information and Engineering Systems, KES 2006, Part I*, volume 4251 of *Lecture Notes in Computer Science (LNCS)*, October 9–11, 2006, Bournemouth, UK. Springer. ISBN: 3-5404-6535-9. See also [757, 758].

[757] Bogdan Gabrys, Robert J. Howlett, and Lakhmi C. Jain, editors. *Proceedings of the 10th International Conference on Knowledge-Based Intelligent Information and Engineering Systems, KES 2006, Part II*, volume 4252 of *Lecture Notes in Computer Science (LNCS)*, October 9–11, 2006, Bournemouth, UK. Springer. ISBN: 3-5404-6537-5. See also [756, 758].

[758] Bogdan Gabrys, Robert J. Howlett, and Lakhmi C. Jain, editors. *Proceedings of the 10th International Conference on Knowledge-Based Intelligent Information and Engineering Systems, KES 2006, Part III*, volume 4253 of *Lecture Notes in Computer Science (LNCS)*, October 9–11, 2006, Bournemouth, UK. Springer. ISBN: 3-5404-6542-1. See also [756, 757].

[759] Pablo Galiasso and Roger L. Wainwright. A hybrid genetic algorithm for the point to multipoint routing problem with single split paths. In *SAC'01: Proceedings of the 2001 ACM symposium on Applied computing*, pages 327–332, March 11–14, 2001, Las Vegas, Nevada, United States. ACM, New York, NY, USA. ISBN: 1-5811-3287-5. doi:10.1145/372202.372354. Online available at http://doi.acm.org/10.1145/372202.372354 and http://citeseer.ist.psu.edu/451433.html [accessed 2008-07-21].

[760] E. A. Galperin. Pareto analysis vis-à-vis balance space approach in multiobjective global optimization. *Journal of Optimization Theory and Applications*, 93(3):533–545, June 1997. ISSN: 0022-3239 (Print) 1573-2878 (Online). doi:10.1023/A:1022639028824. Online available at http://www.springerlink.com/content/m71638106736h581/fulltext.pdf and http://dx.doi.org/10.1023/A:1022639028824 [accessed 2007-09-10].

[761] E. A. Galperin and E. J. Kansa. Application of global optimization and radial basis functions to numerical solutions of weakly singular volterra integral equations. *Computers & Mathematics with Applications*, 43(3-5):491–499, February-March 2002. doi:10.1016/S0898-1221(01)00300-5. Online available at http://dx.doi.org/10.1016/S0898-1221(01)00300-5 [accessed 2007-09-01].

[762] Roberto Galski, Fabiano de Sousa, Fernando Ramos, and Issamu Muraoka. Spacecraft thermal design with the generalized extremal optimization algorithm. *Inverse Problems in Science and Engineering*, 15(1):61–75, January 2007. doi:10.1080/17415970600573924. Online available at http://www.lmt.coppe.ufrj.br/ipdo/papers08-08-24 [accessed .]

[763] Luca Maria Gambardella and Marco Dorigo. Solving symmetric and asymmetric TSPs by ant colonies. In *International Conference on Evolutionary Computation*, pages 622–627, 1996. In proceedings [1006]. Online available at http://citeseer.ist.psu.edu/gambardella96solving.html and http://www.idsia.ch/~luca/icec96-acs.pdf [accessed 2007-08-05].

[764] Luca Maria Gambardella, Éric D. Taillard, and Marco Dorigo. Ant colonies for the quadratic assignment problem. *The Journal of the Operational Research Society*, 50(2):167–176, February 1999. doi:10.2307/3010565. Online available at http://www.idsia.ch/~luca/tr-idsia-4-97.pdf and http://citeseer.ist.psu.edu/378986.html [accessed 2007-08-19].

[765] Luca Maria Gambardella, Andrea E. Rizzoli, Fabrizio Oliverio, Norman Casagrande, Alberto V. Donati, Roberto Montemanni, and Enzo Lucibello. Ant colony optimization for vehicle routing in advanced logistics systems. In *Proceedings of MAS 2003 International Workshop on Modelling and Applied Simulation*, pages 3–9, October 2003, Bergeggi, Italy. Online available at http://www.idsia.ch/~luca/MAS2003_18.pdf [accessed 2007-08-19].

[766] Xavier Gandibleux, Marc Sevaux, Kenneth Sörensen, and Vincent T'kindt, editors. *Metaheuristics for Multiobjective Optimisation*, volume 535 of *Lecture Notes in Economics and Mathematical Systems*. Springer, Berlin, January 2004. ISBN: 3-5402-0637-X, 978-3-54020-637-8.

[767] Q. Gao, K. J. Blow, D. J. Holding, and I. Marshall. Analysis of energy conservation in sensor networks. *Wireless Networks*, 11(6):787–794, November 2005. ISSN: 1022-0038 (Print) 1572-8196 (Online). Online available at http://www.cs.kent.ac.uk/pubs/2005/2193/content.pdf and http://portal.acm.org/citation.cfm?id=1160416 [accessed 2007-08-01].

[768] Yang Gao, Joshua Zhexue Huang, Hongqiang Rong, and Daqian Gu. Learning classifier system ensemble for data mining. In *GECCO'05: Proceedings of the 2005 workshops on Genetic and evolutionary computation*, pages 63–66, 2005. doi:10.1145/1102256.1102268. In proceedings [199]. Online available at http://doi.acm.org/10.1145/1102256.1102268 and http://www.cs.bham.ac.uk/~wbl/biblio/gecco2005wks/papers/0063.pdf [accessed 2007-09-12].

[769] Yong Gao and Joseph Culberson. An analysis of phase transition in nk landscapes. *Journal of Artificial Intelligence Research (JAIR)*, 17:309–332, October 2002. ISSN: 1076-9757. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.20.2690 [accessed http://www.jair.org/media/1081/live-1081-2104-jair.pdf]2009-02-26.

[770] Yu Gao and Yong-Jun Wang. A memetic differential evolutionary algorithm for high dimensional functions' optimization. In *Third International Conference on Natural Computation (ICNC 2007), Part 4*, pages 188–192, 2007. doi:10.1109/ICNC.2007.60. In proceedings [998].

[771] Yuelin Gao and Yuhong Duan. An adaptive particle swarm optimization algorithm with new random inertia weight. In *Advanced Intelligent Computing Theories and Applications. With Aspects of Contemporary Intelligent Computing Techniques*, volume 2, pages 342–350, 2007. doi:10.1007/978-3-540-74282-1. In proceedings [970].

[772] Yuelin Gao and Zihui Ren. Adaptive particle swarm optimization algorithm with genetic mutation operation. In *Proceedings of the Third International Conference on Natural Computation (ICNC 2007)*, volume 2, pages 211–215, 2007. doi:10.1109/ICNC.2007.161. In proceedings [996].

[773] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Series of Books in the Mathematical Sciences. W. H. Freeman & Co., New York, USA, January 1979. ISBN: 0-7167-1045-5, 978-0-71671-045-5.

[774] Josselin Garnier and Leila Kallel. Statistical distribution of the convergence time of evolutionary algorithms for long-path problems. *IEEE Transactions on Evolutionary Computation*, 4(1):16–30, April 2000. ISSN: 1089-778X. CODEN: ITEVF5. INSPEC Accession Number: 6617599. doi:10.1109/4235.843492. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.31.4329 [accessed 2008-08-13].

[775] Sergey Gavrilets. *Fitness Landscapes and the Origin of Species*. Number MPB-41 in Monographs in Population Biology. Princeton University Press, July 2004. ISBN: 0-6911-1983-X.

[776] Nicholas Geard. An exploration of NK landscapes with neutrality. Technical Report 14213, School of Information Technology and Electrical Engineering, University of Queensland, St Lucia Q 4072, Australia, October 2001. Supervisor: Janet Wiles. Online available at http://eprints.ecs.soton.ac.uk/14213/1/ng_thesis.pdf [accessed 2008-07-01].

[777] Nicholas Geard, Janet Wiles, Jennifer Hallinan, Bradley Tonkes, and Ben Skellett. A comparison of neutral landscapes – NK, NKp and NKq. In *Congress on Evolutionary Computation (CEC2002)*, pages 205–210, 2002. In proceedings [703]. Online available at http://citeseer.ist.psu.edu/506371.html and http://eprints.ecs.soton.ac.uk/14208/1/cec1-comp.pdf [accessed 2008-06-01].

[778] Daniel K. Gehlhaar and David B. Fogel. Tuning evolutionary programming for conformationally flexible molecular docking. In *Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming*, pages 419–429, 1996. In proceedings [709].

[779] Daniel K. Gehlhaar, Gennady Verkhivker, Paul A. Rejto, David B. Fogel, Lawrence Jerome Fogel, and Stephan T. Freer. Docking conformationally flexible small molecules into a protein binding site through evolutionary programming. In *Evolutionary Programming IV: Proceedings of the Fourth Annual Conference on Evolutionary Programming*, 1995. In proceedings [1380].

[780] Kurt Geihs. Why robots play soccer (keynote extended abstract). In Judith Bishop and Trish Alexander, editors, *Annual Conference of the South African Institute of Computer Scientists and Information Technologists (SAICSIT 2006)*, International Proceedings Series. SAICSIT, ACM, 2006. ISBN: 1-5959-3567-3.

[781] Alexander F. Gelbukh and Carlos A. Reyes García, editors. *Proceedings of the MICAI 2006: Advances in Artificial Intelligence, 5th Mexican International Conference on Artificial Intelligence*, volume 4293/2006 of *Lecture Notes in Computer Science (LNCS), Subseries Lecture Notes in Artificial Intelligence (LNAI)*, November 13–17, 2006, Apizaco, México. Springer Berlin / Heidelberg. ISBN: 3-5404-9026-4, 978-3-54049-026-5. doi:10.1007/11925231. See also [493].

[782] Alexander F. Gelbukh and Angel Fernando Kuri Morales, editors. *Proceedings of the MICAI 2007: Advances in Artificial Intelligence, 6th Mexican International Conference on Artificial Intelligence*, volume 4827/2007 of *Lecture Notes in Computer Science (LNCS), Subseries Lecture Notes in Artificial Intelligence (LNAI)*, November 4–10, 2007, Aguascalientes, México. Springer Berlin / Heidelberg. ISBN: 978-3-54076-630-8. doi:10.1007/978-3-540-76631-5.

[783] Alexander F. Gelbukh, Alvaro de Albornoz, and Hugo Terashima-Marín, editors. *Proceedings of the MICAI 2005: Advances in Artificial Intelligence, 4th Mexican Inter-*

national Conference on Artificial Intelligence, volume 3789/2005 of *Lecture Notes in Computer Science (LNCS), Subseries Lecture Notes in Artificial Intelligence (LNAI)*, November 14–18, 2005, Monterrey, México. Springer Berlin / Heidelberg. ISBN: 3-5402-9896-7, 978-3-54029-896-0. doi:10.1007/11579427.

[784] D. Gelperin and B. Hetzel. The growth of software testing. *Communications of the ACM*, 31:687–695, 1988. ISSN: 0001-0782. Online available at http://doi.acm.org/10.1145/62959.62965 [accessed 2007-09-15].

[785] Stuart Geman, Elie Bienenstock, and René Doursat. Neural networks and the bias/-variance dilemma. *Neural Computation*, 4(1):1–58, 1992. ISSN: 0899-7667.

[786] Mitsuo Gen and Runwei Chen. *Genetic Algorithms (Engineering Design and Automation)*. Wiley Series in Engineering Design and Automation. John Wiley and Sons Ltd, February 2001. ISBN: 978-0-47131-531-5.

[787] Mitsuo Gen and Runwei Cheng. *Genetic Algorithms and Engineering Design*. Wiley-Interscience, January 7, 1997. ISBN: 0-4711-2741-8, 978-0-47112-741-3. Partly online available at http://books.google.de/books?id=MCHCaJAHFJAC [accessed 2008-04-01].

[788] Michael R. Genesereth, editor. *Proceedings of the National Conference on Artificial Intelligence, AAAI*, August 22–26, 1983, Washington, DC, USA. AAAI Press. ISBN: 0-2625-1052-9. See http://www.aaai.org/Conferences/AAAI/aaai83.php [accessed 2007-09-06].

[789] Ingrid Gerdes, Klaus Klawonn, and Rudolf Kruse. *Evolutionäre Algorithmen*. Friedrich Vieweg & Sohn Verlag / GWV Fachverlage GmbH (Springer Science+Business Media), Wiesbaden, Germany, July 2004. ISBN: 3-5280-5570-7.

[790] Brian P. Gerkey, Sebastian Thrun, and Geoff Gordon. Parallel stochastic hill-climbing with small teams. In Lynne E. Parker, Frank E. Schneider, and Alan C. Schultz, editors, *Proceedings from the 2005 International Workshop on Multi-Robot Systems*, volume III of *Multi-Robot Systems: From Swarms to Intelligent Automata*, pages 65–77. Springer, 2005. ISBN: 978-1-40203-388-9. Presented at 2005 International Workshop on Multi-Robot Systems. L.E.Parker et al. Online available at http://www.cs.cmu.edu/~ggordon/gerkey-thrun-gordon.parish.pdf [accessed 2007-08-18].

[791] Hannes Geyer, Peter Ulbig, and Siegfried Schulz. Encapsulated evolution strategies for the determination of group contribution model parameters in order to predict thermodynamic properties. In *PPSN V: Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, pages 978–987, 1998. In proceedings [624].

[792] Hannes Geyer, Peter Ulbig, and Siegfried Schulz. Use of evolutionary algorithms for the calculation of group contribution parameters in order to predict thermodynamic properties – part 2: Encapsulated evolution strategies. *Computers and Chemical Engineering*, 23(7):955–973, July 1, 1999. doi:10.1016/S0098-1354(99)00270-7.

[793] Hannes Geyer, Peter Ulbig, and Siegfried Schulz. Verschachtelte evolutionsstrategien zur optimierung nichtlinearer verfahrenstechnischer regressionsprobleme. *Chemie Ingenieur Technik*, 72(4):369–373, 2000. ISSN: 1522-2640. doi:10.1002/1522-2640(200004)72:4<369::AID-CITE369>3.0.CO;2-W. Online available at http://www3.interscience.wiley.com/cgi-bin/fulltext/76500452/PDFSTART [accessed 2007-08-27].

[794] Andreas Geyer-Schulz. Holland classifier systems. In *APL'95: Proceedings of the international conference on Applied programming languages*, pages 43–55, 1995, San Antonio, Texas, United States. ACM Press, New York, NY, USA. ISBN: 0-8979-1722-7. doi:10.1145/206913.206955. Online available at http://doi.acm.org/10.1145/206913.206955 [accessed 2007-09-12].

[795] Andreas Geyer-Schulz. *Fuzzy Rule-Based Expert Systems and Genetic Machine Learning*, volume 3 of *Studies in Fuzziness and Soft Computing*. Physica-Verlag, 2nd rev edition edition, January 1997. ISBN: 978-3-79080-964-0. First edition: 1995.

[796] Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Pearson Education, Prentice Hall, second edition, September 19, 2002. ISBN: 978-0-13305-699-0.

[797] Ashish Ghosh and Lakhmi C. Jain, editors. *Evolutionary Computation in Data Mining*, volume 163/2005 of *Studies in Fuzziness and Soft Computing*. Springer Berlin / Heidelberg, 2005. ISBN: 978-3-54022-370-2, 3-5402-2370-3. doi:10.1007/3-540-32358-9.

[798] Ashish Ghosh and Shigeyoshi Tsutsui, editors. *Advances in Evolutionary Computing – Theory and Applications*. Natural Computing Series. Springer-Verlag New York, Inc., New York, NY, USA, November 22, 2002. ISBN: 3-5404-3330-9, 978-3-54043-330-9. Series editors: G. Rozenberg, Thomas Bäck, Ágoston E. Eiben, J.N. Kok, and H.P. Spaink. Partly online available at `http://books.google.de/books?id=OGMEMC9P3vMC` [accessed 2008-10-18].

[799] Sukumar Ghosh. *Distributed Systems: An Algorithmic Approach*, volume 12 of *Chapman & Hall/CRC Computer & Information Science Series*. CRC Press, 2006. ISBN: 1-5848-8564-5, 978-1-58488-564-1. Cat. #: C5645. Partly online available at `http://books.google.de/books?id=aVjVzuav7cIC` [accessed 2008-11-25].

[800] Mario Giacobini, Anthony Brabazon, Stefano Cagnoni, Gianni Di Caro, Rolf Drechsler, Muddassar Farooq, Andreas Fink, Evelyne Lutton, Penousal Machado, Stefan Minner, Michael O'Neill, Juan Romero, Franz Rothlauf, Giovanni Squillero, Hideyuki Takagi, Sima Uyar, and Shengxiang Yang, editors. *Applications of Evolutionary Computing, Proceedings of EvoWorkshops 2007: EvoCoMnet, EvoFIN, EvoIASP, EvoINTERACTION, EvoMUSART, EvoSTOC and EvoTransLog*, volume 4448/2007 of *Lecture Notes in Computer Science (LNCS)*, April 11–13, 2007, Valencia, Spain. Springer, Berlin / Heidelberg. ISBN: 978-3-54071-804-8.

[801] Mario Giacobini, Anthony Brabazon, Stefano Cagnoni, Gianni A. Di Caro, Rolf Drechsler, Anikó Ekárt, Anna I Esparcia-Alcázar, Muddassar Farooq, Andreas Fink, Jon McCormack, Michael ONeill, Juan Romero, Franz Rothlauf, Giovanni Squillero, A. Şima Uyar, and Shengxiang Yang, editors. *Applications of Evolutionary Computing – Proceedings of EvoWorkshops 2008: EvoCOMNET, EvoFIN, EvoHOT, EvoIASP, EvoMUSART, EvoNUM, EvoSTOC, and EvoTransLog*, volume 4974/2008 of *Lecture Notes in Computer Science (LNCS)*, March 26–28, 2008, Naples, Italy. Springer, Berlin / Heidelberg. ISBN: 3-5407-8760-7, 978-3-54078-760-0. doi:10.1007/978-3-540-78761-7.

[802] Mario Giacobini, Penousal Machado, Anthony Brabazon, Jon McCormack, Stefano Cagnoni, Michael O'Neill, Gianni A. Di Caro, Ferrante Neri, Anikó Ekárt, Mike Preuss, Anna I Esparcia-Alcázar, Franz Rothlauf, Muddassar Farooq, Ernesto Tarantino, Andreas Fink, and Shengxiang Yang, editors. *Applications of Evolutionary Computing – Proceedings of EvoWorkshops 2009: EvoCOMNET, EvoENVIRONMENT, EvoFIN, EvoGAMES, EvoHOT, EvoIASP, EvoINTERACTION, EvoMUSART, EvoNUM, EvoSTOC, EvoTRANSLOG*, volume 5484/2009 of *Lecture Notes in Computer Science (LNCS), Subseries: SL 1 – Theoretical Computer Science and General Issues*, April 15–17, 2009, Tübingen, Germany. Springer, Berlin / Heidelberg. ISBN: 3-6420-1128-4, 978-3-64201-128-3. doi:10.1007/978-3-642-01129-0.

[803] K. C. Giannakoglou, D. T. Tsahalis, J. Périaux, K. D. Papailiou, and T. Fogarty, editors. *Proceedings of the EUROGEN2001 Conference: Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems*, September 19–21, 2001, Athens, Greece. International Center for Numerical Methods in Engineering (Cmine), Gran Capitan s/n, 08034 Barcelona, Spain. ISBN: 8-4899-2597-6. Published in 2002. See `http://www.mech.ntua.gr/~eurogen2001` [accessed 2007-09-16].

[804] Basilis Gidas. Nonstationary markov chains and convergence of the annealing algorithm. *Journal of Statistical Physics*, 39(1-2):73–131, April 1985. ISSN: 0022-4715 (Print) 1572-9613 (Online). doi:10.1007/BF01007975. Online available

at http://www.springerlink.com/content/x3821j328gv33085/fulltext.pdf [accessed 2008-03-26].

[805] Yolanda Gil and Raymond J. Mooney, editors. *Proceedings, The Twenty-First National Conference on Artificial Intelligence (AAAI) and the Eighteenth Innovative Applications of Artificial Intelligence Conference (IAAI)*, July 16–20, 2006, Boston, Massachusetts, USA. AAAI Press. See http://www.aaai.org/Conferences/AAAI/aaai06.php [accessed 2007-09-06] and http://www.aaai.org/Conferences/IAAI/iaai06.php [accessed 2007-09-06].

[806] Walter Gilbert. Why genes in pieces. *Nature*, 271(5645):501, February 9, 1978. doi:10.1038/271501a0.

[807] John H. Gillespie. Molecular evolution over the mutational landscape. *Evolution*, 38 (5):1116–1129, September 1984. doi:10.2307/2408444.

[808] Jean-Numa Gillet and Yunlong Sheng. Simulated quenching with temperature rescaling for designing diffractive optical elements. In Alexander J. Glass, Joseph W. Goodman, Milton Chang, Arthur H. Guenther, and Toshimitsu Asakura, editors, *Proceedings of the 18th Congress of the International Commission for Optics*, volume 3749 of *Proceedings of SPIE*, pages 683–684, July 1999.

[809] Marie-Pierre Gleizes, Valérie Camps, and Pierre Glize. A theory of emergent computation based on cooperative self-organization for adaptive artificial systems. In *Fourth European Congress of Systems Science*. National Ministry of Education Online Cooperation Center for Artificial Intelligence, September 20–24, 1999, Valencia, Spain. Online available at ftp://ftp.irit.fr/pub/IRIT/SMAC/DOCUMENTS/PUBLIS/ECS99.pdf and http://www.irit.fr/ACTIVITES/EQ_SMI/SMAC/Publications.html [accessed 2008-10-08].

[810] Fred Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549, 1986. ISSN: 0305-0548. doi:10.1016/0305-0548(86)90048-1. Online available at http://dx.doi.org/10.1016/0305-0548(86)90048-1 [accessed 2008-03-27].

[811] Fred Glover. Tabu search – part i. *Operations Research Society of America (ORSA) Journal on Computing*, 1(3):190–206, Summer 1989. Online available at http://leeds-faculty.colorado.edu/glover/TS%20-%20Part%20I-ORSA.pdf [accessed 2008-03-27].

[812] Fred Glover. Tabu search – part ii. *Operations Research Society of America (ORSA) Journal on Computing*, 2(1):190–206, Winter 1990. Online available at http://leeds-faculty.colorado.edu/glover/TS%20-%20Part%20II-ORSA.pdf [accessed 2008-03-27].

[813] Fred Glover and Gary A. Kochenberger, editors. *Handbook of Metaheuristics*, volume 57 of *International Series in Operations Research & Management Science*. Kluwer Academic Publishers / Springer, New York, USA, 2003. ISBN: 978-1-40207-263-5, 978-0-30648-056-0, 0-3064-8056-5, 1-4020-7263-5. doi:10.1007/b101874. Series Editor Frederick S. Hillier.

[814] Fred Glover and M. Laguna. Tabu search. In *Handbook of Combinatorial Optimization*, volume 3, pages 621–757. Kluwer Academic Publishers, Springer Netherlands, 1998. In collection [1612].

[815] Fred Glover and Manuel Laguna. Tabu search. In *Modern Heuristic Techniques for Combinatorial Problems*. John Wiley & Sons, 1993. ISBN: 978-0-47022-079-5, 0-7923-9965-X. In collection [1716]. Partly online available at http://citeseer.ist.psu.edu/glover97tabu.html [accessed 2007-09-16]. Online available at http://www.dei.unipd.it/~fisch/ricop/tabu_search_glover_laguna.pdf [accessed 2008-11-06].

[816] Fred Glover, Éric D. Taillard, and Dominique de Werra. A user's guide to tabu search. *Annals of Operations Research*, 41(1):3–28, 1993. ISSN: 0254-5330, 1572-9338. doi:10.1007/BF02078647. Special issue on Tabu search. Online available

at http://www.springerlink.com/content/k607q47112622n45/fulltext.pdf [accessed 2008-08-29].

[817] Helen G. Gobb and John J. Grefenstette. Genetic algorithms for tracking changing environments. In *Proceedings of the 5th International Conference on Genetic Algorithms, ICGA*, pages 523–529, 1993. In proceedings [730]. Online available at ftp://ftp.aic.nrl.navy.mil/pub/papers/1993/AIC-93-004.ps [accessed 2008-07-19].

[818] William L. Goffe, Gary D. Ferrier, and John Rogers. Global optimization of statistical functions with simulated annealing. *Journal of Econometrics*, 60(1-2):65–99, January-February 1994. Online available at http://dx.doi.org/10.1016/0304-4076(94)90038-8 and http://cook.rfe.org/anneal-preprint.pdf [accessed 2007-09-15]. See also http://econpapers.repec.org/software/codfortra/simanneal.htm [accessed 2007-09-15].

[819] J. Peter Gogarten and Elena Hilario. Inteins, introns, and homing endonucleases: recent revelations about the life cycle of parasitic genetic elements. *BMC Evolutionary Biology*, 6(94), 2006. doi:10.1186/1471-2148-6-94. Online available at http://dx.doi.org/10.1186/1471-2148-6-94 and http://www.biomedcentral.com/content/pdf/1471-2148-6-94.pdf [accessed 2008-03-23].

[820] David E. Goldberg. *Computer-aided gas pipeline operation using genetic algorithms and rule learning*. PhD thesis, University of Michigan. Ann Arbor, MI, 1983.

[821] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, January 1989. ISBN: 0-2011-5767-5.

[822] David E. Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Proceedings of the First Workshop on Foundations of Genetic Algorithms FOGA*, pages 69–93, 1990. In proceedings [1924]. Online available at http://www.cse.unr.edu/~sushil/class/gas/papers/Select.pdf [accessed 2007-08-24].

[823] David E. Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Proceedings of Foundations of Genetic Algorithms*, pages 69–93, 1990. In proceedings [1924]. Online available at http://www.cse.unr.edu/~sushil/class/gas/papers/Select.pdf [accessed 2007-07-29].

[824] David E. Goldberg and Jon Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms and their application*, pages 41–49, 1987. In proceedings [857].

[825] David E. Goldberg, Kalyanmoy Deb, and Bradley Korb. Messy genetic algorithms: motivation, analysis, and first results. *Complex Systems*, 3(5):493–530, 1989. ISSN: 0891-2513.

[826] David E. Goldberg, Kalyanmoy Deb, and Bradley Korb. Messy genetic algorithms revisited: Studies in mixed size and scale. *Complex Systems*, 4(4):415–444, 1990. ISSN: 0891-2513.

[827] David E. Goldberg, Kalyanmoy Deb, and Jeffrey Horn. Massive multimodality, deception, and genetic algorithms. In *Parallel Problem Solving from Nature 2*, 1992. In proceedings [1357]. Online available at http://citeseer.ist.psu.edu/goldberg92massive.html [accessed 2008-03-16].

[828] David E. Goldberg, Kalyanmoy Deb, Kalyanmoy Deb, Hillol Kargupta, Hillol Kargupta, and Georges Harik. Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 56–64, 1993. In proceedings [730]. Also: IlliGAL Report 93004, February 1993, Illinois Genetic Algorithms Laboratory (IlliGAL), Department of General Engineering, University of Illinois at Urbana-Champaign, 117 Transportation Building, 104 S. Mathews Avenue, Urbana-Champaign, Illinois, 61801-2996, USA. Online available at http://citeseerx.ist.psu.edu/viewdoc/

summary?doi=10.1.1.49.3524 and https://eprints.kfupm.edu.sa/60781/ [accessed 2008-10-18].

[829] Andrea Goldsmith. *Wireless Communications*. Cambridge University Press, The Edinburgh Building, Shaftesbury Road, Cambridge, CB2 8RU, UK, August 2005. Partly online available at http://books.google.com/books?id=mF7X_8D7_BIC [accessed 2008-07-27].

[830] Asunción Gómez-Pérez, Rafael González-Cabero, and Manuel Lama. Ode sws: A framework for designing and composing semantic web services. *IEEE Intelligent Systems*, 19(4):24–31, July-August 2004. ISSN: 1541-1672. doi:10.1109/MIS.2004.32. Online available at http://iswc2004.semanticweb.org/demos/14/paper.pdf [accessed 2007-09-02]. See also [831].

[831] Asunción Gómez-Pérez, Rafael González-Cabero, and Manuel Lama. A framework for designing and composing semantic web services. In *Semantic Web Services, First International Semantic Web Services Symposium, Proceedings of 2004 AAAI Spring Symposium Series*, March 22–24, 2004, History Corner, main quad (Building 200), Stanford University, CA, USA. Online available at http://www.daml.ecs.soton.ac.uk/SSS-SWS04/44.pdf [accessed 2007-09-02]. See also [830].

[832] Teofilo F. Gonzalez, editor. *Handbook of Approximation Algorithms and Metaheuristics*. Chapmann & Hall/CRC Computer and Information Science Series. Chapmann & Hall/CRC Press (Tailor & Francis Group), 2007. ISBN: 1-5848-8550-5, 978-1-58488-550-4. Partly online available at http://books.google.de/books?id=QK3_VU8ngK8C [accessed 2008-09-10].

[833] Erik D. Goodman, editor. *Late Breaking Papers at Genetic and Evolutionary Computation Conference (GECCO'01)*, July 7–11, 2001, Holiday Inn Golden Gateway Hotel, San Francisco, California, USA. See also [1937].

[834] Janaki Gopalan, Reda Alhajj, and Ken Barker. Discovering accurate and interesting classification rules using genetic algorithm. In Sven F. Crone, Stefan Lessmann, and Robert Stahlbock, editors, *Proceedings of the 2006 International Conference on Data Mining, DMIN 2006*, pages 389–395. CSREA Press, June 2006, Las Vegas, Nevada, USA. ISBN: 1-6013-2004-3. Online available at http://ww1.ucmss.com/books/LFS/CSREA2006/DMI5509.pdf [accessed 2007-07-29].

[835] Martina Gorges-Schleuter. Asparagos: An asynchronous parallel genetic optimization strategy. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 422–427, 1989. In proceedings [1820].

[836] Martina Gorges-Schleuter. Explicit parallelism of genetic algorithms through population structures. In *PPSN I: Proceedings of the 1st Workshop on Parallel Problem Solving from Nature*, pages 150–159, 1991. In proceedings [1842].

[837] James Gosling and Henry McGilton. The java language environment – a white paper. Technical Report, Sun Microsystems, Inc., May 1996. Online available at http://java.sun.com/docs/white/langenv/ [accessed 2007-07-03].

[838] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *Java(TM) Language Specification*. The Java Series. Prentice Hall PTR, 3rd edition, June 2005. ISBN: 978-0-32124-678-3. Online available at http://java.sun.com/docs/books/jls/ [accessed 2007-09-14].

[839] Simon Goss, R. Beckers, Jean-Louis Deneubourg, S. Aron, and Jacques M. Pasteels. How trail laying and trail following can solve foraging problems for ant colonies. In R. N. Hughes, editor, *Behavioural Mechanisms of Food Selection*, NATO ASI Series, G 20, pages 661–678. Springer-Verlag, Berlin, April 1990. ISBN: 0-3875-1762-6, 978-0-38751-762-9.

[840] William Sealy Gosset. The probable error of a mean. *Biometrika*, 6(1):1–25, March 1908. Because the author was not allowed to publish this article, he used the pseudonym *Student*. Online available at http://www.york.ac.uk/depts/maths/histstat/student.pdf [accessed 2007-09-30]. Reprinted in [841].

[841] William Sealy Gosset. The probable error of a mean. In E. S. Pearson and John Wishart, editors, *"Student's" Collected Papers*, pages 11–34. Cambridge University Press for the Biometrika Trustees, 1942. With a Foreword by Launce McMullen. Reprint of [840].

[842] Jens Gottlieb and Günther R. Raidl, editors. *Proceedings of the 4th European Conference on Evolutionary Computation in Combinatorial Optimization, EvoCOP 2004*, volume 3004/2004 of *Lecture Notes in Computer Science (LNCS)*, April 5–7, 2004, Coimbra, Portugal. Springer. ISBN: 3-5402-1367-8, 978-3-54021-367-3. doi:10.1007/b96499.

[843] Jens Gottlieb and Günther R. Raidl, editors. *Proceedings of the 6th European Conference on Evolutionary Computation in Combinatorial Optimization, EvoCOP 2006*, volume 3906/2006 of *Lecture Notes in Computer Science (LNCS)*, April 10–12, 2006, Budapest, Hungary. Springer. ISBN: 3-5403-3178-6.

[844] Georg Gottlob and Toby Walsh, editors. *IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence*, August 9–15, 2003, Acapulco, México. Morgan Kaufmann, San Francisco, CA, USA. ISBN: 0-1270-5661-0. Online available at http://dli.iiit.ac.in/ijcai/IJCAI-2003/content.htm [accessed 2008-04-01]. See also [1086].

[845] Harvey Gould and Jan Tobochnik. *An Introduction to Computer Simulation Methods: Part 2, Applications to Physical Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995. ISBN: 0-2015-0604-1, 978-0-20116-503-6. Edited by Julia Berrisford.

[846] Ramesh Govindan, Joseph M. Hellerstein, Wei Hong, Samuel Madden, Michael Franklin, and Scott Shenker. The sensor network as a database. Technical Report 02-771, University of South California, Computer Science Department, September 2002. Online available at ftp://ftp.usc.edu/pub/csinfo/tech-reports/papers/02-771.pdf and http://citeseer.ist.psu.edu/601935.html [accessed 2007-08-01].

[847] Paul Grace. Genetic programming and protocol configuration. Master's thesis, Computing Department, Lancaster University, Lancaster, LA1 4YW, Lancashire, UK, September 2000. Supervisor: Professor Gordon Blair. Online available at http://www.lancs.ac.uk/postgrad/gracep/msc.pdf [accessed 2008-06-23].

[848] Peter Graf and Daniel L. Schacter. Implicit and explicit memory for new associations in normal and amnesic subjects. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 11(3):501–518, July 1985. ISSN: 0278-7393.

[849] Pierre-Paul Grassé. La reconstruction du nid et les coordinations inter-individuelles chez bellicositermes natalensis et cubitermes sp. la théorie de la stigmergie: Essai d'interprétation des termites constructeurs. *Insectes Sociaux*, 6:41–48, 1959, Paris.

[850] Adam Greenfield. *Everyware: The Dawning Age of Ubiquitous Computing*. New Riders Publishing, March 20, 2006. ISBN: 0-3213-8401-6, 978-0-32138-401-0.

[851] Garrison W. Greenwood, Xiaobo Sharon Hu, and Joseph G. D'Ambrosio. Fitness functions for multiple objective optimization problems: Combining preferences with pareto rankings. In *Foundations of Genetic Algorithms 4*, pages 437–455, 1996. In proceedings [172].

[852] John J. Grefenstette. Credit assignment in rule discovery systems based on genetic algorithms. *Zeitschrift Machine Learning*, 3:225–245, October 1988. ISSN: 0885-6125 (Print) 1573-0565 (Online). doi:10.1023/A:1022614421909. Received: December 21, 1987, Revised: July 15, 1988. Online available at http://www.springerlink.com/content/n83r83v087685215/fulltext.pdf [accessed 2008-05-29].

[853] John J. Grefenstette. Conditions for implicit parallelism. In *Foundations of genetic algorithms*, pages 252–261, 1990. In proceedings [1924]. Online available at http://citeseer.ist.psu.edu/41022.html [accessed 2007-07-29].

[854] John J. Grefenstette. Deception considered harmful. In *Proceedings of the Second Workshop on Foundations of Genetic Algorithms*, pages 75–91,

1992. In proceedings [2209]. Online available at http://citeseer.ist.psu.edu/grefenstette92deception.html [accessed http://www.citeulike.org/user/pduval/article/1433820]2007-08-12.

[855] John J. Grefenstette. Predictive models using fitness distributions of genetic operators. In *Foundations of Genetic Algorithms 3*, pages 139–161, 1994. In proceedings [2214]. Online available at http://reference.kfupm.edu.sa/content/p/r/predictive_models_using_fitness_distribu_672029.pdf and http://citeseer.ist.psu.edu/grefenstette95predictive.html [accessed 2008-07-20].

[856] John J. Grefenstette, editor. *Proceedings of the 1st International Conference on Genetic Algorithms and their Applications*, July 24–26, 1985, Carnegie-Mellon University, Pittsburgh, PA, USA. Lawrence Erlbaum Associates, Inc., Mahwah, NJ, USA. ISBN: 0-8058-0426-9.

[857] John J. Grefenstette, editor. *Genetic Algorithms and Their Applications: Proceedings of the Second International Conference on Genetic Algorithms and their Applications*, July 28–31, 1987, Massachusetts Institute of Technology, Cambridge, Massachusetts, USA. American Association for Artificial Intelligence, Lawrence Erlbaum Associates, Inc., Hillsdale/Mahwah, NJ, USA. ISBN: 0-8058-0158-8.

[858] John J. Greffenstette and James E. Baker. How genetic algorithms work: A critical look at implicit parallelism. In *Proceedings of the third International Conference on Genetic Algorithms ICGA*, pages 20–27, 1989. In proceedings [1820].

[859] Horst Greiner. Robust filter design by stochastic optimization. *Proceedings of SPIE (The International Society for Optical Engineering)*, 2253:150–161, November 4, 1994. ISBN: 978-0-81941-562-2. doi:10.1117/12.192107. Advances in Design and Design Techniques: Posters, June 5, 1994, Grenoble, France, Optical Interference Coatings, Florin Abeles, ed.

[860] Horst Greiner. Robust optical coating design with evolutionary strategies. *Applied Optics*, 35(28):5477–5483, October 1, 1996.

[861] Crina Grosan and Ajith Abraham. Hybrid evolutionary algorithms: Methodologies, architectures, and reviews. In *Hybrid Evolutionary Algorithms*, pages 1–17. Springer, 2007. doi:10.1007/978-3-540-73297-6_1. In collection [862].

[862] Crina Grosan, Ajith Abraham, and Hisao Ishibuchi, editors. *Hybrid Evolutionary Algorithms*, volume 75 of *Studies in Computational Intelligence*. Springer Berlin / Heidelberg, 2007. ISBN: 978-3-54073-296-9. doi:10.1007/978-3-540-73297-6. Partly online available at http://books.google.de/books?id=II7LCqiGFlEC [accessed 2009-03-01]. Series editor: Janusz Kacprzyk.

[863] Frederic Gruau. Genetic synthesis of boolean neural networks with a cell rewriting developmental process. In *Proceedings of the Workshop on Combinations of Genetic Algorithms and Neural Networks (COGANN92)*, pages 55–74, 1992. doi:10.1109/COGANN.1992.273948. In proceedings [1823].

[864] Frédéric Gruau and L. Darrell Whitley. Adding learning to the cellular development of neural networks: Evolution and the baldwin effect. *Evolutionary Computation*, 1 (3):213–233, 1993. ISSN: 1063-6560. doi:10.1162/evco.1993.1.3.213. Online available at http://citeseer.ist.psu.edu/gruau93adding.html [accessed 2008-09-12].

[865] Dick Grune and Ceriel J. H. Jacobs. *Parsing techniques a practical guide*. Ellis Horwood Limited, Chichester, England, August 1991. ISBN: 978-0-13651-431-2. Online available at http://citeseer.ist.psu.edu/grune90parsing.html [accessed 2007-09-14].

[866] Frank Guerin and Wamberto Weber Vasconcelos, editors. *AISB 2008 Convention: Communication, Interaction and Social Intelligence*, volume 12 of *Computing & Philosophy*, April1–4, 2008, University of Aberdeen, UK. The Society for the Study of Artificial Intelligence and the Simulation of Behaviour (SSAISB). ISBN: 1-9029-5671-0. Online available at http://www.aisb.org.uk/publications/proceedings.shtml [accessed 2008-09-10].

[867] Juan J. Merelo Guervós, Panagiotis Adamidis, Hans-Georg Beyer, José Luis Fernández-Villacañas Martín, and Hans-Paul Schwefel, editors. *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature – PPSN VII*, volume 2439/2002 of *Lecture Notes in Computer Science (LNCS)*, September 7–11, 2002, Granada, Spain. Springer. ISBN: 3-5404-4139-5, 978-3-54044-139-7. doi:10.1007/3-540-45712-7. See http://ppsn2002.ugr.es/index.html [accessed 2007-09-05].

[868] Michael Guntsch and Martin Middendorf. Pheromone modification strategies for ant algorithms applied to dynamic tsp. In *Proceedings of the EvoWorkshops on Applications of Evolutionary Computing*, pages 213–222, 2001. In proceedings [235]. Online available at http://pacosy.informatik.uni-leipzig.de/pv/Personen/middendorf/Papers/ [accessed 2007-08-19].

[869] Michael Guntsch, Martin Middendorf, and Hartmut Schmeck. An ant colony optimization approach to dynamic TSP. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 860–867, 2001. In proceedings [1937]. Online available at http://citeseer.ist.psu.edu/693476.html [accessed 2007-08-19].

[870] L. S. Gurin and Leonard A. Rastrigin. Convergence of the random search method in the presence of noise. *Automation and Remote Control*, 26:1505–1511, 1965.

[871] Steven Matt Gustafson. *An Analysis of Diversity in Genetic Programming*. PhD thesis, University of Nottingham, School of Computer Science & IT, Nottingham, U.K, February 2004. Online available at http://www.gustafsonresearch.com/thesis_html/index.html and http://citeseer.ist.psu.edu/gustafson04analysis.html [accessed 2008-07-22].

[872] Steven Matt Gustafson, Anikó Ekárt, Edmund K. Burke, and Graham Kendall. Problem difficulty and code growth in genetic programming. *Genetic Programming and Evolvable Machines*, 5(3):271–290, September 2004. ISSN: 1389-2576 (Print) 1573-7632 (Online). doi:10.1023/B:GENP.0000030194.98244.e3. Online available at http://www.gustafsonresearch.com/research/publications/gustafson-gpem2004.pdf and http://citeseer.ist.psu.edu/730858.html [accessed 2008-07-21]. Submitted March 4, 2003; Revised August 7, 2003.

[873] Tomasz Dominik Gwiazda. *Crossover for single-objective numerical optimization problems*, volume 1 of *Genetic Algorithms Reference*. Lomianki, e-book: Tomasz Dominik Gwiazda, May 2006. ISBN: 8-3923-9583-2.

[874] Lorenz Gygax. *Statistik für Nutztierethologen – Einführung in die statistische Denkweise: Was ist, was macht ein statistischer Test?* Zentrum für tiergerechte Haltung, 1.1 edition, June 2003. Online available at http://www.proximate-biology.ch/documents/introEtho.pdf [accessed 2008-08-15].

H

[875] Atidel Ben Hadj-Alouane and James C. Bean. A genetic algorithm for the multiple-choice integer program. Technical Report 92-50, Department of Industrial and Operations Engineering, The University of Michigan, Ann Arbour, MI 48109-2117, USA, September 1992. Revised in July 1993. Online available at http://hdl.handle.net/2027.42/3516 [accessed 2008-11-15].

[876] Yacov Y. Haimes, editor. *Proceedings of the 6th International Conference on Multiple Criteria Decision Making: Decision Making with Multiple Objectives (MCDM'1984)*, volume 242 of *Lecture Notes in Economics and Mathematical Systems*, June 4–6, 1984, Cleveland, Ohio, USA. Springer-Verlag Berlin and Heidelberg GmbH & Co. KG. ISBN: 978-3-54015-223-1. Published December 31, 1985.

[877] Yacov Y. Haimes and Ralph E. Steuer, editors. *Proceedings of the 14th International Conference on Multiple Criteria Decision Making: Research and Practice in Multi*

Criteria Decision Making (MCDM'1998), volume 487 of *Lecture Notes in Economics and Mathematical Systems*, June 12–18, 1998, University of Virginia, Charlottesville, Virginia, USA. Springer. ISBN: 978-3-54067-266-1. See http://www.virginia.edu/~risk/mcdm98.html [accessed 2007-09-10]. Published June 15, 2000.

[878] Yacov Y. Haimes, Warren Hall, and Herbert T. Freedman. *Multiobjective Optimization in Water Resource Systems*, volume 3 of *Developments in Water Science*. Elsevier Scientific Publishing Co., New York, March 1975. ISBN: 0-4444-1313-8, 978-0-44441-313-0. ASIN: B000UUMGXE.

[879] B. Hajek. Cooling schedules for optimal annealing. *Mathematics of Operations Research*, (13), 1988.

[880] Paul R. Halmos. *Naive Set Theory*. Undergraduate Texts in Mathematics. Springer-Press New York Inc., first edition, June 2001. ISBN: 0-3879-0092-6. New Edition January 1998, ISBN: 978-0387900926.

[881] Ulrich Hammel and Thomas Bäck. Evolution strategies on noisy functions: How to improve convergence properties. In *Proceedings of the Third Conference on Parallel Problem Solving from Nature, PPSN III*, pages 159–168, 1994. In proceedings [492]. Online available at http://citeseer.ist.psu.edu/24308.html [accessed 2008-07-19].

[882] Richard W. Hamming. Error-detecting and error-correcting codes. *Bell System Technical Journal*, 29(2):147–169, 1950. Online available at http://guest.engelschall.com/~sb/hamming/ and http://garfield.library.upenn.edu/classics/classics_h.html [accessed 2007-08-13].

[883] Michelle Okaley Hammond and Terence Claus Fogarty. Co-operative oulipian generative literature using human based evolutionary computing. In *Late breaking paper at Genetic and Evolutionary Computation Conference (GECCO'2005)*, 2005. In proceedings [1764]. Distributed on CD-ROM at GECCO-2005. Online available at http://www.cs.bham.ac.uk/~wbl/biblio/gecco2005lbp/papers/56-hammond.pdf [accessed 2007-08-29].

[884] Lin Han and Xingshi He. A novel opposition-based particle swarm optimization for noisy problems. In *ICNC'07: Proceedings of the Third International Conference on Natural Computation*, volume 3, pages 624–629, 2007. doi:10.1109/ICNC.2007.119. In proceedings [997].

[885] David J. Hand, Heikki Mannila, and Padhraic Smyth. *Principles of Data Mining*. MIT Press, Cambridge, MA, August 2001. ISBN: 0-2620-8290-X

[886] Hisashi Handa, Dan Lin, Lee Chapman, and Xin Yao. Robust solution of salting route optimisation using evolutionary algorithms. In *Evolutionary Computation, proceedings of CEC 2006*, pages 3098–3105, 2006. doi:10.1109/CEC.2006.1688701. Online available at http://escholarship.lib.okayama-u.ac.jp/industrial_engineering/2/ [accessed 2008-06-19].

[887] James V. Hansen, Paul Benjamin Lowry, Rayman Meservy, and Dan McDonald. Genetic programming for prevention of cyberterrorism through previous dynamic and evolving intrusion detection. *Decision Support Systems*, 43(4):1362–1374, August 2007. doi:10.1016/j.dss.2006.04.004. Special Issue Clusters. Online available at http://dx.doi.org/10.1016/j.dss.2006.04.004 and http://ssrn.com/abstract=877981 [accessed 2008-06-17].

[888] Nikolaus Hansen and Andreas Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: the covariance matrix adaptation. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 312–317, 1996. In proceedings [1006]. Online available at http://citeseer.ist.psu.edu/hansen96adapting.html and http://www.bionik.tu-berlin.de/ftp-papers/CMAES.ps.Z [accessed 2007-09-20].

[889] Nikolaus Hansen and Andreas Ostermeier. Convergence properties of evolution strategies with the derandomized covariance matrix adaption: The $(\mu/\mu_i, \lambda)$-cma-es. In Hans-Jürgen Zimmermann, editor, *Eufit'97 – 5th European Congress on Intelligent*

Techniques and Soft Computing, pages 650–654. Verlag Mainz, Wissenschaftsverlag, 1997, Aachen. Online available at http://citeseer.ist.psu.edu/356709.html [accessed 2007-08-27].

[890] Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001. Online available at http://www.bionik.tu-berlin.de/user/niko/cmaartic.pdf and http://citeseer.ist.psu.edu/hansen01completely.html [accessed 2007-08-27].

[891] Nikolaus Hansen, Andreas Ostermeier, and Andreas Gawelczyk. On the adaptation of arbitrary normal mutation distributions in evolution strategies: The generating set adaptation. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 57–64, 1995. In proceedings [636]. Online available at http://citeseer.ist.psu.edu/261932.html [accessed 2007-08-27].

[892] P. Hansen. The steepest ascent mildest descent heuristic for combinatorial programming. In *Talk presented at the Congress on Numerical Methods in Combinatorial Optimization*. Academic Press, 1986, Capri, Italy.

[893] Pierre Hansen, editor. *Proceedings of the 5th International Conference on Multiple Criteria Decision Making: Essays and Surveys on Multiple Criteria Decision Making (MCDM'1982)*, volume 209 of *Lecture Notes in Economics and Mathematical Systems*, 1982, Mons, Belgium. Springer. ISBN: 978-0-38711-991-5. Published in March 1983.

[894] Jin-Kao Hao, Evelyne Lutton, Edmund M. A. Ronald, Marc Schoenauer, and Dominique Snyers, editors. *Selected Papers of the Third European Conference on Artificial Evolution, AE'97*, volume 1363 of *Lecture Notes in Computer Science (LNCS)*, October 22–24, 1997, Nîmes, France. Springer Berlin/Heidelberg. ISBN: 3-5406-4169-6, 978-3-54064-169-8. doi:10.1007/BFb0026588. Published in May 1998.

[895] Simon Harding and Julian Francis Miller. Evolution of robot controller using cartesian genetic programming. In *Genetic Programming, Proceedings of EuroGP 2005*, pages 62–73, 2005. doi:10.1007/b107383. In proceedings [1116]. Online available at http://www.cartesiangp.co.uk/papers/2005/hmeurogp2005.pdf [accessed 2009-06-26]2007-11-03.

[896] Georges Raif Harik. *Learning gene linkage to efficiently solve problems of bounded difficulty using genetic algorithms*. PhD thesis, University of Michigan, Ann Arbor, 1997. Doctoral Committee: Professor Keki and Professor David and E. Goldberg and Georges Raif Harik and Georges Raif Harik and Co-chairs Keki and B. Irani and B. Irani and David E. Goldberg. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.54.7092 [accessed 2008-10-17].

[897] Werner Harke. *Smart Home – Vernetzung von Haustechnik und Kommunikationssystemen im Wohnungsbau*. Hüthig Verlag / C. F. Müller, December 31, 2003. ISBN: 978-3-78807-713-6, 3-7880-7713-1.

[898] Lisa Lavoie Harlow, Stanley A. Mulaik, and James H. Steiger, editors. *What If There Were No Significance Tests?* Lawrence Erlbaum Associates, August 1997. ISBN: 0-8058-2634-3, 978-0-80582-634-0. Partly online available at http://books.google.de/books?id=lZ2ybZzrnroC [accessed 2008-08-15].

[899] Richard Harper, editor. *Inside the smart home*. Springer, 2003. ISBN: 1-8523-3688-9, 978-1-85233-688-2. Partly online available at http://books.google.com/books?id=SvcHvHuv86gC [accessed 2008-07-27].

[900] Kim Harries and Peter W. H. Smith. Exploring alternative operators and search strategies in genetic programming. In *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 147–155, 1997. In proceedings [1208]. Online available at http://citeseer.ist.psu.edu/harries97exploring.html and http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/harries.gp97_paper.ps.gz [accessed 2008-04-12].

[901] William E. Hart, Natalio Krasnogor, and J. E. Smith, editors. *Recent Advances in Memetic Algorithms*, volume 166/2005 of *Studies in Fuzziness and Soft Computing*. Springer, Berlin, Germany, 2005. ISBN: 3-5402-2904-3, 978-3-54022-904-9. doi:10.1007/3-540-32363-5. Partly online available at http://books.google.de/books?id=LYf7YW4DmkUC [accessed 2008-04-01].

[902] Brad Harvey, James A. Foster, and Deborah Frincke. Byte code genetic programming. In *Late Breaking Papers at the Genetic Programming 1998 Conference*, 1998. In proceedings [1198]. Online available at http://www.csds.uidaho.edu/deb/jvm.pdf and http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/harvey_1998_bcGP.html [accessed 2008-09-16].

[903] Brad Harvey, James Foster, and Deborah Frincke. Category: Genetic programming towards byte code genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, page 1234, 1999. In proceedings [142], Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.22.4974 and http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/harvey_1999_TBCGP.html [accessed 2008-09-16].

[904] Inman Harvey. The puzzle of the persistent question marks: A case study of genetic drift. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 15–22, 1993. In proceedings [730]. Also published as Technical Report Cognitive Science Research Paper CSRP 278, April 1993. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.56.3830 [accessed 2008-09-10].

[905] Randy L. Haupt and Sue Ellen Haupt. *Practical Genetic Algorithms*. Wiley-Interscience, December 19, 1997. ISBN: 0-4711-8873-5, 978-0-47118-873-5, 978-0-47145-565-3. Second edition: June 2004, partly online available at http://books.google.de/books?id=WO64iOg_abIC [accessed 2008-06-08].

[906] Barbara Hayes-Roth and Richard E. Korf, editors. *Proceedings of the 12th National Conference on Artificial Intelligence, AAAI*, July 31–August 4, 1994, Convention Center, Seattle, WA, USA. AAAI Press. ISBN: 0-2626-1102-3. 2 volumes, see http://www.aaai.org/Conferences/AAAI/aaai94.php [accessed 2007-09-06].

[907] Thomas D. Haynes, Roger L. Wainwright, and Sandip Sen. Evolving cooperation strategies. Technical Report UTULSA-MCS-94-10, The University of Tulsa, Tulsa, OK, USA, December 16, 1994. See also [908]. Online available at http://www.mcs.utulsa.edu/~rogerw/papers/Haynes-icmas95.pdf and http://citeseer.ist.psu.edu/1958.html [accessed 2007-10-03].

[908] Thomas D. Haynes, Roger L. Wainwright, and Sandip Sen. Evolving cooperating strategies. In Victor Lesser, editor, *Proceedings of the first International Conference on Multiple Agent Systems*, page 450. AAAI Press/MIT Press, June 12-14, 1995, San Francisco, USA. ISBN: 0-2626-2102-9. 1 page poster, see also [907].

[909] Thomas D. Haynes, Roger L. Wainwright, Sandip Sen, and Dale Schoenefeld. Strongly typed genetic programming in evolving cooperation strategies. In *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pages 271–278, 1995. In proceedings [636]. Online available at http://www.mcs.utulsa.edu/~rogerw/papers/Haynes-icga95.pdf and http://citeseer.ist.psu.edu/15037.html [accessed 2007-10-03].

[910] Thomas D. Haynes, Dale A. Schoenefeld, and Roger L. Wainwright. Type inheritance in strongly typed genetic programming. In *Advances in Genetic Programming 2*, chapter 18, pages 359–376. MIT Press, 1996. In collection [61]. Online available at http://citeseer.ist.psu.edu/haynes96type.html [accessed 2007-10-04].

[911] Richard Heady, George Luger, Arthur Maccabe, and Mark Servilla. The architecture of a network level intrusion detection system. Technical Report CS90-20, LA-SUB–93-219, W-7405-ENG-36, DE97002400, AHC29703%%44, Department of Computer Science, University of New Mexico, Albuquerque, NM, USA, August 15, 1990.

doi:10.2172/425295. Online available at http://www.osti.gov/energycitations/product.biblio.jsp?osti_id=425295 [accessed 2008-09-07].

[912] David Heath. *An introduction to experimental design and statistics for biology*. CRC Press, October 25, 1995. ISBN: 1-8572-8132-2, 978-1-85728-132-3. Catalogue no. TF2557. Partly online available at http://books.google.de/books?id=sJVHoJ2ML40C [accessed 2008-08-18].

[913] Thomas L. Heath. *The Thirteen Books of Euclid's Elements*. Dover Publications, New York, second edition, 1956. ISBN: 0-4866-0088-2, 0-4866-0089-0, 0-4866-0090-4. Three volumes. Original publication: Cambridge University Press, 1925. Euclid's original: [1559].

[914] Wendi Rabiner Heinzelman, Joanna Kulik, and Hari Balakrishnan. Adaptive protocols for information dissemination in wireless sensor networks. In *MobiCom'99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*, pages 174–185, 1999, Seattle, Washington, United States. ACM Press, New York, NY, USA. ISBN: 1-5811-3142-9. Online available at http://citeseer.ist.psu.edu/kulik99adaptive.html and http://www.comet.columbia.edu/~campbell/e6906/papers/heinzelman99.pdf [accessed 2007-08-13].

[915] Jochen Heistermann. *Genetische Algorithmen. Theorie und Praxis evolutionärer Optimierung*. Teubner Verlag, 1994. ISBN: 3-8154-2057-1, 978-3-81542-057-7.

[916] Jörg Heitkötter and David Beasley, editors. *Hitch-Hiker's Guide to Evolutionary Computation: A List of Frequently Asked Questions (FAQ)*. ENCORE (The EvolutioNary Computation REpository Network), 1998. USENET: comp.ai.genetic. Online available at http://www.cse.dmu.ac.uk/~rij/gafaq/top.htm and http://alife.santafe.edu/~joke/encore/www/ [accessed 2007-07-03].

[917] Jim Hendler and Devika Subramanian, editors. *Proceedings of the Sixteenth National Conference on Artificial Intelligence (AAAI) and Eleventh Conference on Innovative Applications of Artificial Intelligence (IAAI)*, July 18–22, 1999, Orlando, Florida, USA. AAAI Press/The MIT Press. ISBN: 0-2625-1106-1. See http://www.aaai.org/Conferences/AAAI/aaai99.php [accessed 2007-09-06] and http://www.aaai.org/Conferences/IAAI/iaai99.php [accessed 2007-09-06].

[918] Michael Hennecke. Ranexp: experimental random number generator package. *Computer Physics Communications*, 79(2):261–267, April 1994. doi:10.1016/0010-4655(94)90072-8. Online available at http://dx.doi.org/10.1016/0010-4655(94)90072-8 [accessed 2007-08-19].

[919] Alain Hertz, Éric D. Taillard, and Dominique de Werra. A tutorial on tabu search. In *Proceedings of Giornate di Lavoro AIRO'95 (Enterprise Systems: Management of Technological and Organizational Changes)*, pages 13–24, 1995, Italy. Online available at http://citeseer.ist.psu.edu/73006.html [accessed 2007-09-15].

[920] Nabil M. Hewahi. Credit apportionment scheme for rule based systems. *Journal of the Islamic University of Gaza*, 9(1):25–41, 2001.

[921] Nabil M. Hewahi and H. Ahmad. Credit apportionment scheme for rule-based systems: Implementation and comparative study. In *Developments in Applied Artificial Intelligence, Proceedings of the 15th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems IEA/AIE*, volume 2358/2002 of *Lecture Notes in Computer Science (LNCS)*, pages 43–56, June 17–20, 2002, Cairns, Australia. Springer, Berlin/Heidelberg. ISBN: 978-3-54043-781-9. doi:10.1007/3-540-48035-8_43.

[922] Nabil M. Hewahi and K. K. Bharadwaj. Bucket brigade algorithm for hierarchical censored production rule-based system. *International Journal of Intelligent Systems*, 11(4):197–225, 1996.

[923] Malcom I. Heywood and A. Nur Zincir-Heywood. Dynamic page-based linear genetic programming. *IEEE Transactions on Systems, Man and Cybernetics – Part B: Cybernetics*, 32(3):380–388, 2002. ISSN: 1083-4419.

[924] Joseph F. Hicklin. Application of the genetic algorithm to automatic program generation. Master's thesis, Department of Computer Science, University of Idaho, Moscow, ID, USA, 1986.

[925] Takahide Higuchi, Shigeyoshi Tsutsui, and Masayuki Yamamura. Theoretical analysis of simplex crossover for real-coded genetic algorithms. In *PPSN VI: Proceedings of the 6th International Conference on Parallel Problem Solving from Nature*, pages 365–374, 2000. doi:10.1007/3-540-45356-3_36. In proceedings [1830].

[926] David Hilbert and Wilhelm Friedrich Ackermann. *Grundzüge der theoretischen Logik*, volume XXVII of *Die Grundlehren der mathematischen Wissenschaften in Einzeldarstellungen mit besonderer Berücksichtigung der Anwendungsgebiete*. W. Blaschke, R. Rammel, E. Hopf, et al., Julius Springer, Berlin, Germany, August 1928. ISBN: 3-5400-5843-5. English version in [927].

[927] David Hilbert and Wilhelm Friedrich Ackermann. *Principles of Mathematical Logic*. American Mathematical Society (AMS) Chelsea Publishing, 1950. ISBN: 0-8218-2024-9, 978-0-82182-024-7. Edited by Robert E. Luce, translated by Lewis M. Hammond, George G. Leckie, and F. Steinhardt. Partly online available at http://books.google.com/books?id=7E34DeXok9gC [accessed 2008-03-25]. Original German version: [926].

[928] W. Daniel Hillis. Co-evolving parasites improve simulated evolution as an optimization procedure. *Physica D: Nonlinear Phenomena*, 42(1-3):228–234, June 1990. doi:10.1016/0167-2789(90)90076-2. Provided by the SAO/NASA Astrophysics Data System.

[929] Geoffrey E. Hinton and Steven J. Nowlan. How learning can guide evolution. *Complex Systems*, 1:495–502, 1987. See also [930]. Online available at http://htpprints.yorku.ca/archive/00000172/ [accessed 2008-09-10].

[930] Geoffrey E. Hinton and Steven J. Nowlan. How learning can guide evolution. In *Adaptive individuals in evolving populations: models and algorithms*, pages 447–454. Addison-Wesley Longman Publishing Co., Inc., 1996. In collection [171]. See also [929].

[931] Kotaro Hirasawa, M. Okubo, Jinglu Hu, and Junichi Murata. Comparison between genetic network programming (GNP) and genetic programming (GP). In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, pages 1276–1282, 2001. In proceedings [1003].

[932] Haym Hirsh and Steve A. Chien, editors. *Proceedings of the Thirteenth Innovative Applications of Artificial Intelligence Conference*, August 7–9, 2001, Seattle, Washington, USA. AAAI. ISBN: 1-5773-5134-7. See http://www.aaai.org/Conferences/IAAI/iaai01.php [accessed 2007-09-06].

[933] Sir Charles Antony Richard (Tony) Hoare. Quicksort. *Computer Journal*, 5(1):10–15, 1962.

[934] Frank Hoffmeister and Thomas Bäck. Genetic algorithms and evolution strategies – similarities and differences. In *PPSN I: Proceedings of the 1st Workshop on Parallel Problem Solving from Nature*, pages 455–469, 1990. Published 1991. In proceedings [1842] and also [935].

[935] Frank Hoffmeister and Thomas Bäck. Genetic algorithms and evolution strategies – similarities and differences. Technical Report SYS-1/92, University of Dortmund - Systems Analysis, 1992. See also [934].

[936] Frank Hoffmeister and Hans-Paul Schwefel. Korr 2.1 – an implementation of a $(\mu \dagger \lambda)$-evolution strategy. Technical Report, Universität Dortmund, Fachbereich Informatik, November 1990. Interner Bericht.

[937] John Henry Holland. Outline for a logical theory of adaptive systems. *Journal of the ACM*, 9(3):297–314, 1962. ISSN: 0004-5411. doi:10.1145/321127.321128. Online available at http://portal.acm.org/citation.cfm?id=321128 [accessed 2007-07-28].

[938] John Henry Holland. *Nonlinear environments permitting efficient adaptation*, volume II of *Computer and Information Sciences*. Academic Press, New York, USA, 1967. Series editor: J. T. Tou.

[939] John Henry Holland. Adaptive plans optimal for payoff-only environments. In *Proceedings of the Second Hawaii International Conference on System Sciences, Periodicals*, pages 917–920. North Holland, January 22–24, 1969, University of Hawaii, Honolulu, USA. DTIC Accession Number: AD0688839.

[940] John Henry Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The University of Michigan Press, Ann Arbor, 1975. ISBN: 0-4720-8460-7, 978-0-47208-460-9, 0-5850-3844-9, 978-0-58503-844-5, 978-0-26258-111-6. Reprinted by MIT Press, April 1992, NetLibrary, Inc.

[941] John Henry Holland. Adaptive algorithms for discovering and using general patterns in growing knowledge bases. *International Journal of Policy Analysis and Information Systems*, 4(3):245–268, 1980.

[942] John Henry Holland. Properties of the bucket brigade algorithm. In *Proceedings of the International Conference on Genetic Algorithms and Their Applications*, pages 1–7, 1985. In proceedings [856].

[943] John Henry Holland. Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach: Volume II*, pages 593–623. Kaufmann, Los Altos, CA, 1986. See also [944].

[944] John Henry Holland. Escaping brittleness: the possibilities of general-purpose learning algorithms applied to parallel rule-based systems. *Computation & Intelligence: Collected Readings*, pages 275–304, 1995. ISBN: 0-2626-2101-0. See also [943].

[945] John Henry Holland. Genetic algorithms. *Scientific American*, 267(1):44–50, July 1992. Online available at http://www.econ.iastate.edu/tesfatsi/holland.GAIntro.htm and http://www.cc.gatech.edu/~turk/bio_sim/articles/genetic_algorithm.pdf [accessed 2007-08-29].

[946] John Henry Holland and Arthur W. Burks. *Adaptive computing system capable of learning and discovery*. Number 619349 filed on 1984-06-11 in United States Patent. United States Patent and Trademark Office, 1987. US Patent Issued on September 29, 1987, Current US Class 706/13, Genetic algorithm and genetic programming system 382/155, LEARNING SYSTEMS 706/62 MISCELLANEOUS, Foreign Patent References 8501601 WO Apr., 1985. Online available at http://www.patentstorm.us/patents/4697242.html and http://www.freepatentsonline.com/4697242.html [accessed 2008-04-01]. Editors: Jerry Smith and John R Lastova.

[947] John Henry Holland and Judith S. Reitman. Cognitive systems based on adaptive algorithms. *ACM SIGART Bulletin*, 63:49, June 1977. ISSN: 0163-5719. Online available at http://doi.acm.org/10.1145/1045343.1045373 [accessed 2007-08-18]. See also [948].

[948] John Henry Holland and Judith S. Reitman. Cognitive systems based on adaptive algorithms. In D. A. Waterman and F. Hayes-Roth, editors, *Pattern directed inference systems*, pages 313–329. Academic Press, New York, NY, 1978. Reprinted in [696]. See also [947].

[949] John Henry Holland, Keith J. Holyoak, and Richard E. Nisbett abd Paul R. Thagard. *Induction: Processes of Inference, Learning, and Discovery*. Computational Models of Cognition & Perception. The MIT Press, first, 1989 reprint edition, March 1986. ISBN: 0-2625-8096-9, 978-0-26258-096-0. Partly online available at http://books.google.de/books?id=Z6EFBaLApE8C [accessed 2008-04-03].

[950] John Henry Holland, Lashon Bernard Booker, Marco Colombetti, Marco Dorigo, David E. Goldberg, Stephanie Forrest, Rick L. Riolo, Robert Elliott Smith, Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson. What is a learning classifier

system? In *Learning Classifier Systems, From Foundations to Applications*, pages 3–32. Springer-Verlag Berlin/Heidelberg, 2000. In collection [1252].

[951] John H. Holmes. Discovering risk of disease with a learning classifier system. In *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA97)*, 1997. In proceedings [98]. Online available at http://citeseer.ist.psu.edu/holmes98discovering.html and http://www.cs.bris.ac.uk/~kovacs/lcs.archive/Holmes1997a.ps.gz [accessed 2007-09-12].

[952] Diana Holstein and Pablo Moscato. Memetic algorithms using guided local search: a case study. In *New ideas in optimization*, pages 235–244. McGraw-Hill Ltd., 1999. ISBN: 0-0770-9506-5. In collection [448].

[953] Robert C. Holte. Combinatorial auctions, knapsack problems, and hill-climbing search. In *Proceedings of the 14th Biennial Conference of the Canadian Society on Computational Studies of Intelligence: Advances in Artificial Intelligence*, volume 2056/2001 of *Lecture Notes in Computer Science (LNCS)*, page 57. Springer Berlin / Heidelberg, June 7–9, 2001, Ottawa, Canada. ISBN: 3-5404-2144-0. Online available at http://citeseer.ist.psu.edu/holte01combinatorial.html [accessed 2007-09-11].

[954] Robert C. Holte and Adele Howe, editors. *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, AAAI*, July 22–26, 2007, Vancouver, British Columbia, Canada. AAAI Press. ISBN: 978-1-57735-323-2. See http://www.aaai.org/Conferences/AAAI/aaai07.php [accessed 2007-09-06].

[955] Gerald J. Holzmann. *The Spin Model Checker – Primer and Reference Manual*. Addison-Wesley Professional/Longman, Amsterdam, The Netherlands, September 4, 2003. ISBN: 0-3212-2862-6, 978-0-32122-862-8. Partly online available at http://books.google.de/books?id=NpBdZKmOHO8C [accessed 2008-10-02].

[956] Jin-Hyuk Hong and Sung-Bae Cho. The classification of cancer based on dna microarray data that uses diverse ensemble genetic programming. *Artificial Intelligence in Medicine*, 36(1):43–58, 2006. Online available at http://sclab.yonsei.ac.kr/~hjinh/PAPER/IJ2006-1.pdf and http://dx.doi.org/10.1016/j.artmed.2005.06.002 [accessed 2007-09-09].

[957] Bryan Horling, Roger Mailler, Mark Sims, and Victor Lesser. Using and maintaining organization in a large-scale distributed sensor network. *Proceedings of the Workshop on Autonomy, Delegation, and Control (AAMAS03)*, July 2003. Online available at ftp://mas.cs.umass.edu/pub/bhorling/03-organization.ps.gz and http://mas.cs.umass.edu/paper/247 [accessed 2007-08-01].

[958] Jeffrey Horn, David E. Goldberg, and Kalyanmoy Deb. Long path problems. In *PPSN III: Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature*, pages 149–158, 1994. In proceedings [492]. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.51.9758 [accessed 2008-08-12].

[959] Jeffrey Horn, Nicholas Nafpliotis, and David E. Goldberg. A niched pareto genetic algorithm for multiobjective optimization. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, volume 1, pages 82–87, 1994. doi:10.1109/ICEC.1994.350037. In proceedings [1411]. Also: IEEE Symposium on Circuits and Systems, pp. 2264–2267, 1991, Online available at http://citeseer.ist.psu.edu/horn94niched.html and http://www.lania.mx/~ccoello/EMOO/horn2.ps.gz [accessed 2007-08-28].

[960] Helmut Hörner. A c++ class library for gp: Vienna university of economics genetic programming kernel (release 1.0, operating instructions). Technical Report, Vienna University of Economics, May 29, 1996. Online available at http://citeseer.ist.psu.edu/253491.html [accessed 2007-09-09].

[961] Nicola Howarth. Abstract syntax tree design. Technical Report 23/08/95 APM.1551.01, Architecture Projects Management Limited, Poseidon House, Castle

Park, Cambridge CB3 0RD, UK, August 23, 1995. Online available at http://www.ansa.co.uk/ANSATech/95/Primary/155101.pdf [accessed 2007-07-03].

[962] Robert J. Howlett and Lakhmi C. Jain, editors. *Proceedings of the Fourth International Conference on Knowledge-Based Intelligent Information Engineering Systems & Allied Technologies, KES 2000, Volume 1*, August 30–September 1, 2000, Brighton, UK. IEEE. ISBN: 0-7803-6400-7. See also [963].

[963] Robert J. Howlett and Lakhmi C. Jain, editors. *Proceedings of the Fourth International Conference on Knowledge-Based Intelligent Information Engineering Systems & Allied Technologies, KES 2000, Volume 2*, August 30–September 1, 2000, Brighton, UK. IEEE. ISBN: 0-7803-6400-7. See also [962].

[964] J. Hromkovic and I. Zámecniková. *Design and Analysis of Randomized Algorithms: Introduction to Design Paradigms*. Texts in Theoretical Computer Science, an EATCS series. Springer, first edition, July 2005. ISBN: 978-3-54023-949-9.

[965] Juraj Hromkovic. *Algorithmics for Hard Computing Problems*. Springer, first edition, June 2001. ISBN: 978-3-54066-860-2.

[966] Ian Hsieh, Kiat-Choong Chen, and Cao An Wang. A genetic algorithm for the minimum tetrahedralization of a convex polyhedron. In *CCCG Proceedings of the 15th Canadian Conference on Computational Geometry, CCCG'03*, pages 115–119, August 2003, Halifax, Canada.

[967] Ting Hu and Wolfgang Banzhaf. Measuring rate of evolution in genetic programming using amino acid to synonymous substitution ratio ka/ks. In *Genetic and Evolutionary Computation Conference*, pages 1337–1338, 2008. In proceedings [1117]. Online available at http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/Hu_2008_gecco.html and http://www.cs.mun.ca/~tingh/papers/GECCO08.pdf [accessed 2008-08-08].

[968] Ting Hu and Wolfgang Banzhaf. Nonsynonymous to synonymous substitution ratio ka/ks: Measurement for rate of evolution in evolutionary computation. In *Proceedings of 10th International Conference on Parallel Problem Solving from Nature*, 2008. In proceedings [1948].

[969] D. Huang. A framework for the credit-apportionment process in rule-based systems. *IEEE Transactions on Systems, Man and Cybernetics*, 19(3):489–498, May/June 1989. doi:10.1109/21.31056.

[970] De-Shuang Huang, Laurent Heutte, and Marco Loog, editors. *Advanced Intelligent Computing Theories and Applications. With Aspects of Contemporary Intelligent Computing Techniques – Proceedings of the Third International Conference on Intelligent Computing, ICIC 2007*, volume 2 of *Communications in Computer and Information Science*, August 21–24, 2007, Qingdao, China. Springer Berlin/Heidelberg. ISBN: 978-3-54074-281-4, 978-3-54074-282-1. doi:10.1007/978-3-540-74282-1. Online available at http://www.springerlink.com/content/k74022/ [accessed 2008-03-28].

[971] Hsien-Da Huang, Jih Tsung Yang, Shu Fong Shen, and Jorng-Tzong Horng. An evolution strategy to solve sports scheduling problems. In *Proceedings of the Genetic and Evolutionary Computation Conference GECCO-99*, volume 1, page 943, 1999. In proceedings [142].

[972] Simon Huband, Philip Hingston, Luigi Barone, and R. Lyndon While. A review of multiobjective test problems and a scalable test problem toolkit. *IEEE Transactions on Evolutionary Computation*, 10(5):477–506, October 2006. ISSN: 1089-778X. INSPEC Accession Number: 9101736. doi:10.1109/TEVC.2005.861417.

[973] Richard R. Hudson, Martin Kreitman, and Montserrat Aguade. A test of neutral molecular evolution based on nucleotide data. *Genetics*, 116(1):153–159, May 1987. Online available at http://www.genetics.org/cgi/reprint/116/1/153.pdf [accessed 2007-08-05].

[974] Barry D. Hughes. *Random Walks and Random Environments: Volume 1: Random Walks*. Oxford University Press, USA, May 16, 1995. ISBN: 0-1985-3788-3, 978-0-19853-788-5.

[975] Phil Husbands and Inman Harvey, editors. *Advances in Artificial Life, Proceedings of the Fourth European Conference, ECAL'97*, Bradford Books, Complex Adaptive Systems, July 1997. The MIT Press. ISBN: 0-2625-8157-4, 978-0-26258-157-8.

[976] Phil Husbands and Jean-Arcady Meyer, editors. *Proceedings of the First European Workshop on Evolutionary Robotics, EvoRbot98*, volume 1468/1998 of *Lecture Notes in Computer Science (LNCS)*, April 16–17, 1998, Paris, France. Springer. ISBN: 3-5406-4957-3.

[977] Tim J. Hutton. Evolvable self-replicating molecules in an artificial chemistry. *Artificial Life*, 8(4):341–356, Winter 2002. ISSN: 1064-5462. Online available at http://www.sq3.org.uk/papers/evselfreps.pdf [accessed 2008-05-01].

[978] Tim J. Hutton. Simulating evolutions first steps. In *Advances in Artificial Life, Proceedings of the 7th European Conference*, 2003. In proceedings [143]. Online available at http://www.sq3.org.uk/papers/ecal03.pdf [accessed 2008-05-01].

[979] Tim J. Hutton. Evolvable self-reproducing cells in a two-dimensional artificial chemistry. *Artificial Life*, 13(1):11–30, Spring 2007. ISSN: 1064-5462. Online available at http://www.sq3.org.uk/papers/cells2007.pdf [accessed 2008-05-01].

[980] Marijn A. Huynen, Peter F. Stadler, and Walter Fontana. Smoothness within ruggedness: The role of neutrality in adaptation. *Proceedings of the National Academy of Science, USA*, 93:397–401, January 1996. Evolution. Online available at http://www.pnas.org/cgi/reprint/93/1/397.pdf and http://citeseer.ist.psu.edu/91710.html [accessed 2008-02-27]. Communicated by Hans Frauenfelder, Los Alamos National Laboratory, Los Alamos, NM, September 20, 1995 (received for review June 29, 1995).

[981] Martijn A. Huynen. Exploring phenotype space through neutral evolution. *Journal of Molecular Evolution*, 43(3):165–169, September 1996. ISSN: 0022-2844 (Print) 1432-1432 (Online). doi:10.1007/PL00006074. Online available at http://citeseer.ist.psu.edu/32668.html and http://www.santafe.edu/research/publications/wpabstract/199510100 [accessed 2008-02-27].

[982] Chii-Ruey Hwang. Simulated annealing: Theory and applications. *Acta Applicandae Mathematicae: An International Survey Journal on Applying Mathematics and Mathematical Applications*, 12(1):108–111, May 1988. ISSN: 0167-8019 (Print) 1572-9036 (Online). doi:10.1007/BF00047572. Academia Sinica. This is a review of [2095]. Online available at http://www.springerlink.com/content/t1n71m85j33556m4/ [accessed 2007-08-25].

[983] Gwo-Jen Hwang, Peng-Yeng Yin, and Shu-Heng Yeh. A tabu search approach to generating test sheets for multiple assessment criteria. *IEEE Transactions on Education*, 49(1):88–97, February 2006. doi:10.1109/TE.2002.858405.

I

[984] Hitoshi Iba. Emergent cooperation for multiple agents using genetic programming. In *PPSN IV: Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*, pages 32–41, 1996. doi:10.1007/3-540-61723-X_967. In proceedings [2118].

[985] Hitoshi Iba. Evolutionary learning of communicating agents. *Information Sciences – Informatics and Computer Science: An International Journal*, 108(1–4):181–205, July 1998. ISSN: 0020-0255. doi:10.1016/S0020-0255(97)10055-X. Online available at http://dx.doi.org/10.1016/S0020-0255(97)10055-X [accessed 2008-07-28].

[986] Hitoshi Iba. Evolving multiple agents by genetic programming. In *Advances in genetic programming 3*, pages 447–466. MIT Press, Cambridge, MA, USA, 1999. In collection [1936]. Online available at http://www.cs.bham.ac.uk/~wbl/aigp3/ch19.pdf [accessed 2007-10-03].

[987] Hitoshi Iba, Tishihide Nozoe, and Kanji Ueda. Evolving communicating agents based on genetic programming. In *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation*, pages 297–302, 1997. In proceedings [106]. Online available at http://lis.epfl.ch/~markus/References/Iba97.pdf [accessed 2008-07-28].

[988] Toshihide Ibaraki, Koji Nonobe, and Mutsunori Yagiura, editors. *Fifth Metaheuristics International Conference (MIC2003) – Metaheuristics: Progress as Real Problem Solvers*, volume 32 of *Operations Research/Computer Science Interfaces*, August 25–28, 2003, Kyoto International Conference Hall, Kyoto, Japan. Springer, Berlin Heidelberg New York. ISBN: 0-3872-5382-3. Published in 2005. See http://www-or.amp.i.kyoto-u.ac.jp/mic2003/ [accessed 2007-09-12].

[989] *BPEL4WS, Business Process Execution Language for Web Services Version 1.1*. IBM, BEA Systems, Microsoft, SAP AG, Siebel Systems, May 2003. Online available at http://www.ibm.com/developerworks/library/specification/ws-bpel/ [accessed 2007-09-03].

[990] *Proceedings of the Second IEE/IEEE International GALESIA Conference*, September 2–4, 1997, University of Strathclyde, Glasgow, UK. IEE. ISBN: 0-8529-6693-8.

[991] *4th International Conference on Hybrid Intelligent Systems (HIS 2004)*, December 5–8, 2005, Kitakyushu, Japan. IEEE Computer Society. ISBN: 0-7695-2291-2. INSPEC Accession Number: 8470883.

[992] *Proceedings of 2006 IEEE Joint Conference on E-Commerce Technology (CEC'06) and Enterprise Computing, E-Commerce and E-Services (EEE'06)*, June 26–29, 2006, The Westin San Francisco Airport, 1 Old Bayshore Highway, Millbrae, United States. IEEE Computer Society, Los Alamitos, California, Washington, Tokyo. ISBN: 978-0-76952-511-2.

[993] *6th International Conference on Hybrid Intelligent Systems (HIS 2006)*, December 13–15, 2006, AUT Technology Park, Auckland, New Zealand. IEEE Computer Society. ISBN: 0-7695-2662-4. see http://his06.hybridsystem.com/ [accessed 2007-09-01].

[994] *Proceedings of IEEE Joint Conference (CEC/EEE 2007) on E-Commerce Technology (9th CEC'07) and Enterprise Computing, E-Commerce and E-Services (4th EEE'07)*, July 23–26, 2007, National Center of Sciences, Tokyo, Japan. IEEE Computer Society, IEEE Computer Society. ISBN: 978-0-76952-913-4.

[995] *Proceedings of the Third International Conference on Advances in Natural Computation, ICNC 2007*, volume 1, August 24–27, 2007, Haikou, China. IEEE Computer Society, Washington, DC, USA. ISBN: 0-7695-2875-9, 978-0-76952-875-5. doi:10.1109/ICNC.2007.763. See also [996, 997, 998, 999].

[996] *Proceedings of the Third International Conference on Advances in Natural Computation, ICNC 2007*, volume 2, August 24–27, 2007, Haikou, China. IEEE Computer Society, Washington, DC, USA. ISBN: 0-7695-2875-9, 978-0-76952-875-5. doi:10.1109/ICNC.2007.763. See also [995, 997, 998, 999].

[997] *Proceedings of the Third International Conference on Advances in Natural Computation, ICNC 2007*, volume 3, August 24–27, 2007, Haikou, China. IEEE Computer Society, Washington, DC, USA. ISBN: 0-7695-2875-9, 978-0-76952-875-5. doi:10.1109/ICNC.2007.766. See also [995, 996, 998, 999].

[998] *Proceedings of the Third International Conference on Advances in Natural Computation, ICNC 2007*, volume 4, August 24–27, 2007, Haikou, China. IEEE Computer Society, Washington, DC, USA. ISBN: 0-7695-2875-9, 978-0-76952-875-5. doi:10.1109/ICNC.2007.765. See also [995, 996, 997, 999].

[999] *Proceedings of the Third International Conference on Advances in Natural Computation, ICNC 2007*, volume 5, August 24–27, 2007, Haikou, China. IEEE Computer Society, Washington, DC, USA. ISBN: 0-7695-2875-9, 978-0-76952-875-5. doi:10.1109/ICNC.2007.762. See also [995, 996, 997, 998].

[1000] *Second IEEE International Conference on Evolutionary Computation, ICEC'95*, volume 1-2, November 29–December 1, 1995, Perth, Australia. IEEE Press, Piscataway, New Jersey, USA. ISBN: 0-7803-2759-4. INSPEC Accession Number: 5250198. See http://ieeexplore.ieee.org/servlet/opac?punumber=3507 [accessed 2007-09-06]. CEC-95 Editors not given by IEEE, Organisers David Fogel and Chris deSilva.

[1001] *The 1998 IEEE International Conference on Evolutionary Computation, Proceedings of IEEE World Congress on Computational Intelligence, CEC98*, May 4–9, 1998, Anchorage, Alaska, USA. IEEE Press. Part of WCCI, see http://ieeexplore.ieee.org/servlet/opac?punumber=5621 [accessed 2007-09-06].

[1002] *Proceedings of the IEEE Congress on Evolutionary Computation, CEC00*, July 6–9, 2000, La Jolla Marriott Hotel, La Jolla, California, USA. IEEE Press, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA. ISBN: 0-7803-6375-2. Library of Congress Control Number: 00-018644. doi:10.1109/ICASSP.2000.862060. CEC-2000 – A joint meeting of the IEEE, Evolutionary Programming Society, Galesia, and the IEE. IEEE Catalog Number: 00TH8512.

[1003] *Proceedings of the IEEE Congress on Evolutionary Computation, CEC2001*, May 27–30, 2001, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea. IEEE Press, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA. ISBN: 0-7803-6658-1. CEC-2001 – A joint meeting of the IEEE, Evolutionary Programming Society, Galesia, and the IEE. IEEE Catalog Number: 01TH8546C.

[1004] *Proceedings of the IEEE Congress on Evolutionary Computation, CEC2004*, June 20–23, 2004, Portland, Oregon. IEEE Press, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA. ISBN: 0-7803-8515-2. CEC 2004 – A joint meeting of the IEEE, the EPS, and the IEE.

[1005] *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2007*, September25–28, 2007, Swissôtel The Stamford, Singapore. IEEE Press, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA. See http://cec2007.nus.edu.sg/ [accessed 2007-08-06].

[1006] *Proceedings of IEEE International Conference on Evolutionary Computation, CEC96*, May 20–22, 1996, Nagoya, Japan. IEEE Press, Piscataway, NJ. See http://ieeexplore.ieee.org/servlet/opac?punumber=3838 [accessed 2007-09-06].

[1007] Christian Igel. Causality of hierarchical variable length representations. In *Proceedings of the 1998 IEEE World Congress on Computational Intelligence*, pages 324–329, 1998. In proceedings [1001]. Online available at http://www.neuroinformatik.ruhr-uni-bochum.de/PEOPLE/igel/CoHVLR.ps.gz and http://citeseer.ist.psu.edu/61421.html [accessed 2007-08-12].

[1008] Christian Igel and Marc Toussaint. On classes of functions for which no free lunch results hold. *Information Processing Letters*, 86(6):317–321, June 30, 2003. ISSN: 0020-0190. doi:10.1016/S0020-0190(03)00222-9. Online available at http://dx.doi.org/10.1016/S0020-0190(03)00222-9 and http://citeseer.ist.psu.edu/623026.html [accessed 2008-03-28].

[1009] Christian Igel and Marc Toussaint. Recent results on no-free-lunch theorems for optimization, March 31, 2003. ArXiv EPrint arXiv:cs/0303032 (Computer Science, Neural and Evolutionary Computing) Online available at http://www.citebase.org/abstract?id=oai:arXiv.org:cs/0303032 [accessed 2008-03-28].

[1010] John F. McLoughlin III and Walter Cedeño. The enhanced evolutionary tabu search and its application to the quadratic assignment problem. In *GECCO'05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 975–982, 2005. doi:10.1145/1068009.1068175. In proceedings [202]. Online available at http://doi.acm.org/10.1145/1068009.1068175 [accessed 2007-08-25].

[1011] Michael Iles and Dwight Lorne Deugo. A search for routing strategies in a peer-to-peer network using genetic programming. In *SRDS'02: Proceedings of the 21st IEEE Symposium on Reliable Distributed Systems (SRDS'02)*, pages

341–346, October 13–16, 2002, Osaka, Japan. IEEE Computer Society, Washington, DC, USA. ISBN: 0-7695-1659-9. INSPEC Accession Number: 7516795. doi:10.1109/RELDIS.2002.1180207.

[1012] Mohammad Ilyas and Imad Mahgoub, editors. *Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems*. CRC, July 16, 2004. ISBN: 978-0-84931-968-6.

[1013] Information Sciences Institute, University of Southern California, 4676 Admiralty Way, Marina del Rey, California 90291, USA. Internet protocol, darpa internet program protocol specification. RFC 791, Defense Advanced Research Projects Agency, Information Processing Techniques Office, 1400 Wilson Boulevard, Arlington, Virginia 22209, USA, September 1981. Online available at http://tools.ietf.org/html/rfc791 [accessed 2008-06-13].

[1014] Lester Ingber. Adaptive simulated annealing (asa): Lessons learned. *Control and Cybernetics*, 25(1):33–54, 1996. Online available at http://www.ingber.com/asa96_lessons.pdf and http://citeseer.ist.psu.edu/592447.html [accessed 2008-03-27].

[1015] Lester Ingber. A simple options training model. *Mathematical Computer Modelling*, 30(5–6):167–182, 1999. Online available at http://www.ingber.com/markets99_spread.pdf and http://citeseer.ist.psu.edu/ingber99simple.html [accessed 2007-08-25].

[1016] Lester Ingber. Trading markets with canonical momenta and adaptive simulated annealing. Technical Report 1996:TMCMASA, Lester Ingber Research, McLean, VA, 1996. Online available at http://www.ingber.com/markets96_brief.pdf and http://www.geocities.com/francorbusetti/ingbergermarkets.pdf [accessed 2007-08-25].

[1017] Lester Ingber. Simulated annealing: Practice versus theory. *Mathematical and Computer Modelling*, 18(11):29–57, 1993. Online available at http://citeseer.ist.psu.edu/589471.html [accessed 2007-09-15]. See also [1018].

[1018] Lester Ingber. Simulated annealing: Practice versus theory. Technical Report, Lester Ingber Research, P.O.B. 857, McLean, VA 22101, 1993. Online available at http://ideas.repec.org/p/lei/ingber/93sa.html and http://www.ingber.com/asa93_sapvt.pdf [accessed 2007-08-18]. See also [1017].

[1019] *Proceedings of BIONETICS 2007, 2nd International Conference on Bio-Inspired Models of Network, Information, and Computing Systems*, December 10–13, 2007, Radisson SAS Beke Hotel, 43. Terez krt., Budapest H-1067, Hungary. Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering (ICST), IEEE, ACM. ISBN: 978-9-63979-905-9, 978-9-63979-911-0.

[1020] *Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03*, IEEE International Symposia on Swarm Intelligence, April 24–26, 2003, University Place Conference Center, Indiana University Purdue University Indianapolis, Indianapolis, Indiana, USA. Institute of Electrical & Electronics Enginee (IEEE). ISBN: 978-0-78037-914-5. doi:10.1109/SIS.2003.1202237. Cat. No.03EX706, http://www.computelligence.org/sis/2003/ [accessed 2007-08-26].

[1021] *Proceedings of the 2005 IEEE Swarm Intelligence Symposium. SIS2005*, IEEE International Symposia on Swarm Intelligence, June 8–10, 2005, The Westin Pasadena, Pasadena, California, USA. Institute of Electrical & Electronics Enginee (IEEE). ISBN: 978-0-78038-916-8.

[1022] *Proceedings of the 2006 IEEE Swarm Intelligence Symposium. SIS'06*, IEEE International Symposia on Swarm Intelligence, May 12–14, 2006, Hilton Indianapolis Hotel, Indianapolis, Indiana, USA. Institute of Electrical & Electronics Enginee (IEEE). http://www.computelligence.org/sis/2006/ [accessed 2007-08-26].

[1023] Hisao Ishibuchi, Tadashi Yoshida, and Tadahiko Murata. Balance between genetic search and local search in hybrid evolutionary multi-criterion optimization

algorithms. In *GECCO'02: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1301–1308, 2002. In proceedings [1245]. Online available at http://citeseer.ist.psu.edu/632937.html [accessed 2007-09-10].

[1024] Takuya Ito, Hitoshi Iba, and Satoshi Sato. Non-destructive depth-dependent crossover for genetic programming. In *Genetic Programming*, pages 71–82, 1998. doi:10.1007/BFb0055929. In proceedings [141].

[1025] S. Sitharama Iyengar and Richard R. Brooks, editors. *Distributed Sensor Networks*. Chapman & Hall/CRC, December 29, 2004. ISBN: 978-1-58488-383-8.

J

[1026] Jabir and J. W. Moore. A search for fundamental principles of software engineering. *Computer Standards & Interfaces*, 19(2):155–160, March 1998. doi:10.1016/S0920-5489(98)00009-9. Online available at http://dx.doi.org/10.1016/S0920-5489(98)00009-9 and http://www.gelog.etsmtl.ca/publications/pdf/249.pdf [accessed 2007-09-02].

[1027] David Jackson. Parsing and translation of expressions by genetic programming. In *GECCO'05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1681–1688, 2005. doi:10.1145/1068009.1068291. In proceedings [202]. Online available at http://doi.acm.org/10.1145/1068009.1068291 [accessed 2008-05-04].

[1028] Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall Advanced Reference Series. Prentice-Hall, Upper Saddle River, NJ, USA, 1988. ISBN: 0-1302-2278-X.

[1029] Anil K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, September 1999. Online available at http://citeseer.ist.psu.edu/jain99data.html and http://www.cs.rutgers.edu/~mlittman/courses/lightai03/jain99data.pdf [accessed 2007-08-11].

[1030] Lakhmi C. Jain, editor. *Proceedings of the 1st International Conference on Knowledge-Based Intelligent Electronic Systems, KES 1997, Part I*, May 21–23, 1997, Adelaide, South Australia. IEEE. ISBN: 0-7803-3755-7. See also [1031].

[1031] Lakhmi C. Jain, editor. *Proceedings of the 1st International Conference on Knowledge-Based Intelligent Electronic Systems, KES 1997, Part II*, May 21–23, 1997, Adelaide, South Australia. IEEE. ISBN: 0-7803-3755-7. See also [1030].

[1032] Lakhmi C. Jain, editor. *Proceedings of the Third International Conference on Knowledge-Based Intelligent Information Engineering Systems, KES 1999*, August 31–September 1, 1999, Adelaide, South Australia. IEEE. ISBN: 0-7803-5578-4.

[1033] Lakhmi C. Jain and R. K. Jain, editors. *Proceedings of the 2nd International Conference on Knowledge-Based Intelligent Electronic Systems, KES 1998, Part I*, April 21–23, 1998, Adelaide, South Australia. IEEE. See also [1034, 1035].

[1034] Lakhmi C. Jain and R. K. Jain, editors. *Proceedings of the 2nd International Conference on Knowledge-Based Intelligent Electronic Systems, KES 1998, Part II*, April 21–23, 1998, Adelaide, South Australia. IEEE. See also [1033, 1035].

[1035] Lakhmi C. Jain and R. K. Jain, editors. *Proceedings of the 2nd International Conference on Knowledge-Based Intelligent Electronic Systems, KES 1998, Part III*, April 21–23, 1998, Adelaide, South Australia. IEEE. See also [1033, 1034].

[1036] Lakhmi C. Jain and Janusz Kacprzyk, editors. *New Learning Paradigms in Soft Computing*, volume 84 (PR-2001-06) of *Studies in fuzziness and soft computing*. Physica-Verlag (A Springer-Verlag Company), Heidelberg, Germany, February 2002. ISBN: 978-3-79081-436-1. Partly online available at http://books.google.de/books?hl=de&id=u2NcykOhYYwC [accessed 2008-05-31].

[1037] Lakhmi C. Jain, N. Baba, and Robert J. Howlett, editors. *Proceedings of the Fifth International Conference on Knowledge-Based Intelligent Information Engineering*

Systems & Allied Technologies, KES 2001, September 6–8, 2001, Osaka-Kyoiku University, Osaka, Japan and Nara-ken New Public Hall, Nara, Japan. See http://www.bton.ac.uk/kes/kes2001/ [accessed 2008-07-28].

[1038] Jürgen Jakumeit, Thomas Barth, Julian Reichwald, Manfred Grauer, Frank Thilo, Thomas Friese, Matthew Smith, and Bernd Freisleben. A grid-based parallel optimization algorithm applied to a problem in metal casting industry. In *Proceedings of the Second International Conference on Bioinspired Optimization Methods and their Application, BIOMA 2006*, 2006. In proceedings [669].

[1039] Thomas Jansen and Ingo Wegener. A comparison of simulated annealing with a simple evolutionary algorithm on pseudo-boolean functions of unitation. *Theoretical Computer Science*, 386(1-2):73–93, October 2007. ISSN: 0304-3975. doi:10.1016/j.tcs.2007.06.003. Online available at http://dx.doi.org/10.1016/j.tcs.2007.06.003 [accessed 2008-04-07].

[1040] Jiri Jaroš and Vaclav Dvořák. An evolutionary design technique for collective communications on optimal diameter-degree networks. In *Genetic and Evolutionary Computation Conference (GECCO)*, 2008. In proceedings [1117].

[1041] G. A. Jastrebski and D. V. Arnold. Improving evolution strategies through active covariance matrix adaptation. In *Proceedings of IEEE World Congress on Computational Intelligence*, pages 9719–9726, 2006. In proceedings [2291].

[1042] Faizad Javed, Barrett R. Bryant, M. Črepinček, Marjan Mernik, and Alan Sprague. Context-free grammar induction using genetic programming. In *ACM-SE 42: Proceedings of the 42nd annual Southeast regional conference*, pages 404–405, 2004, Huntsville, Alabama. ACM Press, New York, NY, USA. ISBN: 1-5811-3870-9. doi:10.1145/986537.986635. Online available at http://doi.acm.org/10.1145/986537.986635 [accessed 2007-09-09].

[1043] Edwin Thompson Jaynes. *Probability Theory: The Logic of Science*. Washington University, preprint edition, 1996. See also printed version [1044]. G. Larry Bretthorst ed. Online available at http://bayes.wustl.edu/etj/prob/book.pdf [accessed 2007-08-08].

[1044] Edwin Thompson Jaynes. *Probability Theory: The Logic of Science*. Cambridge University Press, June 2003. ISBN: 978-0-52159-271-0. See also [1043]. Larry Bretthorst ed. Online available at http://bayes.wustl.edu/etj/prob/book.pdf [accessed 2007-08-08].

[1045] Wassim Jaziri, editor. *Local Search Techniques: Focus on Tabu Search*. INTECH Education and Publishing, Kirchengasse 43/3, 1070 Vienna, Austria, September 2008. ISBN: 978-3-90261-334-9. Online available at http://intechweb.org/downloadfinal.php?is=978-3-902613-34-9&type=B [accessed 2009-01-13].

[1046] David R. Jefferson, Robert J. Collins, Claus Cooper, Michael Dyer, Margot Flowers, Richard Korf, Charles Tayler, and Alan Wang. Evolution as a theme in artificial life: The genesys/tracker system. In *Artificial Life II*, pages 549–578, 1990. In proceedings [1248].

[1047] Márk Jelasity and Alberto Montresor. Epidemic-style proactive aggregation in large overlay networks. In *Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*, pages 102–109, March 2004. IEEE Computer Society, Tokyo, Japan. Online available at http://citeseer.ist.psu.edu/jelasity04epidemicstyle.html and http://en.scientificcommons.org/9048443 [accessed 2007-08-13].

[1048] Márk Jelasity, Alberto Montresor, and Özalp Babaoğlu. Gossip-based aggregation in large dynamic networks. *ACM Transactions on Computer Systems*, 23(3):219–252, 2005. ISSN: 0734-2071. Online available at http://www.cs.unibo.it/bison/publications/aggregation-tocs.pdf and http://portal.acm.org/citation.cfm?id=1082470 [accessed 2007-08-01].

[1049] Henrik Jeldtoft Jensen. *Self-organized Criticality: Emergent Complex Behavior in Physical and Biological Systems*, volume 10 of *Cambridge Lecture Notes in Physics*. Cambridge University Press, 1998. ISBN: 0-5214-8371-9, 978-0-52148-371-1. Partly online available at http://books.google.de/books?id=7u3x1Luq50cC [accessed 2008-08-23].

[1050] Jaein Jeong, Xiaofan Fred Jiang, and David E. Culler. Design and analysis of micro-solar power systems for wireless sensor networks. Technical Report UCB/EECS-2007-24, EECS Department, University of California, Berkeley, February 2007. Online available at http://www.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-24.html [accessed 2007-08-01].

[1051] Max Jerrell. Applications of public global optimization software to difficult econometric functions. In *Computing in Economics and Finance 200*. Society for Computational Economics, 2000, Departament d'Economia i Empresa, Universitat Pompeu Fabra, Ramon Trias Fargas, 25,27, 08005, Barcelona, Spain. Number 161. Online available at http://econpapers.repec.org/paper/scescecf0/161.htm [accessed 2007-09-01].

[1052] Licheng Jiao, Lipo Wang, Xinbo Gao, Jing Liu, and Feng Wu, editors. *Proceedings of the Second International Conference on Advances in Natural Computation, ICNC 2006, Part I*, volume 4221 of *Lecture Notes in Computer Science (LNCS)*, September 24–28, 2006, Xi'an, China. Springer Berlin / Heidelberg. ISBN: 3-5404-5901-4, 978-3-54045-901-9. doi:10.1007/11881070. See also [1053].

[1053] Licheng Jiao, Lipo Wang, Xinbo Gao, Jing Liu, and Feng Wu, editors. *Proceedings of the Second International Conference on Advances in Natural Computation, ICNC 2006, Part II*, volume 4222 of *Lecture Notes in Computer Science (LNCS)*, September 24–28, 2006, Xi'an, China. Springer Berlin / Heidelberg. ISBN: 3-5404-5907-3, 978-3-54045-907-1. doi:10.1007/11881223. See also [1052].

[1054] Fernando Jiménez, José L. Verdegay, and Antonio F. Gómez-Skarmeta. Evolutionary techniques for constrained multiobjective optimization problems. In Kalyanmoy Deb, editor, *Multi-criterion Optimization Using Evolutionary Methods*, pages 115–116, 1999. In proceedings [142]. Online available at http://citeseer.ist.psu.edu/208779.html [accessed 2007-08-24].

[1055] Yaochu Jin, editor. *Knowledge Incorporation in Evolutionary Computation*, volume 167 of *Studies in Fuzziness and Soft Computing*. Springer, Springer Distribution Center GmbH, Haberstrasse 7, 69126 Heidelberg, Germany, 2004. ISBN: 978-3-54022-902-5. Partly online available at http://books.google.de/books?id=kD_FXkAKn9IC [accessed 2008-05-31].

[1056] Wilhelm Ludvig Johannsen. *Elemente der exakten Erblichkeitslehre (mit Grundzügen der Biologischen Variationsstatistik)*. Gustav Fischer, Jena, Germany, second german, revised and very extended edition, 1909/1913. Limitations of natural selection on pure lines. Online available at http://www.zum.de/stueber/johannsen/elemente/index.html [accessed 2008-08-20].

[1057] Matthias John and Max J. Ammann. Design of a wide-band printed antenna using a genetic algorithm on an array of overlapping sub-patches. In *IEEE International Workshop on Antenna Technology: Small Antennas and Novel Metamaterials (IWAT2006)*, pages 92–95, March 6–8, 2006, White Plains, NY, USA. ISBN: 0-7803-9443-7. Online available at http://www.ctvr.ie/docs/RF%20Pubs/01608983.pdf [accessed 2008-09-03].

[1058] Matthias John and Max J. Ammann. Optimisation of a wide-band printed monopole antenna using a genetic algorithm. In *Loughborough Antennas & Propagation Conference (PAPC)*, pages 237–240. Loughborough University, April 11–12, 2006. Online available at http://www.ctvr.ie/docs/RF%20Pubs/LAPC_2006_MJ.pdf [accessed 2008-09-02].

[1059] *Genetic Algorithms in Engineering and Computer Science, European Short Course on Genetic Algorithms and Evolution Strategies, EUROGEN 1995*, 1995, Las Palmas de Gran Canaria, Spain. John Wiley & Sons.

[1060] Derek M. Johnson, Ankur M. Teredesai, and Robert T. Saltarelli. Genetic programming in wireless sensor networks. In *Genetic Programming*, pages 96–107, 2005. doi:10.1007/b107383. In proceedings [1116]. Online available at http://www.cs.rit.edu/~amt/pubs/EuroGP05FinalTeredesai.pdf [accessed 2008-06-17].

[1061] R. L. Johnston, editor. *Applications of Evolutionary Computation in Chemistry*, volume 110 of *Structure and Bonding*. Springer, Berlin, Germany, 2004. ISBN: 978-3-54040-258-9, 3-5404-0258-6. Partly online available at http://books.google.de/books?id=AzbZIA5aT2oC [accessed 2008-04-03].

[1062] Timothy D. Johnstona. Selective costs and benefits in the evolution of learning. *Advances in the Study of Behavior*, 12:65–106, 1982. ISBN: 978-0-12004-512-9. doi:10.1016/S0065-3454(08)60046-7.

[1063] Jeffrey A. Joines and Christopher R. Houck. On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with ga's. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, pages 579–584, 1994. doi:10.1109/ICEC.1994.349995. In proceedings [1411]. Online available at http://www.cs.cinvestav.mx/~constraint/papers/gacons.ps.gz and http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.28.1065 [accessed 2008-11-15].

[1064] Gareth Jones. Genetic and evolutionary algorithms. In Paul von Ragué Schleyer and Johnny Gasteiger, editors, *Encyclopedia of Computational Chemistry*, volume III - Databases and Expert Systems, chapter 29. John Wiley & Sons, Ltd., September 1998. ISBN: 978-0-47196-588-6. Online available at http://www.wiley.com/legacy/wileychi/ecc/samples/sample10.pdf [accessed 2007-08-17].

[1065] Joel Jones. Abstract syntax tree implementation idioms. In *Proceedings of The 10th Conference on Pattern Languages of Programs PLoP'2003*, September 8–12, 2003, Monticello, Illinois, USA. A workshop presentation online available at http://hillside.net/plop/plop2003/Papers/Jones-ImplementingASTs.pdf [accessed 2007-07-03].

[1066] Josh Jones and Terry Soule. Comparing genetic robustness in generational vs. steady state evolutionary algorithms. In *GECCO'06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 143–150, 2006. In proceedings [352]. Online available at http://doi.acm.org/10.1145/1143997.1144024 [accessed 2007-08-24].

[1067] Terry Jones. A description of holland's royal road function. *Evolutionary Computation*, 2(4):411–417, 1994. See also [1068].

[1068] Terry Jones. A description of holland's royal road function. Working Papers 94-11-059, Santa Fe Institute, 1399 Hyde Park Road, Santa Fe, NM 87501, USA, November 1994. See also [1067].

[1069] Terry Jones. *Evolutionary Algorithms, Fitness Landscapes and Search*. PhD thesis, The University of New Mexico. Albuquerque, New Mexico, USA, May 1995. Online available at http://www.cs.unm.edu/~forrest/dissertations-and-proposals/terry.pdf [accessed 2008-07-02].

[1070] Terry Jones and Stephanie Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 184–192, 1995. In proceedings [636]. Online available at http://citeseer.ist.psu.edu/jones95fitness.html and http://www.santafe.edu/research/publications/workingpapers/95-02-022.ps [accessed 2008-07-20].

[1071] Diane Jordan, John Evdemon, Alexandre Alves, Assaf Arkin, Sid Askary, Charlton Barreto, Ben Bloch, Francisco Curbera, Mark Ford, Yaron Goland, Alejandro Guízar, Neelakantan Kartha, Canyang Kevin Liu, Rania Khalaf, Dieter König, Mike

Marin, Vinkesh Mehta, Satish Thatte, Danny van der Rijn, Prasad Yendluri, and Alex Yiu. *Web Services Business Process Execution Language Version 2.0*. Organization for the Advancement of Structured Information Standards (OASIS), April 11, 2007. Online available at http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf [accessed 2007-10-25]. Technical Committee: OASIS Web Services Business Process Execution Language (WSBPEL) TC.

[1072] Aravind K. Joshi. How much context-sensitivity is necessary for characterizing structural descriptions – tree adjoining grammars. In D. Dowty, L. Karttunen, and A. Zwicky, editors, *Natural Language Processing – Theoretical, Computational and Psychological Perspective*, pages 206–250. Cambridge University Press, New York, NY, 1985. Originally presented in 1983.

[1073] Aravind K. Joshi and Owen Rambow. A formalism for dependency grammar based on tree adjoining grammar. In *Proceedings of the Conference on Meaning-Text Theory, MTT 2003*, June 16–18, 2003, Paris, France. Online available at http://www1.cs.columbia.edu/~rambow/papers/joshi-rambow-2003.pdf [accessed 2007-09-15].

[1074] Aravind K. Joshi and Yves Schabes. Tree-adjoining grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 69–124. Springer, Berlin, New York, 1997. Online available at http://citeseer.ist.psu.edu/joshi97treeadjoining.html [accessed 2007-09-15].

[1075] R. S. Judson, M. E. Colvin, Juan C. Meza, A. Huffer, and D. Gutierrez. Do intelligent configuration search techniques outperform random search for large molecules? *International Journal of Quantum Chemistry*, 44(2):277–290, 1992. ISSN: 0020-7608. CODEN: IJQCB2.

[1076] Mirko Junge. Wilcoxon's signed-rank distribution, October 5, 2000. Online available at http://www.geocities.com/junge_m/pdf/wilcoxon.pdf [accessed 2008-08-18].

K

[1077] Professor Kaelo. *Some population-set based methods for unconstrained global optimization*. PhD thesis, School of Computational and Applied Mathematics, Witwatersrand University, Private-Bag-3, Wits-2050, Johannesburg, South Africa, 2005. Online available at http://hdl.handle.net/123456789/1771 [accessed 2008-07-23].

[1078] Professor Kaelo and M. Montaz Ali. A numerical study of some modified differential evolution algorithms. *European Journal of Operational Research*, 169(3):1176–1184, March 16, 2006. ISSN: 0377-2217. doi:10.1016/j.ejor.2004.08.047. Online available at http://witsetd.wits.ac.za:8080/dspace/bitstream/123456789/1771/2/PaperA.pdf and http://dx.doi.org/10.1016/j.ejor.2004.08.047 [accessed 2008-07-23].

[1079] Professor Kaelo and M. Montaz Ali. Differential evolution algorithms using hybrid mutation. *Computational Optimization and Applications*, 37(2):231–246, June 2007. ISSN: 0926-6003 (Print) 1573-2894 (Online). doi:10.1007/s10589-007-9014-3.

[1080] Tatiana Kalganova. An extrinsic function-level evolvable hardware approach. In *Genetic Programming, Proceedings of EuroGP'2000*, pages 60–75, 2000. In proceedings [1666]. Online available at http://citeseer.ist.psu.edu/kalganova00extrinsic.html [accessed 2007-09-09].

[1081] Tatiana Kalganova and Julian Francis Miller. Evolving more efficient digital circuits by allowing circuit layout evolution and multi-objective fitness. In Adrian Stoica, Jason Lohn, and Didier Keymeulen, editors, *EH'99: The First NASA/DoD Workshop on Evolvable Hardware*, pages 54–63, July 19-21, 1999. IEEE Computer Society, Pasadena, California. ISBN: 0-7695-0256-3. Online available at http://citeseer.ist.psu.edu/kalganova99evolving.html [accessed 2007-11-03].

[1082] Tatiana Kalganova and Julian Francis Miller. Evolving more efficient digital circuits by allowing circuit layout evolution and multi-objective fitness. In *Evolv-*

able Hardware, Proceedings of 1st NASA/DoD Workshop on Evolvable Hardware (EH'99), page 54. IEEE Computer Society, July 1999, Pasadena, CA, USA. ISBN: 0-7695-0256-3. Online available at http://citeseer.ist.psu.edu/kalganova99evolving.html [accessed 2007-09-09].

[1083] Leila Kallel, Bart Naudts, and Alex Rogers, editors. *Theoretical Aspects of Evolutionary Computing*. Natural Computing Series. Springer, Berlin/London, UK, June 15, 2001. ISBN: 3-5406-7396-2, 978-3-54067-396-5. Series editors: G. Rozenberg, Thomas Bäck, Ágoston E. Eiben, J.N. Kok, and H.P. Spaink.

[1084] Olav Kallenberg. *Foundations of modern probability*. Probability and Its Applications. Springer, New York, second edition, January 8, 2002. ISBN: 978-0-38795-313-7. OCLC: 60740995. Partly online available at http://books.google.de/books?id=L6fhXh13OyMC [accessed 2008-08-19].

[1085] Olav Kallenberg. *Probabilistic symmetries and invariance principles*. Probability and its Applications. Springer, New York, July 27, 2005. ISBN: 978-0-38725-115-8, 0-3879-4957-7, 0-3879-5313-2. OCLC: 46937587.

[1086] Subbarao Kambhampati and Craig A. Knoblock, editors. *Proceedings of IJCAI-03 Workshop on Information Integration on the Web (IIWeb-03)*, August 9–10, 2003, Acapulco, México. Online available at http://www.isi.edu/info-agents/workshops/ijcai03/proceedings.htm [accessed 2008-04-01]. See also [844].

[1087] Peter Kampstra. Evolutionary computing in telecommunications – a likely ec success story. Master's thesis, Vrije Universiteit, Amsterdam, August 2005. BMI Paper, Supervisors: Rob D. van der Mei and Gusz Eiben (≡ Ágoston E. Eiben). Online available at http://www.few.vu.nl/stagebureau/werkstuk/werkstukken/werkstuk-kampstra.pdf [accessed 2008-09-04].

[1088] Peter Kampstra, Rob D. van der Mei, and Ágoston E. Eiben. Evolutionary computing in telecommunication network design: A survey, 2006. Online available at http://www.few.vu.nl/~mei/articles/2006/kampstra/art.pdf [accessed 2008-09-04].

[1089] Ali Kamrani, Wang Rong, and Ricardo Gonzalez. A genetic algorithm methodology for data mining and intelligent knowledge acquisition. *Computers & Industrial Engineering*, 40(4):361–377, September 2001. ISSN: 0360-8352. doi:10.1016/S0360-8352(01)00036-5. Online available at http://dx.doi.org/10.1016/S0360-8352(01)00036-5 [accessed 2009-05-12].

[1090] Cem Kaner, Jack Falk, and Hung Quoc Nguyen. *Testing Computer Software*. John Wiley and Sons, second edition, April 12, 1993. ISBN: 0-4713-5846-0, 978-0-47135-846-6.

[1091] Gopal K. Kanji. *100 Statistical Tests*. Sage Publications Ltd, April 1999. ISBN: 0-7619-6151-8, 978-0-76196-151-2.

[1092] Holger Karl and Andreas Willig. *Protocols and Architectures for Wireless Sensor Networks*. Wiley & Sons, first edition, June 24, 2005. ISBN: 978-0-47009-510-2.

[1093] Charles L. Karr and L. Michael Freeman, editors. *Industrial Applications of Genetic Algorithms*. International Series on Computational Intelligence. CRC Press, Boca Raton, FL, December 29, 1998. ISBN: 0-8493-9801-0, 978-0-84939-801-8.

[1094] Charles L. Karr and Eric Wilson. A self-tuning evolutionary algorithm applied to an inverse partial differential equation. *Applied Intelligence*, 19(3):147–155, November 2003. ISSN: 0924-669X (Print) 1573-7497 (Online). doi:10.1023/A:1026097605403. Online available at http://www.springerlink.com/content/x7t26442h41q0221/fulltext.pdf and http://dx.doi.org/10.1023/A:1026097605403 [accessed 2007-08-27].

[1095] Hironobu Katagiri, Kotaro Hirasawa, Jinglu Hu, and Junichi Murata. Network structure oriented evolutionary model– genetic network programming – and its comparison with genetic programming. In *Genetic and Evolutionary Computation Conference Late Breaking Papers*, pages 219–226, 2001. In proceedings [833]. See also [1096, 1097].

[1096] Hironobu Katagiri, Kotaro Hirasawa, Jinglu Hu, and Junichi Murata. Network structure oriented evolutionary model – genetic network programming – and its compar-

ison with genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, page 179, 2001. Poster online available at http://www.cs.bham.ac.uk/~wbl/biblio/gecco2001/d02.pdf [accessed 2008-06-15]. See also [1095, 1097].

[1097] Hironobu Katagiri, Kotaro Hirasawa, Jinglu Hu, Junichi Murata, and M. Kosaka. Network structure oriented evolutionary model: Genetic network programming – its comparison with genetic programming –. *Transactions of the Society of Instrument and Control Engineers*, 38(5):485–494, May 2002. ISSN: 0453-4654. Accession number: 02A0504907. Journal Code: S0104A. See also [1095, 1096].

[1098] Stuart Alan Kauffman. Adaptation on rugged fitness landscapes. In Daniel L. Stein, editor, *Lectures in the Sciences of Complexity: The Proceedings of the 1988 Complex Systems Summer School*, volume Lecture I of *Santa Fe Institute Studies in the Sciences of Complexity*, pages 527–618. Addison Wesley Publishing Company, Redwood City, June-July 1988, Santa Fe, New Mexico, USA. ISBN: 978-0-20151-015-7, 0-2015-1015-4. Published in September 1989.

[1099] Stuart Alan Kauffman. *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press, May 1993. ISBN: 0-1950-7951-5.

[1100] Stuart Alan Kauffman and Simon Asher Levin. Towards a general theory of adaptive walks on rugged landscapes. *Journal of Theoretical Biology*, 128(1):11–45, September 7, 1987. ISSN: 0022-5193.

[1101] Stuart Alan Kauffman and Edward D. Weinberger. The nk model of rugged fitness landscapes and its application to maturation of the immune response. *Journal of Theoretical Biology*, 141(2):211–245, November 21, 1989. Online available at http://dx.doi.org/10.1016/S0022-5193(89)80019-0 [accessed 2007-10-14].

[1102] Ulrich Kaufmann, Roland Reichle, Christof Hoppe, and Philipp Andreas Baer. An unsupervised approach for adaptive color segmentation. In *International Workshop on Robot Vision, VISAPP 2007*. Springer, 2007. Online available at http://neuro.informatik.uni-ulm.de/basilic/Publications/2007/KRHB07/AdaptColorSeg-388.pdf [accessed 2007-09-15].

[1103] Henry Kautz and Bruce Porter, editors. *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI) and Twelfth Conference on Innovative Applications of Artificial Intelligence (IAAI)*, July 30–August 3, 2000, Austin, Texas, USA. AAAI Press/The MIT Press. ISBN: 0-2625-1112-6. See http://www.aaai.org/Conferences/AAAI/aaai00.php [accessed 2007-09-06] and http://www.aaai.org/Conferences/IAAI/iaai00.php [accessed 2007-09-06].

[1104] W. H. Kautz. Unit-distance error-checking codes. *IRE Transactions on Electronic Computers*, EC-7:177–180, 1958. ISSN: 0367-9950.

[1105] Steven M. Kay. *Fundamentals of Statistical Signal Processing, Volume I: Estimation Theory*, volume 1. Prentice Hall PTR, us es edition, March 26, 1993. ISBN: 978-0-13345-711-7.

[1106] Uzay Kaymak and Magne Setnes. Extended fuzzy clustering algorithms. Research Paper ERS; ERS-2000-51-LIS, Erasmus Research Institute of Management (ERIM), RSM Erasmus University, November 2000. Online available at https://ep.eur.nl/handle/1765/57 and http://ideas.repec.org/p/dgr/eureri/200050.html [accessed 2007-08-11].

[1107] Sanza Kazadi, H. Lin, J. Ogita, V. Huang, P. Hung, D. Lee, and D. Tsikata. Conjugate schema and basis representation of crossover and mutation. *Evolutionary Computation*, 6(2):129–160, September 1998. Online available at http://journal-ci.csse.monash.edu.au/ci/vol07/kazadi01/kazadi01.pdf [accessed 2008-10-31].

[1108] Michael J. Kearns, Yishay Mansour, Andrew Y. Ng, and Dana Ron. An experimental and theoretical comparison of model selection methods. In *COLT'95: Proceedings of the eighth annual conference on Computational learning theory*, pages 21–30. ACM Press, New York, NY, USA, 1995, Santa Cruz, California, United States. ISBN:

0-8979-1723-5. doi:10.1145/225298.225301. Online available at http://www.cis.upenn.edu/~mkearns/papers/ms.pdf and http://doi.acm.org/10.1145/225298.225301 [accessed 2008-03-02]. See also [1109].

[1109] Michael J. Kearns, Yishay Mansour, Andrew Y. Ng, and Dana Ron. An experimental and theoretical comparison of model selection methods. *Machine Learning*, 27(1):7–50, 1997. See also [1108].

[1110] Tom Kehler and Stan Rosenschein, editors. *Proceedings of the 5th National Conference on Artificial Intelligence, AAAI*, volume Science (volume 1), August 11–15, 1986, Philadelphia, PA, USA. Morgan Kaufmann, San Francisco, CA, USA. See http://www.aaai.org/Conferences/AAAI/aaai86.php [accessed 2007-09-06] and also [1111].

[1111] Tom Kehler and Stan Rosenschein, editors. *Proceedings of the 5th National Conference on Artificial Intelligence, AAAI*, volume Engineering (volume 2), August 11–15, 1986, Philadelphia, PA, USA. Morgan Kaufmann, San Francisco, CA, USA. See http://www.aaai.org/Conferences/AAAI/aaai86.php [accessed 2007-09-06] and also [1110].

[1112] Maarten Keijzer. Scaled symbolic regression. *Genetic Programming and Evolvable Machines*, 5(3):259–269, September 2004. ISSN: 1389-2576 (Print) 1573-7632 (Online). doi:10.1023/B:GENP.0000030195.77571.f9. Online available at http://www.springerlink.com/content/x035121165125175/fulltext.pdf and http://dx.doi.org/10.1023/B:GENP.0000030195.77571.f9 [accessed 2007-09-09].

[1113] Maarten Keijzer, editor. *Late Breaking Papers at the 2004 Genetic and Evolutionary Computation Conference*, June 26–30, 2004, Red Lion Hotel, 1415 5th Avenue, Seattle, WA 98101, USA. Springer Berlin/Heidelberg. Distributed on CD-ROM and (online) in [545].

[1114] Maarten Keijzer, J. J. Merelo, G. Romero, and Marc Schoenauer. Evolving objects: A general purpose evolutionary computation library. In *Proceedings of 5th International Conference on Artificial Evolution, Evolution Artificielle, EA 2001*, pages 829–888, 2001. In proceedings [428]. Online available at http://www.lri.fr/~marc/EO/EO-EA01.ps.gz and http://citeseer.ist.psu.edu/keijzer01evolving.html [accessed 2007-08-24]. See also http://eodev.sourceforge.net/ [accessed 2007-08-24].

[1115] Maarten Keijzer, Una-May O'Reilly, Simon M. Lucas, Ernesto Costa, and Terence Soule, editors. *Proceedings of the 7th European Conference on Genetic Programming, EuroGP2004*, volume 3003/2004 of *Lecture Notes in Computer Science (LNCS)*, April 5-7, 2004, Coimbra, Portugal. Springer Berlin/Heidelberg. ISBN: 3-5402-1346-5.

[1116] Maarten Keijzer, Andrea Tettamanzi, Pierre Collet, Jano I. van Hemert, and Marco Tomassini, editors. *Proceedings of the 8th European Conference on Genetic Programming, EuroGP2005*, volume 3447/2005 of *Lecture Notes in Computer Science (LNCS)*, March 30-April 1, 2005, Lausanne, Switzerland. Springer Berlin/Heidelberg. ISBN: 3-5402-5436-6. See http://evonet.lri.fr/eurogp2005/ [accessed 2007-09-01].

[1117] Maarten Keijzer, Giuliano Antoniol, Clare Bates Congdon, Kalyanmoy Deb, Benjamin Doerr, Nikolaus Hansen, John H. Holmes, Gregory S. Hornby, Daniel Howard, James Kennedy, Sanjeev Kumar, Fernando G. Lobo, Julian Francis Miller, Jason Moore, Frank Neumann, Martin Pelikan, Jordan Pollack, Kumara Sastry, Kenneth Stanley, Adrian Stoica, El-Ghazali Talbi, and Ingo Wegener, editors. *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2008*, July 12–16, 2008, Renaissance Atlanta Hotel Downtown, 590 West Peachtree Street NW, Atlanta, Georgia 30308 USA. ACM Press, New York, NY, 10286-1405, USA. ISBN: 978-1-60558-131-6. ACM Order Number: 910081. See http://www.sigevo.org/gecco-2008/ [accessed 2008-07-16]. See also [409, 1393, 1911, 1705].

[1118] Jozef Kelemen and Petr Sosík, editors. *Advances in Artificial Life, Proceedings of the 6th European Conference, ECAL 2001*, volume 2159/-1 of *Lecture Notes in Computer Science (LNCS)*, September 10–14, 2001, Prague, Czech Republic. Springer. ISBN: 3-5404-2567-5, 978-3-54042-567-0. doi:10.1007/3-540-44811-X.

[1119] Robert E. Keller and Wolfgang Banzhaf. Genetic programming using mutation, reproduction and genotype-phenotype mapping from linear binary genomes into linear lalr(1) phenotypes. In *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 116–122, 1996. In proceedings [1207]. Online available at http://citeseer.ist.psu.edu/4569.html [accessed 2007-09-09].

[1120] Robert E. Keller and Wolfgang Banzhaf. Genetic programming using genotype-phenotype mapping from linear genomes into linear phenotypes. In *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 116–122, 1996. In proceedings [1207]. Online available at http://citeseer.ist.psu.edu/385878.html and http://web.cs.mun.ca/~banzhaf/papers/lalr_gp96.ps.gz [accessed 2007-09-09].

[1121] Robert E. Keller and Wolfgang Banzhaf. The evolution of genetic code in genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1077–1082, 1999. In proceedings [142]. Online available at http://citeseer.ist.psu.edu/244531.html and http://www.cs.mun.ca/~banzhaf/papers/t.pdf [accessed 2007-09-09].

[1122] Robert E. Keller, Wolfgang Banzhaf, Jörn Mehnen, and Klaus Weinert. Cad surface reconstruction from digitized 3d point data with a genetic programming/evolution strategy hybrid. In *Advances in genetic programming: volume 3*, chapter 3, pages 41–65. The MIT Press, 1999. In collection [1936]. Online available at http://www.cs.bham.ac.uk/~wbl/aigp3/ch03.pdf [accessed 2007-08-18].

[1123] David Kempe, Alin Dobra, and Johannes Gehrke. Gossip-based computation of aggregate information. In *Proceedings of 44th Symposium on Foundations of Computer Science (FOCS 2003)*, pages 482–491, October 11–14, 2003, Cambridge, MA, USA. IEEE Computer Society, Los Alamitos, CA, USA. ISBN: 0-7695-2040-5. Online available at http://www.cs.cornell.edu/johannes/papers/2003/focs2003-gossip.pdf and http://citeseer.ist.psu.edu/kempe03gossipbased.html [accessed 2007-09-15].

[1124] James Kennedy and Russel C. Eberhart. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks, 1995*, volume 4, pages 1942–1948, November 27–December 1, 1995, Perth, WA, Australia. ISBN: 0-7803-2768-3. Online available at http://www.engr.iupui.edu/~shi/Coference/psopap4.html [accessed 2007-08-21].

[1125] James Kennedy, Russell C. Eberhart, and Yuhui Shi. *Swarm Intelligence: Collective, Adaptive*. Morgan Kaufmann, 2001. ISBN: 1-5586-0595-9, 978-1-55860-595-4. Partly online available at http://books.google.com/books?id=pHku3gYfL2UC&hl=de [accessed 2008-08-22] and online available at http://www.engr.iupui.edu/~eberhart/web/PSObook.html [accessed 2008-08-22].

[1126] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language*. Prentice Hall, Inc., Englewood Cliffs, New Jersey, USA, first/second edition, 1978/1988. ISBN: 0-1311-0163-3, 0-1311-0362-8, 0-1311-0370-9, 978-0-13110-362-7.

[1127] M. E. Keskin and Özlem Terzi. Modeling water temperature using gene expression programming. In *Proceedings of the 14th Turkish Symposium on Artificial Intelligence and Neural Networks, TAINN 2005*, pages 280–285, 2005, Izmir, Turkey.

[1128] Eik Fun Khor, Kay Chen Tan, Tong Heng Lee, and C. K. Goh. A study on distribution preservation mechanism in evolutionary multi-objective optimization. *Artificial Intelligence Review*, 23(1):31–33, March 2005. ISSN: 0269-2821 (Print) 1573-7462 (Online). doi:10.1007/s10462-004-2902-3. Online available at http://www.springerlink.com/content/h3w111g418gn1045/fulltext.pdf [accessed 2007-08-25].

[1129] Rajiv Khosla, Robert J. Howlett, and Lakhmi C. Jain, editors. *Proceedings of the 9th International Conference on Knowledge-Based Intelligent Information and Engineering Systems, KES 2005, Part I*, volume 3681 of *Lecture Notes in Computer*

Science (LNCS), September 14–16, 2005, Melbourne, Australia. Springer. ISBN: 3-5402-8894-5. See also [1130, 1131, 1132].

[1130] Rajiv Khosla, Robert J. Howlett, and Lakhmi C. Jain, editors. *Proceedings of the 9th International Conference on Knowledge-Based Intelligent Information and Engineering Systems, KES 2005, Part II*, volume 3682 of *Lecture Notes in Computer Science (LNCS)*, September 14–16, 2005, Melbourne, Australia. Springer. ISBN: 3-5402-8895-3. See also [1129, 1131, 1132].

[1131] Rajiv Khosla, Robert J. Howlett, and Lakhmi C. Jain, editors. *Proceedings of the 9th International Conference on Knowledge-Based Intelligent Information and Engineering Systems, KES 2005, Part III*, volume 3683 of *Lecture Notes in Computer Science (LNCS)*, September 14–16, 2005, Melbourne, Australia. Springer. ISBN: 3-5402-8896-1. See also [1129, 1131, 1132].

[1132] Rajiv Khosla, Robert J. Howlett, and Lakhmi C. Jain, editors. *Proceedings of the 9th International Conference on Knowledge-Based Intelligent Information and Engineering Systems, KES 2005, Part IV*, volume 3684 of *Lecture Notes in Computer Science (LNCS)*, September 14–16, 2005, Melbourne, Australia. Springer. ISBN: 3-5402-8897-X. See also [1129, 1130, 1131].

[1133] Sami Khuri and Teresa Chiu. Heuristic algorithms for the terminal assignment problem. In *SAC'97: Proceedings of the 1997 ACM symposium on Applied computing*, pages 247–251, February 28–March 2, 1997, San Jose, California, United States. ACM, New York, NY, USA. ISBN: 0-8979-1850-9. doi:10.1145/331697.331748. Online available at http://doi.acm.org/10.1145/331697.331748 and http://citeseer.ist.psu.edu/khuri97heuristic.html [accessed 2008-06-15].

[1134] Sami Khuri, Thomas Bäck, and Jörg Heitkötter. The zero/one multiple knapsack problem and genetic algorithms. In *Proceedings of the 1994 ACM Symposium on Applied Computing, SAC'94*, pages 188–193, March 6–8, 1994, Phoenix, Arizona, USA. ACM, New York, NY, USA. ISBN: 0-8979-1647-6. doi:10.1145/326619.326694. Online available at http://doi.acm.org/10.1145/326619.326694 and http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.40.4393 [accessed 2008-06-23].

[1135] Jong-Hwan Kim and Jeong-Yul Jeon. Evolutionary programming-based high-precision controller design. In *Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming*, pages 73–81, 1996. In proceedings [709].

[1136] Jong-Hwan Kim and Hyun-Sik Shim. Evolutionary programming-based optimal robust locomotion control of autonomous mobile robots. In *Evolutionary Programming IV: Proceedings of the Fourth Annual Conference on Evolutionary Programming*, pages 631–644, 1995. In proceedings [1380].

[1137] Motoo Kimura. Evolutionary rate at the molecular level. *Nature*, 217(129):624–626, February 1968.

[1138] Motoo Kimura. *The Neutral Theory of Molecular Evolution*. Cambridge University Press, 1983. ISBN: 978-0-52131-793-1. reprint edition, 1985, February.

[1139] Ross Kindermann and J. Laurie Snell. *Markov Random Fields and Their Applications*. American Mathematical Society (AMS), Providence, Rhode Island, USA, 1980. ISBN: 0-8218-3381-2, 0-8218-5001-6. CONM/1.E. Online available at http://www.ams.org/online_bks/conm1/ [accessed 2008-03-28].

[1140] Kenneth E. Kinnear, Jr., editor. *Advances in Genetic Programming*, volume 1 of *Complex Adaptive Systems*. MIT Press, Cambridge, MA, USA, April 7, 1994. ISBN: 0-2621-1188-8, 978-0-26211-188-1. Partly online available at http://books.google.com/books?id=eu2JplnQdBkC [accessed 2008-09-16].

[1141] Kenneth E. Kinnear, Jr. Fitness landscapes and difficulty in genetic programming. In *Proceedings of the 1994 IEEE World Conference on Computational Intelligence*, volume 1, pages 142–147, 1994. doi:10.1109/ICEC.1994.350026. In proceedings [1411]. Online available at http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/

ieee94_kinnear.html and http://citeseer.ist.psu.edu/kinnear94fitness.html [accessed 2008-05-11].

[1142] Scott Kirkpatrick, C. Daniel Gelatt, Jr., and Mario P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 13, 1983. doi:10.1126/science.220.4598.671. Online available at http://fezzik.ucd.ie/msc/cscs/ga/kirkpatrick83optimization.pdf and http://citeseer.ist.psu.edu/kirkpatrick83optimization.html [accessed 2008-03-26].

[1143] I. M. A. Kirkwood, S. H. Shami, and Mark C. Sinclair. Discovering simple fault-tolerant routing rules using genetic programming. In *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms*, 1997. In proceedings [1902]. Online available at http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/kirkam_1997_dsftrr.html and http://citeseer.ist.psu.edu/120285.html [accessed 2008-06-24]. See also [1857].

[1144] Marc Kirschner and John Gerhart. Evolvability. *Proceedings of the National Academy of Science of the USA (PNAS)*, 95(15):8420–8427, July 21, 1998. ISSN: 0027-8424. Online available at http://www.pnas.org/cgi/content/full/95/15/8420 [accessed 2007-08-05].

[1145] Kim Kirsner, Craig Speelman, Murray Maybery, Angela O'Brien-Malone, Mike Anderson, and Colin MacLeod, editors. *Implicit and Explicit Mental Processes*. Lawrence Erlbaum Associates, Mahwah, New Jersey/London, USA, 1998. ISBN: 0-8058-1359-4, 978-0-80581-359-3. Partly online available at http://books.google.de/books?id=3o3mFVUtkjkC and http://www.questia.com/read/89307263?title=Implicit%20and%20Explicit%20Mental%20Processes [accessed 2008-12-01].

[1146] Hajime Kita and Yasuhito Sano. Genetic algorithms for optimization of noisy fitness functions and adaptation to changing environments. In *2003 Joint Workshop of Hayashibara Foundation and 2003 Workshop on Statistical Mechanical Approach to Probabilistic Information Processing (SMAPIP)*, July 14–15, 2003, Okayama, Japan. Online available at http://www.smapip.is.tohoku.ac.jp/~smapip/2003/hayashibara/proceedings/HajimeKita.pdf [accessed 2008-07-19].

[1147] Stefan Klahold, Steffen Frank, Robert E. Keller, and Wolfgang Banzhaf. Exploring the possibilities and restrictions of genetic programming in Java bytecode. In *Late Breaking Papers at the Genetic Programming 1998 Conference*, pages 120–124, 1998. In proceedings [1198].

[1148] Stephen Cole Kleene. On notation for ordinal numbers. *The Journal of Symbolic Logic*, 3(4):150–155, 1938.

[1149] Jack P. C. Kleijnen. Experimental design for sensitivity analysis, optimization, and validation of simulation models. In *Handbook of Simulation*, chapter 6, pages 173–223. Wiley & Sons, 1998. In collection [134]. Online available at http://arno.uvt.nl/show.cgi?fid=3572 [accessed 2008-10-14].

[1150] D. G. Kleinbaum, L. L. Kupper, and K. E. Muller, editors. *Applied regression analysis and other multivariable methods*. PWS Publishing Co., Boston, MA, USA, 1988. ISBN: 0-8715-0123-6.

[1151] Mieczyslaw A. Kłopotek, Slawomir T. Wierzchoń, and Krzysztof Trojanowski, editors. *Intelligent Information Processing and Web Mining: Proceedings of the International IIS: IIPWMA05*, Advances in Soft Computing, June 2005, Gdansk, Poland. Springer, Berlin, Heidelberg, New York. ISBN: 978-3-54025-055-2.

[1152] Anthony C. Klug. Equivalence of relational algebra and relational calculus query languages having aggregate functions. *Journal of the ACM (JACM)*, 29(3):699–717, July 1982. ISSN: 0004-5411. Online available at http://doi.acm.org/10.1145/322326.322332 [accessed 2007-07-28].

[1153] Dimitri Knjazew and David E. Goldberg. Solving permutation problems with ordering messy genetic algorithms. In *Advances in Evolutionary Computing – Theory and Applications*, pages 321–. Springer, 2003. In collection [1221].

[1154] Joshua D. Knowles and David W. Corne. Properties of an adaptive archiving algorithm for storing nondominated vectors. *IEEE Transactions on Evolutionary Computation*, 7:100–116, April 2003.

[1155] Barbara J. Knowlton and Larry R. Squire. Remembering and knowing: two different expressions of declarative memory. *Journal of Experimental Psychology Learning, Memory and Cognition*, 21(3):699–710, May 1995. ISSN: 0278-7393.

[1156] Donald E. Knuth. Backus normal form vs. backus naur form. *Communications of the ACM*, 7(12):735–736, 1964. ISSN: 0001-0782. doi:10.1145/355588.365140. Online available at http://doi.acm.org/10.1145/355588.365140 [accessed 2007-09-15].

[1157] Donald E. Knuth. Semantics of context-free languages. *Theory of Computing Systems/Mathematical Systems Theory*, 2(2):127–145, 1968. ISSN: 1432-4350 (Print) 1433-0490 (Online). doi:10.1007/BF01692511. See [1158]. Online available at http://www.springerlink.com/content/m2501m07m4666813/fulltext.pdf [accessed 2007-09-15].

[1158] Donald E. Knuth. Correction: Semantics of context-free languages. *Theory of Computing Systems/Mathematical Systems Theory*, 5(1):95–96, 1971. ISSN: 1432-4350 (Print) 1433-0490 (Online). doi:10.1007/BF01702865. See [1157]. Online available at http://www.springerlink.com/content/rj10u682v25g6506/fulltext.pdf [accessed 2007-09-15].

[1159] Donald E. Knuth. Big omicron and big omega and big theta. *SIGACT News*, 8 (2):18–24, 1976. ISSN: 0163-5700. Online available at http://portal.acm.org/citation.cfm?id=1008328.1008329 [accessed 2007-08-11].

[1160] Donald E. Knuth. The genesis of attribute grammars. In *WAGA: Proceedings of the international conference on Attribute grammars and their applications*, pages 1–12, 1990, Paris, France. Springer-Verlag New York, Inc., New York, NY, USA. ISBN: 0-3875-3101-7. Online available at http://www.dcs.warwick.ac.uk/~sk/cs325/gag.pdf [accessed 2007-09-15].

[1161] Donald Ervin Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Addison-Wesley, third edition, 1997. ISBN: 0-2018-9684-2, 978-0-20103-802-6.

[1162] Donald Ervin Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming (TAOCP)*. Addison-Wesley, Reading, Massachusetts, third edition, 1997. ISBN: 0-2018-9683-4.

[1163] Donald Ervin Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming (TAOCP)*. Addison-Wesley, Reading, Massachusetts, second edition, 1998. ISBN: 0-2018-9685-0.

[1164] King-Tim Ko, Kit-Sang Tang, Cheung-Yau Chan, Kim-Fung Man, and Sam Kwong. Using genetic algorithms to design mesh networks. *Computer*, 30(8):56–61, August 1997. ISSN: 0018-9162. doi:10.1109/2.607086.

[1165] Geoff Koch. Discovering multi-core: Extending the benefits of moore's law. *Technology@Intel Magazine*, July 2005. Online available at http://www.intel.com/technology/magazine/computing/multi-core-0705.pdf [accessed 2007-09-11]. See also http://www.intel.com/technology/magazine/ [accessed 2007-09-11].

[1166] Peter Köchel. *Algorithmen und Programmierung (Course Material)*. TU Chemnitz, Fakultät für Informatik, Professur Modellierung und Simulation, Straße der Nationen 62, 09107 Chemnitz, Germany, 2007. Online available at http://www.tu-chemnitz.de/informatik/ModSim/ [accessed 2007-07-03].

[1167] Murat Köksalan and Stanley Zionts, editors. *Proceedings of the 15th International Conference on Multiple Criteria Decision Making: Multiple Criteria Decision Making in the New Millennium (MCDM'2000)*, volume 507 of *Lecture Notes in Economics and Mathematical Systems*, July 10–14, 2000, Middle East Technical University, Ankara, Turkey. Springer. ISBN: 978-3-54042-377-5. See http://mcdm2000.ie.metu.edu.tr/ [accessed 2007-09-10]. Published November 9, 2001.

[1168] Krasimir Kolarov. Landscape ruggedness in evolutionary algorithms. In *Proceedings of the IEEE Conference on Evolutionary Computation*, pages 19–24, 1997. In proceedings [106]. Online available at http://citeseer.ist.psu.edu/14650.html and http://de.scientificcommons.org/27191 [accessed 2007-08-19].

[1169] Andrei Nikolajevich Kolmogorov. *Foundations of the Theory of Probability*. Chelsa Publishing Company New York, second edition, 1956, June 1960. ISBN: 978-0-82840-023-7. monograph "Grundbegriffe der Wahrscheinlichkeitsrechnung" 1933, book 1950, Online available at http://www.mathematik.com/Kolmogorov/ [accessed 2007-09-15].

[1170] Andreas König, Mario Köppen, Ajith Abraham, Christian Igel, and Nikola Kasabov, editors. *7th International Conference on Hybrid Intelligent Systems (HIS 2007)*, September 17–19, 2007, Fraunhofer-Center, University of Kaiserslautern, Kaiserslautern, Germany. IEEE Computer Society. ISBN: 0-7695-2946-1. Library of Congress Control Number: 2007936727. Product Number E2946. see http://his07.hybridsystem.com/ [accessed 2007-09-01].

[1171] Erricos John Kontoghiorghes, editor. *Handbook of Parallel Computing and Statistics*. Statistics: Textbooks and Monographs. Chapman & Hall / CRC Press / Marcel Dekker Inc, December 21, 2005. ISBN: 0-8247-4067-X, 978-0-82474-067-2. Partly online available at http://books.google.de/books?id=BnNnKPkFH2kC [accessed 2008-08-03].

[1172] Sanjeev Jagannatha Koppal and Srinivasa G Narasimhan. Clustering appearance for scene analysis. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 1323–1330, June 2006. Online available at http://www.ri.cmu.edu/pubs/pub_5376.html [accessed 2007-08-11].

[1173] Mario Köppen, David H. Wolpert, and William G. Macready. Remarks on a recent paper on the "no free lunch" theorems. *IEEE Transactions on Evolutionary Computation*, 5(3):295–296, June 2001. doi:10.1109/4235.930318. Online available at http://www.no-free-lunch.org/KoWM01.pdf and http://ti.arc.nasa.gov/people/dhw/papers/76.ps [accessed 2008-03-28].

[1174] Emin Erkan Korkmaz and Göktürk Üçoluk. Genetic programming for grammar induction. In *Genetic and Evolutionary Computation Conference 2001 – Late Breaking Papers*, pages 245–251, 2001. In proceedings [833]. Online available at http://citeseer.ist.psu.edu/451812.html and http://de.scientificcommons.org/464341 [accessed 2007-10-14].

[1175] András Kornai. Natural languages and the chomsky hierarchy. In *Proceedings of the second conference on European chapter of the Association for Computational Linguistics*, pages 1–7, 1985, Geneva, Switzerland. Association for Computational Linguistics, Morristown, NJ, USA. doi:10.3115/976931.976932. Online available at http://portal.acm.org/citation.cfm?doid=976931.976932 and http://www.aclweb.org/anthology-new/E/E85/E85-1001.pdf [accessed 2007-09-14].

[1176] Peter Korošec and Juri Šilc. Real-parameter optimization using stigmergy. In *Proceedings of the Second International Conference on Bioinspired Optimization Methods and their Application, BIOMA 2006*, pages 73–84, October 2006. Online available at http://csd.ijs.si/silc/articles/BIOMA06DASA.pdf [accessed 2007-08-05].

[1177] Witold Kosiński, editor. *Advances in Evolutionary Algorithms*. IN-TECH Education and Publishing, Kirchengasse 43/3, 1070 Vienna, Austria, November 2008. ISBN: 978-9-53761-911-4. Online available at http://intechweb.org/downloadfinal.php?is=%20978-953-7619-11-4&type=B [accessed 2009-01-13].

[1178] Alex Kosorukoff. Human-based genetic algorithm. In *Proceedings of the 2001 IEEE International Conference on Systems, Man, and Cybernetics*, volume 5, pages 3464–3469, January 2001, Tucson, AZ, USA. ISBN: 0-7803-7087-2. Also: IlliGAL Report No. 2001004, Online available at http://citeseer.ist.psu.edu/

kosorukoff01human.html and http://citeseer.ist.psu.edu/441929.html [accessed 2007-07-28].

[1179] Tim Kovacs. Xcs classifier system reliably evolves accurate, complete, and minimal representations for boolean functions. In *Soft Computing in Engineering Design and Manufacturing*, pages 59–68. Springer-Verlag, 1998. In collection [379]. Online available at http://www.cs.bris.ac.uk/Publications/Papers/1000239.pdf [accessed 2007-08-18].

[1180] Tim Kovacs. Two views of classifier systems. In *Fourth International Workshop on Learning Classifier Systems - IWLCS-2001*, pages 367–371, 7 2001. San Francisco, California, USA. In proceedings [1944]. Online available at http://citeseer.ist.psu.edu/kovacs02two.html and http://www.cs.bris.ac.uk/Publications/Papers/1000647.pdf [accessed 2007-09-11].

[1181] Tim Kovacs, Xavier Llorà, Keiki Takadama, Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors. *Revised Selected Papers of the International Workshops on Learning Classifier Systems, IWLCS 2003-2005*, volume 4399/2007 of *Lecture Notes in Computer Science (LNCS), subseries Lecture Notes in Artificial Intelligence (LNAI)*, April 19, 2007. Springer Berlin/Heidelberg. ISBN: 978-3-54071-230-5. doi:10.1007/978-3-540-71231-2.

[1182] John Koza, Forrest H. Bennett III, David Andre, and Martin A. Keane. The design of analog circuits by means of genetic programming. In *Evolutionary Design by Computers*, chapter 16, pages 365–385. Morgan Kaufmann, 1999. In collection [181]. Online available at http://www.genetic-programming.com/jkpdf/edc1999.pdf [accessed 2007-10-03].

[1183] John R. Koza. *Non-Linear Genetic Algorithms for Solving Problems*. United States Patent and Trademark Office, 1988. United States Patent 4,935,877. Filed May 20, 1988. Issued June 19, 1990. Australian patent 611,350 issued september 21, 1991. Canadian patent 1,311,561 issued december 15, 1992.

[1184] John R. Koza. Hierarchical genetic algorithms operating on populations of computer programs. In *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence IJCAI-89*, pages 768–774, 1989. In proceedings [1950]. Partly online available at http://dli.iiit.ac.in/ijcai/IJCAI-89-VOL1/PDF/123.pdf [accessed 2008-05-29].

[1185] John R. Koza. A hierarchical approach to learning the boolean multiplexer function. In *Proceedings of the First Workshop on Foundations of Genetic Algorithms*, pages 171–191, 1990. In proceedings [1924]. Online available at http://citeseer.ist.psu.edu/koza91hierarchical.html and http://books.google.de/books?id=Df12yLrlUZYC [accessed 2008-05-29].

[1186] John R. Koza. Concept formation and decision tree induction using the genetic programming paradigm. In *Parallel Problem Solving from Nature - Proceedings of 1st Workshop, PPSN 1*, pages 124–128, 1990. In proceedings [1842]. Online available at http://citeseer.ist.psu.edu/61578.html [accessed 2008-05-29].

[1187] John R. Koza. Evolution and co-evolution of computer programs to control independent-acting agents. In *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, pages 366–375, 1990. In proceedings [1397]. Online available at http://www.genetic-programming.com/jkpdf/sab1990.pdf [accessed 2008-05-29].

[1188] John R. Koza. Genetic evolution and co-evolution of computer programs. In *Artificial Life II*, pages 603–629, 1990. Revised November 29, 1990. In proceedings [1248]. Online available at http://citeseer.ist.psu.edu/177879.html [accessed 2008-05-29].

[1189] John R. Koza. Genetically breeding populations of computer programs to solve problems in artificial intelligence. In *Proceedings of the Second International Conference on Tools for AI, Herndon, Virginia, USA*, pages 819–827. IEEE Computer Society Press, Los Alamitos, CA, USA, 6-9 1990. Online available at http://citeseer.ist.

psu.edu/koza90genetically.html and http://www.lania.mx/~ccoello/koza90.ps.gz [accessed 2007-09-09].

[1190] John R. Koza. The genetic programming paradigm: Genetically breeding populations of computer programs to solve problems. Technical Report STAN-CS-90-1314, Computer Science Department, Stanford University, Margaret Jacks Hall, Stanford, CA 94305, June 1990. Online available at http://citeseer.ist.psu.edu/koza90genetic.html [accessed 2008-05-29]. See also [1921].

[1191] John R. Koza. A genetic approach to econometric modeling. In Paul Bourgine and Bernard Walliser, editors, *Economics and Cognitive Science*, pages 57–75. Pergamon Press, Oxford, UK, 1991. Online available at http://www.genetic-programming.com/jkpdf/ecs1991cecoia1990.pdf [accessed 2008-05-29]. Submitted under the title "Discovering An Econometric Model By Genetic Breeding Of A Population Of Mathematical Functions" to CECOIA, June 2–6, Paris, France, 1990, The 2nd Conference Economics and Artificial Intelligence, online available at http://www.genetic-programming.com/jkpdf/cecoia1990.pdf and http://citeseer.ist.psu.edu/koza90discovering.html [accessed 2008-05-29].

[1192] John R. Koza. Evolving a computer program to generate random numbers using the genetic programming paradigm. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 37–44, 1991. In proceedings [170]. Online available at http://citeseer.ist.psu.edu/112986.html and http://www.genetic-programming.com/jkpdf/icga1991.pdf [accessed 2008-05-29].

[1193] John R. Koza. Concept formation and decision tree induction using the genetic programming paradigm. In *Parallel Problem Solving from Nature - Proceedings of 1st Workshop, PPSN 1*, pages 124–128, 1990. In proceedings [1842]. Online available at http://citeseer.ist.psu.edu/61578.html [accessed 2007-09-09].

[1194] John R. Koza. *Non-Linear Genetic Algorithms for Solving Problems by Finding a Fit Composition of Functions*. United States Patent and Trademark Office, August 1992. United States Patent 5,136,686. Filed March 28, 1990, Issued August 4, 1992.

[1195] John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. Complex Adaptive Systems. The MIT Press, July 4, 1994. ISBN: 0-2621-1189-6.

[1196] John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. A Bradford Book. The MIT Press, Cambridge, Massachusetts, USA, 1992. ISBN: 0-2621-1170-5, 978-0-26211-170-6. Partly online available at http://books.google.de/books?id=Bhtxo60BV0EC [accessed 2008-08-16]. 1992 first edition, 1993 second edition.

[1197] John R. Koza, editor. *Late Breaking Papers at the First Annual Conference Genetic Programming (GP-96)*, July 28–31, 1996, Stanford University, CA, USA. Stanford Bookstore. ISBN: 0-1820-1031-7.

[1198] John R. Koza, editor. *Late Breaking Papers at the Genetic Programming 1998 Conference*, July 22-25, 1998, University of Wisconsin, Madison, Wisconsin, USA. See also [1209].

[1199] John R. Koza and David Andre. Automatic discovery using genetic programming of an unknown-sized detector of protein motifs containing repeatedly-used subexpressions. In *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 89–97, 1995. In proceedings [1757]. Online available at http://www.genetic-programming.com/jkpdf/ml1995motif.pdf and http://citeseer.ist.psu.edu/83080.html [accessed 2007-10-03].

[1200] John R. Koza and David Andre. Classifying protein segments as transmembrane domains using architecture-altering operations in genetic programming. In *Advances in Genetic Programming 2*, chapter 8, pages 155–176. MIT Press, 1996. ISBN: 0-2620-1158-1. In collection [61]. Online available at http://www.genetic-programming.com/jkpdf/aigp2aatmjk1996.pdf and http://citeseer.ist.psu.edu/75759.html [accessed 2007-10-03].

[1201] John R. Koza and Martin A. Keane. Cart centering and broom balancing by genetically breeding populations of control strategy programs. In *Proceedings of International Joint Conference on Neural Networks*, volume I, pages 198–201, January 15–19, 1990, Washington, USA. Lawrence Erlbaum, Hillsdale, NJ, USA.

[1202] John R. Koza and Martin A. Keane. Genetic breeding of non-linear optimal control strategies for broom balancing. In *In Proceedings of the Ninth International Conference on Analysis and Optimization of Systems*, pages 47–56, June 1990, Antibes, France. Springer-Verlag, Berlin, Germany.

[1203] John R. Koza and James P. Rice. Genetic generation of both the weights and architecture for a neural network. In *International Joint Conference on Neural Networks, IJCNN-91*, volume II, pages 397–404. IEEE Computer Society Press, July 8–14, 1991, Washington State Convention and Trade Center, Seattle, WA, USA. ISBN: 0-7803-0164-1. doi:10.1109/IJCNN.1991.155366. Online available at http://citeseer.ist.psu.edu/koza91genetic.html [accessed 2008-05-29].

[1204] John R. Koza and James P. Rice. Automatic programming of robots using genetic programming. In *Proceedings of Tenth National Conference on Artificial Intelligence*, pages 194–201, 1992. In proceedings [1986]. Online available at http://citeseer.ist.psu.edu/koza92automatic.html and http://www.genetic-programming.com/jkpdf/aaai1992.pdf [accessed 2007-09-07].

[1205] John R. Koza, David Andre, Forrest H. Bennett III, and Martin A. Keane. Evolution of a low-distortion, low-bias 60 decibel op amp with good frequency generalization using genetic programming. In *Late Breaking Papers at the Genetic Programming 1996 Conference*, pages 94–100, 1996. In proceedings [1197]. Online available at http://www.genetic-programming.com/jkpdf/gp1996lbpamplifier.pdf and http://citeseer.ist.psu.edu/105348.html [accessed 2007-10-03].

[1206] John R. Koza, David Andre, Forrest H. Bennett III, and Martin A. Keane. Use of automatically defined functions and architecture-altering operations in automated circuit synthesis using genetic programming. In *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 132–149, 1996. In proceedings [1207]. Online available at http://citeseer.ist.psu.edu/119355.html and http://www.genetic-programming.com/jkpdf/gp1996adfaa.pdf [accessed 2007-10-03].

[1207] John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors. *Proceedings of the First Annual Conference Genetic Programming (GP-96)*, July 28–31, 1996, Stanford University, CA, USA. MIT Press.

[1208] John R. Koza, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max Garzon, Hitoshi Iba, and Rick L. Riolo, editors. *Genetic Programming 1997: Proceedings of the Second Annual Conference GP-97*, July 13-16, 1997, Stanford University, CA, USA. Morgan Kaufmann, San Francisco, CA, USA. See also [1956].

[1209] John R. Koza, Wolfgang Banzhaf, Kumar Chellapilla, Kalyanmoy Deb, Marco Dorigo, David B. Fogel, Max H. Garzon, David E. Goldberg, Hitoshi Iba, and Rick Riolo, editors. *Proceedings of the Third Annual Genetic Programming Conference (GP-98)*, July 22-25, 1998, University of Wisconsin, Madison, Wisconsin, USA. Morgan Kaufmann, Los Altos, CA, USA. ISBN: 1-5586-0548-7. see http://www.genetic-programming.org/gp98cfp.html [accessed 2007-09-01] and [1198].

[1210] John R. Koza, Forrest H. Bennett III, David Andre, and Martin A. Keane. *Genetic Programming III: Darwinian Invention and Problem Solving*. Morgan Kaufmann, first edition, May 1999. ISBN: 978-1-55860-543-5.

[1211] John R. Koza, Forrest H. Bennett III, David Andre, and Martin A. Keane. Automatic design of analog electrical circuits using genetic programming. In Hugh Cartwright, editor, *Intelligent Data Analysis in Science*, chapter 8, pages 172–200. Oxford University Press, Oxford, 2000.

[1212] John R. Koza, Martin A. Keane, Matthew J. Streeter, William Mydlowec, Jessen Yu, and Guido Lanza. *Genetic Programming IV: Routine Human-Competitive Ma-*

chine Intelligence. Genetic Programming. Springer, first edition, May 2005. ISBN: 978-0-38725-067-0. Partly online available at http://books.google.de/books?id=YQxWzAEnINIC [accessed 2008-09-03].

[1213] Dexter C. Kozen. *The Design and Analysis of Algorithms (Texts & Monographs in Computer Science)*. Springer New York, January 1992. ISBN: 978-0-38797-687-7.

[1214] Oliver Kramer. *Self-Adaptive Heuristics for Evolutionary Computation*, volume 147 of *Studies in Computational Intelligence*. Springer Berlin / Heidelberg, July 2008. ISBN: 978-3-54069-280-5. Series editor: Janusz Kacprzyk.

[1215] Natalio Krasnogor and Jim Smith. A tutorial for competent memetic algorithms: model, taxonomy, and design issues. *IEEE Transactions on Evolutionary Computation*, 9(5):474–488, October 2005. doi:10.1109/TEVC.2005.850260. Online available at http://www.cs.nott.ac.uk/~nxk/PAPERS/IEEE-TEC-lastVersion.pdf [accessed 2008-03-28].

[1216] R. Krishnapuram, A. Joshi, and L. Yi O. Nasraoui. Low-complexity fuzzy relational clustering algorithms for web mining. *IEEE-FS*, 9:595–607, August 2001. Online available at http://citeseer.ist.psu.edu/krishnapuram01lowcomplexity.html and http://de.scientificcommons.org/583343 [accessed 2007-08-11].

[1217] Guido Krüger. *Handbuch der Java-Programmierung*. Addison-Wesley, 4. aktualisierte edition, 2006. ISBN: 3-8273-2361-4, 3-8273-2447-5. Online available at http://www.javabuch.de/ [accessed 2007-07-03].

[1218] John Krumm, Gregory D. Abowd, Aruna Seneviratne, and Thomas Strang, editors. *UbiComp 2007: Ubiquitous Computing. Proceedings of the 9th International Conference*, volume 4717 of *Lecture Notes in Computer Science*, September 16–19, 2007, Innsbruck, Austria. Springer Berlin / Heidelberg. ISBN: 978-3-54074-852-6. doi:10.1007/978-3-540-74853-3.

[1219] Ben Kuipers and Bonnie Webber, editors. *Proceedings of the Fourteenth National Conference on Artificial Intelligence and Ninth Innovative Applications of Artificial Intelligence Conference, AAAI 97, IAAI 97*, July 27–31, 1997, Providence, Rhode Island, USA. AAAI Press/The MIT Press. ISBN: 0-2625-1095-2. See http://www.aaai.org/Conferences/AAAI/aaai97.php [accessed 2007-09-06] and http://www.aaai.org/Conferences/IAAI/iaai97.php [accessed 2007-09-06].

[1220] Anup Kumar, Rakesh M. Pathak, M. C. Gupta, and Yash P. Gupta. Genetic algorithm based approach for designing computer network topolgies. In *CSC'93: Proceedings of the 1993 ACM conference on Computer science*, pages 358–365, February 16–18, 1993, Indianapolis, Indiana, United States. ACM, New York, NY, USA. ISBN: 0-8979-1558-5. doi:10.1145/170791.170871. Online available at http://doi.acm.org/10.1145/170791.170871 [accessed 2008-08-01].

[1221] Sanjeev Kumar and Peter J. Bentley. Computational embryology: past, present and future. In *Advances in evolutionary computing: theory and applications*, pages 461–477. Springer, 2003. In collection [798]. Online available at http://www.cs.ucl.ac.uk/staff/P.Bentley/KUBECH1.pdf [accessed 2007-08-17].

[1222] Sourav Kundu, Kazuto Seto, and Shigeru Sugino. Genetic algorithm based design of passive elements for vibration control. In *Proceedings of 4th MOVIC Conference (Conference On Motion and Vibration Control)*, volume 3, pages 1183–1188, August 1998. Online available at http://citeseer.ist.psu.edu/349539.html and http://en.scientificcommons.org/362076 [accessed 2007-08-13].

[1223] Sourav Kundu, Kazuto Seto, and Shigeru Sugino. Genetic algorithm application to vibration control of tall flexible structures. In *Proceedings of The First IEEE International Workshop on Electronic Design, Test and Applications (DELTA'02)*, pages 333–337, January 29–31, 2002, Christchurch, New Zealand. IEEE Computer Society, Los Alamitos, CA, USA. ISBN: 0-7695-1453-7. doi:10.1109/DELTA.2002.994641.

[1224] Vera Kurkova, Nigel C. Steele, Roman Neruda, and Miroslav Karny, editors. *Proceedings of the 5th International Conference on Artificial Neural Networks and Ge-*

netic Algorithms, ICANNGA, 2001, Prague, Czech Republic. Springer Verlag, Berlin. ISBN: 978-3-21183-651-4.

[1225] Frank Kursawe. A variant of evolution strategies for vector optimization. In *Proceedings of the 1st Workshop on Parallel Problem Solving from Nature, PPSN I*, pages 193–197, 1990. In proceedings [1842]. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.47.8050 and http://www.lania.mx/~ccoello/kursawe.ps.gz [accessed 2009-04-23].

[1226] Ibrahim Kuscu. Evolving a generalised behavior: Artificial ant problem revisited. In *7th Annual Conference on Evolutionary Programming*, page 799 ff., 1998. ISBN: 3-5406-4891-7. In proceedings [1670].

[1227] Chung Min Kwan and C. S. Chang. Application of evolutionary algorithm on a transportation scheduling problem – the mass rapid transit. In *Proceedings of the 2005 IEEE Congress on Evolutionary Computation (CEC'2005)*, volume 2, pages 987–994, 2005. In proceedings [449]. Online available at http://www.lania.mx/~ccoello/EMOO/kwan05.pdf.gz [accessed 2007-08-27].

[1228] Raymond S. K. Kwan, Ann S. K.Kwan, and Anthony Wren. Driver scheduling using genetic algorithms with embedded combinatorial traits. In *Proceedings of 7th International Conference on Computer-Aided Scheduling of Public Transport*, volume 471, pages 81–102. Springer, Berlin, August 1997, Cambridge/Boston, MA, USA. ISBN: 3-5406-5775-4. Online available at http://www.citeulike.org/user/ilapla/article/1443013 and http://citeseer.ist.psu.edu/kwan97driver.html [accessed 2007-07-29].

[1229] John Kymissis, Clyde Kendall, Joseph Paradiso, and Neil Gershenfeld. Parasitic power harvesting in shoes. In *ISWC'98: Proceedings of the 2nd IEEE International Symposium on Wearable Computers*, page 132, 1998. IEEE Computer Society, Washington, DC, USA. ISBN: 0-8186-9074-7. Online available at http://www.media.mit.edu/resenv/papers.html and http://citeseer.ist.psu.edu/kymissis98parasitic.html [accessed 2007-08-01].

L

[1230] Jeffrey C. Lagraias, James A. Reeds, Margaret H. Wright, and Paul E. Wright. Convergence properties of the nelder-mead simplex method in low dimensions. *SIAM Journal on Optimization (SIOPT)*, 9(1):112–147, 1998. ISSN: 1052-6234 (print) / 1095-7189 (electronic). Online available at http://www.aoe.vt.edu/~cliff/aoe5244/nelder_mead_2.pdf [accessed 2008-06-14].

[1231] Chih-Chung Lai, Chuan-Kang Ting, and Ren-Song Ko. An effective genetic algorithm for improving wireless sensor network lifetime. In *GECCO'07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 2260–2260, 2007. doi:10.1145/1276958.1277395. In proceedings [2037]. Online available at http://doi.acm.org/10.1145/1276958.1277395 [accessed 2008-10-25]. Poster Session: Real-world applications: posters. See also [1231].

[1232] Chih-Chung Lai, Chuan-Kang Ting, and Ren-Song Ko. An effective genetic algorithm to improve wireless sensor network lifetime for large-scale surveillance applications. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2007*, pages 3531–3538, 2007. doi:10.1109/CEC.2007.4424930. In proceedings [1005]. See also [1231].

[1233] Jouni Lampinen and Ivan Zelinka. Mechanical engineering design optimization by differential evolution. In *New Ideas in Optimization*, pages 127–146. McGraw-Hill, 1999. In collection [448].

[1234] Jouni Lampinen and Ivan Zelinka. On stagnation of the differential evolution algorithm. In *Proceedings of MENDEL 2000, 6th International Mendel Conference on Soft Computing*, pages 76–83, 2000. In proceedings [1591]. Online avail-

able at http://citeseer.ist.psu.edu/317991.html and http://www.lut.fi/~jlampine/MEND2000.ps [accessed 2007-08-13].

[1235] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, July 1978. ISSN: 0001-0782. doi:10.1145/359545.359563. Operating Systems. Editor: R. Stockton Gaines. Online available at http://research.microsoft.com/users/lamport/pubs/time-clocks.pdf and http://portal.acm.org/citation.cfm?id=359563 [accessed 2007-08-01]. Also: Report CA-7603-2911, Massachusetts Comptuter Association, Wakefield, Massachusetts, USA, March 1976.

[1236] Edmung Landau. *Handbuch der Lehre von der Verteilung der Primzahlen*. B. G. Teubner, Leipzig, Germany, 1909. Reprinted by Chelsea, New York, 1953.

[1237] Christopher G. Langdon, editor. *Artificial Life: The Proceedings of an Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems*, volume 6 of *Santa Fe Institute Studies in the Science of Complexity*, September 1987, Los Alamos, New Mexico, USA. Addisson-Wesley Publishing Company, Reading, MA, USA / Redwood City, CA, USA. ISBN: 0-2010-9356-1, 978-0-20109-356-8. Published in January 1989.

[1238] William B. Langdon. *Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming!* Genetic Programming. Springer, April 30, 1998. ISBN: 0-7923-8135-1.

[1239] William B. Langdon. The halting probability in von Neumann architectures. Technical Report CSM-456, Computer Science, University of Essex, UK, July 2006. 2 page summary of [1243]. Online available at http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/langdon_2006_eurogp2p.html [accessed 2008-11-09].

[1240] William B. Langdon and Riccardo Poli. Better trained ants for genetic programming. Technical Report CSRP-98-12, University of Birmingham, School of Computer Science, April 2, 1998. Online available at http://citeseer.ist.psu.edu/105696.html and ftp://ftp.cs.bham.ac.uk/pub/tech-reports/1998/CSRP-98-12.ps.gz [accessed 2007-08-05].

[1241] William B. Langdon and Riccardo Poli. Fitness causes bloat: Mutation. In *Proceedings of the First European Workshop on Genetic Programming*, pages 37–48, 1998. In proceedings [141]. Online available at http://citeseer.ist.psu.edu/langdon98fitness.html and ftp://ftp.cwi.nl/pub/W.B.Langdon/papers/WBL.euro98_bloatm.ps.gz [accessed 2007-09-09].

[1242] William B. Langdon and Riccardo Poli. *Foundations of Genetic Programming*. Springer, Berlin, first edition, January 2002. ISBN: 3-5404-2451-2.

[1243] William B. Langdon and Riccardo Poli. The halting probability in von neumann architectures. In *9th European Conference on Genetic Programming*, pages 225–237, 2006. In proceedings [429]. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.61.7637 and http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/langdon_2006_eurogp.html [accessed 2008-11-09]. See also [1239].

[1244] William B. Langdon, Terry Soule, Riccardo Poli, and James A. Foster. The evolution of size and shape. In *Advances in Genetic Programming 3*, chapter 8, pages 163–190. MIT Press, 1999. In collection [1936]. Online available at http://www.cs.bham.ac.uk/~wbl/aigp3/ch08.ps.gz and http://citeseer.ist.psu.edu/langdon99evolution.html [accessed 2007-09-07].

[1245] William B. Langdon, Erick Cantú-Paz, Keith E. Mathias, Rajkumar Roy, David Davis, Riccardo Poli, Karthik Balakrishnan, Vasant Honavar, Günter Rudolph, Joachim Wegener, Larry Bull, Mitchell A. Potter, Alan C. Schultz, Julian F. Miller, Edmund K. Burke, and Natasa Jonoska, editors. *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, July 9–13, 2002, The Roosevelt

Hotel, 45th and Madison Avenue, New York, NY, USA. Morgan Kaufmann Publishers Inc. ISBN: 1-5586-0878-8. See also [331, 1572, 154].

[1246] Øyvind Langsrud. *Fisher's Exact Test*. langsrud.com, February 18, 2008. Online available at http://www.langsrud.com/fisher.htm [accessed 2008-12-08].

[1247] Christopher G. Langton, editor. *Artificial Life III: Proceedings of the Workshop on Artificial Life*, volume XVII of *Santa Fe Institute Studies in the Sciences of Complexity*, June 15–19, 1992, Santa Fe, New Mexico, USA. Addison Wesley Longman. ISBN: 0-2016-2494-X, 978-0-20162-494-6. Published in August 1993.

[1248] Christopher G. Langton, C.E. Taylor, D.J. Farmer, and S. Rasmussen, editors. *Artificial Life II: Proceedings of the Workshop on Artificial Life*, volume X of *Santa Fe Institute Studies in the Science of Complexity*, February 1990, Santa Fe, New Mexico, USA. Addison-Wesley. ISBN: 0-2015-2571-2, 978-0-20152-571-7. Published 1992, republished by Westview Press (March 28, 2003).

[1249] Pedro A. González Lanza and Jesús M. Zamarreño Cosme. A short-term temperature forecaster based on a state space neural network. *Engineering Applications of Artificial Intelligence*, 15(5):459–464, September 2002. doi:10.1016/S0952-1976(02)00089-1.

[1250] Pier Luca Lanzi and Alessandro Perrucci. Extending the representation of classifier conditions part ii: From messy coding to s-expressions. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 99)*, pages 345–352, 1999. In proceedings [142]. Online available at http://webspace.elet.polimi.it/lanzi/papers//lanzi1999GECCOgp.pdf [accessed 2007-08-01].

[1251] Pier Luca Lanzi and Rick L. Riolo. A roadmap to the last decade of learning classifier system research (from 1989 to 1999). In *Learning Classifier Systems: From Foundations to Applications*, page 33. Springer-Verlag Berlin/Heidelberg, 2000. In collection [1252].

[1252] Pier Luca Lanzi, Wolfgang Stolzmann, and Steward W. Wilson, editors. *Learning Classifier Systems, From Foundations to Applications*, volume 1813/2000 of *Lecture Notes in Artificial Intelligence, subseries of Lecture Notes in Computer Science (LNCS)*. Springer-Verlag Berlin/Heidelberg, London, UK, 2000. ISBN: 3-5406-7729-1. CODEN: LNCSD9.

[1253] Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors. *Advances in Learning Classifier Systems, Revised Papers of the Third International Workshop on Learning Classifier Systems (IWLCS-2000)*, volume 1996/2001 of *Lecture Notes in Computer Science (LNCS)*, September 15–16, 2000, Paris, France. Springer. ISBN: 3-5404-2437-7. A joint workshop of the sixth International Conference on Simulation of Adaptive Behaviour (SAB2000, see also [1398]) and the sixth International Conference on Parallel Problem Solving from Nature (PPSN VI, see also [1830]), published in 2001.

[1254] Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors. *Revised Papers of the 5th International Workshop on Learning Classifier Systems, IWLCS 2002*, volume 2661/2003 of *Lecture Notes in Computer Science (LNCS)*, September 7–8, 2002, Granada, Spain. Springer. ISBN: 3-5402-0544-6. Published in 2003.

[1255] Pier Luca Lanzi, Daniele Loiacono, Stewart W. Wilson, and David E. Goldberg. Generalization in the xcsf classifier system: Analysis, improvement, and extension. Technical Report 2005012, IlliGAL, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, March 2005. Online available at http://www.illigal.uiuc.edu/pub/papers/IlliGALs/2005012.pdf [accessed 2007-09-12]. See also [1256].

[1256] Pier Luca Lanzi, Daniele Loiacono, Stewart W. Wilson, and David E. Goldberg. Generalization in the xcsf classifier system: Analysis, improvement, and extension. *Evolutionary Computation Journal*, 2006. See also [1255].

[1257] Patrick LaRoche and A. Nur Zincir-Heywood. 802.11 network intrusion detection using genetic programming. In *GECCO'05: Proceedings of the 2005 workshops on Genetic and evolutionary computation*, pages 170–171, 2005. doi:10.1145/1102256.1102295. In proceedings [1766]. Online available at http://larocheonline.ca/~plaroche/papers/gecco-laroche-heywood.pdf and http://doi.acm.org/10.1145/1102256.1102295 [accessed 2008-06-16].

[1258] Christian W. G. Lasarczyk and Wolfgang Banzhaf. An algorithmic chemistry for genetic programming. In *Proceedings of the 8th European Conference on Genetic Programming*, pages 1–12, 2005. In proceedings [1116]. Online available at http://ls11-www.cs.uni-dortmund.de/downloads/papers/LaBa05.pdf and http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/eurogp_LasarczykB05.html [accessed 2008-04-30].

[1259] Christian W. G. Lasarczyk and Wolfgang Banzhaf. Total synthesis of algorithmic chemistries. In *GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation*, volume 2, pages 1635–1640, 2005. In proceedings [199]. Online available at http://www.cs.bham.ac.uk/~wbl/biblio/gecco2005/docs/p1635.pdf and http://doi.acm.org/10.1145/1068009.1068285 [accessed 2008-04-02].

[1260] Jörg Lässig, Karl Heinz Hoffmann, and Mihaela Enăchescu. Threshold selecting: Best possible probability distribution for crossover selection in genetic algorithms. In *Genetic and Evolutionary Computation Conference*, pages 2181–2185, 2008. In proceedings [1117].

[1261] Marco Laumanns, Eckart Zitzler, and Lothar Thiele. A unified model for multi-objective evolutionary algorithms with elitism. In *Proceedings of the 2000 Congress on Evolutionary Computation CEC00*, pages 46–53, 2000. In proceedings [1002]. Online available at http://citeseer.ist.psu.edu/laumanns00unified.html [accessed 2007-08-27].

[1262] Marco Laumanns, Lothar Thiele, Kalyanmoy Deb, and Eckart Zitzler. On the convergence and diversity-preservation properties of multi-objective evolutionary algorithms. Technical Report 108, Computer Engineering and Networks Laboratory (TIK), Department of Electrical Engineering, Swiss Federal Institute of Technology (ETH) Zurich and Kanpur Genetic Algorithms Laboratory (KanGAL), Department of Mechanical Engineering, Indian Institute of Technology Kanpur, Gloriastrasse 35, CH-8092 Zurich, Switzerland, 2001. Online available at http://e-collection.ethbib.ethz.ch/browse/alph/zitzlereckart.html and http://citeseer.ist.psu.edu/laumanns01convergence.html [accessed 2007-08-14].

[1263] E. L. Lawler, Jan Karel Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys. *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*. Wiley Series in Discrete Mathematics & Optimization. Wiley Interscience, Chichester, UK, September 1985. ISBN: 0-4719-0413-9, 978-0-47190-413-7.

[1264] Gregory F. Lawler. *Introduction to Stochastic Processes*. Chapman & Hall/CRC Press LLC, 2000 N.W. Corporate Blvd., Boca Raton, Florida 33431, USA, second edition, July 1995. ISBN: 0-4129-9511-5, 978-1-58488-651-8.

[1265] Steve Lawrence and C. Lee Giles. Overfitting and neural networks: Conjugate gradient and backpropagation. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN'00)*, volume 1, pages 1114–1119. IEEE Computer Society, July 24–27, 2000, Como, Italy. Online available at http://citeseer.ist.psu.edu/lawrence00overfitting.html [accessed 2007-09-13].

[1266] A. C. G. Verdugo Lazo and P. N. Rathie. On the entropy of continuous probability distributions. *IEEE Transactions on Information Theory*, 24(1):120–122, 1978.

[1267] Joshua Lederberg and Alexa T. McCray. 'ome sweet 'omics - a genealogical treasury of words. *The Scientist*, 15(7):8, April 2001. Online available at http://lhncbc.nlm.nih.gov/lhc/docs/published/2001/pub2001047.pdf [accessed 2007-08-06].

[1268] Jack Yiu-Bun Lee and P. C. Wong. The effect of function noise on gp efficiency. In *Progress in Evolutionary Computation*, pages 1–16, 1995. doi:10.1007/3-540-60154-6_43. In proceedings [2282].

[1269] S. Lee, S. Soak, K. Kim, H. Park, and M. Jeon. Statistical properties analysis of real world tournament selection in genetic algorithms. *Applied Intelligence*, 28(2):195–205, April 2008. ISSN: 0924-669X (Print) 1573-7497 (Online). doi:10.1007/s10489-007-0062-2. Online available at http://www.springerlink.com/content/amr3nq330x13u32t/fulltext.pdf [accessed 2008-03-20].

[1270] Katharina Anna Lehmann and Michael Kaufmann. Evolutionary algorithms for the self-organized evolution of networks. In *GECCO'05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 563–570, 2005. doi:10.1145/1068009.1068105. In proceedings [202]. Online available at http://doi.acm.org/10.1145/1068009.1068105 and http://www-pr.informatik.uni-tuebingen.de/mitarbeiter/katharinazweig/downloads/299-lehmann.pdf [accessed 2008-05-28]. Lehmann is now known as Zweig.

[1271] Derrick Henry Lehmer. Mathematical methods in large-scale computing units. *Math. Rev.*, 13(1):495, 1952.

[1272] Derrick Henry Lehmer. Mathematical methods in large-scale computing units. In *Proceedings of the 2nd Symposium on Large-Scale Digital Calculating Machinery, 1949*, pages 141–146. Harvard University Press, 1951, Cambridge, MA, USA.

[1273] Kwong Sak Leung, Kin Hong Lee, and Sin Man Cheang. Genetic parallel programming – evolving linear machine codes on a multiple-alu processor. In S. Yaacob, M. Nagarajan, and A. Chekima, editors, *Proceedings of International Conference on Artificial Intelligence in Engineering and Technology – ICAIET 2002*, pages 207–213, June 17–18, 2002, University of Malaysia Sabah, Kota Kinabalu, Sabah, Malaysia. ISBN: 9-8321-8892-X. See http://hdl.handle.net/2006/187099 [accessed 2008-09-17].

[1274] Joel R. Levin. What if there were no more bickering about statistical significance tests? *Research in the Schools*, 5(2):43–53, 1998. Online available at http://www.personal.psu.edu/users/d/m/dmr/sigtest/6mspdf.pdf [accessed 2008-08-15].

[1275] David Levine. Application of a hybrid genetic algorithm to airline crew scheduling. *Computers & Operations Research*, 23(6):547–558, 1996. ISSN: 0305-0548. Online available at http://www.citeulike.org/user/ilapla/article/1443054 and http://citeseer.ist.psu.edu/178394.html [accessed 2007-07-29].

[1276] Robert Michael Lewis, Virginia Joanne Torczon, and Michael W. Trosset. Direct search methods: Then and now. Technical Report NASA/CR-2000-210125 and ICASE Report No. 2000-26, Institute for Computer Applications in Science and Engineering, NASA Langley Research Center, Hampton, VA, USA (Operated by Universities Space Research Association), National Aeronautics and Space Administration Langley Research Center Hampton, Virginia 23681-2199, May 2000. Online available at http://historical.ncstrl.org/tr/pdf/icase/TR-2000-26.pdf [accessed 2008-06-14].

[1277] Junyi Li and R. Russell Rhinehart. Heuristic random optimization. *Computers and Chemical Engineering*, 22(3):427–444, February 1998. doi:10.1016/S0098-1354(97)00005-7. Received 10 August 1995; revised 28 March 1996.

[1278] Kangshun Li, Yuanxiang Li, Haifang Mo, and Zhangxin Chen. A new algorithm of evolving artificial neural networks via gene expression programming. *Journal of the Korean Society for Industrial and Applied Mathematics*, 9(2), 2005.

[1279] Rui Li, Michael T.M. Emmerich, Jeroen Eggermont, and Ernst G.P. Bovenkamp. Mixed-integer optimization of coronary vessel image analysis using evolution strategies. In *GECCO'06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1645–1652, 2006. doi:10.1145/1143997.1144268. In pro-

ceedings [352]. Online available at http://doi.acm.org/10.1145/1143997.1144268 [accessed 2007-08-27].

[1280] Xiaodong Li, Jürgen Branke, and Tim Blackwell. Particle swarm with speciation and adaptation in a dynamic environment. In *GECCO'06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 51–58, 2006. doi:10.1145/1143997.1144005. In proceedings [352]. Online available at http://doi.acm.org/10.1145/1143997.1144005 [accessed 2007-08-19].

[1281] Suiliong Liang, A. Nur Zincir-Heywood, and Malcom I. Heywood. The effect of routing under local information using a social insect metaphor. In *CEC'02: Proceedings of the Congress on Evolutionary Computation*, pages 1438–1443, 2002. IEEE Computer Society, Washington, DC, USA. In proceedings [703]. Online available at http://users.cs.dal.ca/~mheywood/X-files/Publications/01004454.pdf [accessed 2008-07-26].

[1282] Suiliong Liang, A. Nur Zincir-Heywood, and Malcolm I. Heywood. Adding more intelligence to the network routing problem: Antnet and ga-agents. *Applied Soft Computing*, 6(3):244–257, March 2006. ISSN: 1568-4946. doi:10.1016/j.asoc.2005.01.005.

[1283] Pierre Liardet, Pierre Collet, Cyril Fonlupt, Evelyne Lutton, and Marc Schoenauer, editors. *Proceedings of the 6th International Conference on Artificial Evolution, Evolution Artificielle, EA 2003*, volume 2936 of *Lecture Notes in Computer Science (LNCS)*, October 27–30, 2003, Marseilles, France. Springer Berlin/Heidelberg. ISBN: 3-5402-1523-9. Published in 2003.

[1284] Leonid Libkin and Limsoon Wong. Query languages for bags and aggregate functions. *Journal of Computer and System Sciences*, 55(2):241–272, October 1997. Online available at http://citeseer.ist.psu.edu/libkin97query.html and http://www.cs.toronto.edu/~libkin/papers/jcss97.ps.gz [accessed 2007-09-12].

[1285] Gunar E. Liepins and Michael D. Vose. Deceptiveness and genetic algorithm dynamics. In *Proceedings of the First Workshop on Foundations of Genetic Algorithms (FOGA)*, pages 36–50, 1991. In proceedings [1924]. Online available at http://www.osti.gov/bridge/servlets/purl/6445602-CfqU6M/ [accessed 2007-11-05].

[1286] Pedro Lima, editor. *Robotic Soccer*. I-Tech Education and Publishing, Vienna, Austria, December 2007. ISBN: 978-3-90261-321-9. Online available at http://s.i-techonline.com/Book/Robotic-Soccer/ISBN978-3-902613-21-9.html [accessed 2008-04-23].

[1287] Jianhua Lin. Adaptive image quantization based on learning classifier systems. In *DCC'95: Proceedings of the Conference on Data Compression*, page 477, March 28–30, 1995. IEEE Computer Society, Washington, DC, USA.

[1288] Shen Lin and Brian Wilson Kernighan. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21(2):498–516, March–April 1973.

[1289] Yung-Chien Lin, Kao-Shing Hwang, and Feng-Sheng Wang. Plant scheduling and planning using mixed-integer hybrid differential evolution with multiplier updating. In *Proceedings of the 2000 Congress on Evolutionary Computation*, volume 1, pages 593–600, 2000. In proceedings [1002].

[1290] Dennis Victor Lindley and W. F. Scott. *New Cambridge Statistical Tables*. Cambridge University Press, The Edinburgh Building, Shaftesbury Road, Cambridge, CB2 8RU, United Kingdom, 1995. ISBN: 0-5214-8485-5, 978-0-52148-485-5. Dewey number: 519.5/0212. Partly online available at http://books.google.de/books?id=CN-1fdPIgRsC [accessed 2008-08-18].

[1291] Charles X. Ling. Overfitting and generalization in learning discrete patterns. *Neurocomputing*, 8(3):341–347, August 1995. Optimization and Combinatorics, Part I-III, Online available at http://dx.doi.org/10.1016/0925-2312(95)00050-G and http://citeseer.ist.psu.edu/514876.html [accessed 2007-09-13].

[1292] Daniel A. Liotard. Algorithmic tools in the study of semiempirical potential surfaces. *International Journal of Quantum Chemistry*, 44(5):723–741, November 5,

1992. doi:10.1002/qua.560440505. Online available at http://www3.interscience.wiley.com/cgi-bin/fulltext/109573913/PDFSTART [accessed 2007-09-15].

[1293] Marc Lipsitch. Adaptation on rugged landscapes generated by iterated local interactions of neighboring genes. In *ICGA, International Conference on Genetic Algorithms*, pages 128–135, 1991. In proceedings [170].

[1294] Jun S. Liu. *Monte Carlo Strategies in Scientific Computing*. Springer, October 2002. ISBN: 0-3879-5230-6.

[1295] Pu Liu, Francis Lau, Michael J. Lewis, and Cho li Wang. A new asynchronous parallel evolutionary algorithm for function optimization. In *Proceedings of the 7th International Conference on Parallel Problem Solving from Nature*, pages 401–410, 2002. In proceedings [867]. Online available at [accessed 2008-04-06].

[1296] Yongguo Liu, Kefei Chen, Xiaofeng Liao, and Wei Zhang. A genetic clustering method for intrusion detection. *Pattern Recognition*, 37(5):927–942, May 2004. ISSN: 0031-3203. doi:10.1016/j.patcog.2003.09.011.

[1297] Xavier Llorà and Kumara Sastry. Fast rule matching for learning classifier systems via vector instructions. In *GECCO'06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1513–1520, 2006. doi:10.1145/1143997.1144244. In proceedings [352] and also [1298]. Online available at http://doi.acm.org/10.1145/1143997.1144244 [accessed 2007-09-12].

[1298] Xavier Llorà and Kumara Sastry. Fast rule matching for learning classifier systems via vector instructions. Technical Report 2006001, IlliGAL, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, January 2006. Online available at http://www.illigal.uiuc.edu/pub/papers/IlliGALs/2006001.pdf [accessed 2007-09-12]. See also [1297].

[1299] Fernando G. Lobo, Cláudio F. Lima, and Zbigniew Michalewicz, editors. *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*. Springer-Verlag Gmbh, Berlin, Germany, March 1, 2007. ISBN: 3-5406-9431-5, 978-3-54069-431-1.

[1300] José Lobo, John H. Miller, and Walter Fontana. Neutrality in technological landscapes. Santa fe working paper, Santa Fe Institute, 1399 Hyde Park Road, Santa Fe, New Mexico 87501, USA, January 26, 2004. Online available at http://fontana.med.harvard.edu/www/Documents/WF/Papers/tech.neut.pdf and http://citeseer.ist.psu.edu/lobo04neutrality.html [accessed 2008-06-06].

[1301] A. Geoff Lockett and Gerd Islei, editors. *Proceedings of the 8th International Conference on Multiple Criteria Decision Making: Improving Decision Making in Organizations (MCDM'1988)*, Lecture Notes in Economics and Mathematical Systems, 1988, Manchester, UK. Springer. ISBN: 978-0-38751-795-7. Published in December 1989.

[1302] Reinhard Lohmann. Structure evolution and incomplete induction. *Biological Cybernetics*, 69(4):319–326, August 1993. ISSN: 0340-1200 (Print) 1432-0770 (Online). doi:10.1007/BF00203128. Online available at http://www.springerlink.com/content/q2q668316m771073/fulltext.pdf [accessed 2007-08-12].

[1303] Reinhard Lohmann. Structure evolution and neural systems. In *Dynamic, Genetic, and Chaotic Programming: The Sixth-Generation*, pages 395–411. Wiley-Interscience, 1992. In collection [1921].

[1304] Jason D. Lohn and Silvano P. Colombano. A circuit representation technique for automated circuit design. *IEEE Transactions on Evolutionary Computation (IEEE-EC)*, 3(3):205, September 1999. Online available at http://ic.arc.nasa.gov/people/jlohn/bio.html and http://citeseer.ist.psu.edu/lohn99circuit.html [accessed 2007-08-07].

[1305] Jason D. Lohn, Silvano P. Colombano, Garyl L. Haith, and Dimitris Stassinopoulos. A parallel genetic algorithm for automated electronic circuit design. In *Proceedings of the Computational Aerosciences Workshop*, February 2000, NASA Ames Research

Center. Online available at http://ic.arc.nasa.gov/people/jlohn/bio.html and http://citeseer.ist.psu.edu/lohn00parallel.html [accessed 2007-08-07].

[1306] Jason D. Lohn, Garyl L. Haith, Silvano P. Colombano, and Dimitris Stassinopoulos. Towards evolving electronic circuits for autonomous space applications. In *Proceedings of the 2000 IEEE Aerospace Conference*, March 2000, Big Sky, MT. Online available at http://ic.arc.nasa.gov/people/jlohn/bio.html and http://citeseer.ist.psu.edu/336276.html [accessed 2007-08-07].

[1307] Jason D. Lohn, Gregory S. Hornby, and Derek Linden. An evolved antenna for deployment on nasa's space technology 5 mission. In *Genetic Programming Theory and Practice II*, 2004. In proceedings [1583]. Online available at http://ic.arc.nasa.gov/people/hornby/papers/lohn_gptp04.ps.gz [accessed 2007-08-17].

[1308] Heitor S. Lopes and Wagner R. Weinert. EGIPSYS: an enhanced gene expression programming approach for symbolic regression problems. *International Journal of Applied Mathematics and Computer Science*, 14(3), 2004. Special Issue: Evolutionary Computation. AMCS Centro Federal de Educacao Tecnologica do Parana / CPGEI Av. 7 de setembro, 3165, 80230-901 Curitiba (PR), Brazil. Online available at http://matwbn.icm.edu.pl/ksiazki/amc/amc14/amc1437.pdf [accessed 2007-08-23].

[1309] Ines Fernando Vega Lopez, Richard T. Snodgrass, and Bongki Moon. Spatiotemporal aggregate computation: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 17(2):271–286, February 2005. ISSN: 1041-4347. Online available at http://citeseer.ist.psu.edu/634034.html [accessed http://www.cs.arizona.edu/ bkmoon/papers/tkde-stagg.pdf]2007-09-12.

[1310] Richard Lowry. *Concepts and Applications of Inferential Statistics*. Vassar College, 124 Raymond Avenue, Poughkeepsie, New York 12604, USA, 2006. Companion site of [1313]. Online available at http://faculty.vassar.edu/lowry/webtext.html and http://dogsbody.psych.mun.ca/VassarStats/webtext.html [accessed 2008-12-08].

[1311] Richard Lowry. Vassarstats 2x2 contingency table tests. In *VassarStats: Web Site for Statistical Computing*, chapter 8 in [1310]. Vassar College, March 30, 2006. In collection [1313]. Online available at http://faculty.vassar.edu/lowry/tab2x2.html and http://dogsbody.psych.mun.ca/VassarStats/tab2x2.html [accessed 2008-12-08].

[1312] Richard Lowry. Vassarstats mann-whitney test. In *VassarStats: Web Site for Statistical Computing*, chapter 11a in [1310]. Vassar College, September 22, 2006. In collection [1313]. Online available at http://faculty.vassar.edu/lowry/utest.html and http://dogsbody.psych.mun.ca/VassarStats/utest.html [accessed 2008-12-07].

[1313] Richard Lowry. *VassarStats: Web Site for Statistical Computing*. Vassar College, 124 Raymond Avenue, Poughkeepsie, New York 12604, USA, 2006. See also [1310]. Online available at http://faculty.vassar.edu/lowry/VassarStats.html and http://dogsbody.psych.mun.ca/VassarStats/ [accessed 2008-12-08].

[1314] *METEOR-S: Semantic Web Services and Processes*. LSDIS Lab (Large Scale Distributed Information Systems), Department of Computer Science, University of Georgia, 2004. Online available at http://lsdis.cs.uga.edu/projects/meteor-s/ [accessed 2007-09-02].

[1315] Haiming Lu. *State-of-the-art Multiobjective Evolutionary Algorithms – Pareto Ranking, Density Estimation and Dynamic Population*. PhD thesis, Faculty of the Graduate College of the Oklahoma State University, Stillwater, Oklahoma, August 2002. Advisor: Gary G. Yen. Publication Number AAT 3080536. Online available at http://www.lania.mx/~ccoello/EMOO/thesis_lu.pdf.gz [accessed 2007-08-25].

[1316] Wei Lu and Issa Traore. Detecting new forms of network intrusions using genetic programming. *Computational Intelligence*, 20(3):475–494, August 2004. Online available at http://www.isot.ece.uvic.ca/publications/journals/coi-2004.pdf [accessed 2008-06-11].

[1317] Sean Luke. Evolving soccerbots: A retrospective. In *Proceedings of the 12th Annual Conference of the Japanese Society for Artificial Intelligence (JSAI)*, 1998. Online available at http://cs.gmu.edu/~sean/papers/robocupShort.pdf and http://citeseer.ist.psu.edu/43338.html [accessed 2007-09-09]. See also [1324, 1325].

[1318] Sean Luke. Code growth is not caused by introns. In *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference*, pages 228–235, 2000. In proceedings [2210]. Online available at http://citeseer.ist.psu.edu/300709.html and http://www.cs.gmu.edu/~sean/papers/intronpaper.pdf [accessed 2007-09-07].

[1319] Sean Luke and Liviu Panait. A survey and comparison of tree generation algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, pages 81–88, 2001. In proceedings [1937]. Online available at http://citeseer.ist.psu.edu/luke01survey.html and http://www.cs.gmu.edu/~sean/papers/treegenalgs.pdf [accessed 2007-07-28].

[1320] Sean Luke and Liviu Panait. A comparison of bloat control methods for genetic programming. *Evolutionary Computation*, 14(3):309–344, 2006. ISSN: 1063-6560.

[1321] Sean Luke and Lee Spector. Evolving graphs and networks with edge encoding: A preliminary report. In *Late Breaking Papers at the First Annual Conference Genetic Programming (GP-96)*, 1996. In proceedings [1197]. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.25.9484 and http://cs.gmu.edu/~sean/papers/graph-paper.pdf [accessed 2008-08-02].

[1322] Sean Luke and Lee Spector. A comparison of crossover and mutation in genetic programming. In *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 240–248, 1997. In proceedings [1208]. Online available at http://citeseer.ist.psu.edu/146629.html and http://hampshire.edu/lspector/pubs/comparison.pdf [accessed 2008-04-12].

[1323] Sean Luke and Lee Spector. Evolving teamwork and coordination with genetic programming. In *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 150–156, 1996. In proceedings [1207]. Online available at http://citeseer.ist.psu.edu/luke96evolving.html and http://www.ifi.uzh.ch/groups/ailab/people/nitschke/refs/Cooperation/cooperation.pdf [accessed 2008-07-28].

[1324] Sean Luke, Charles Hohn, Jonathan Farris, Gary Jackson, and James Hendler. Coevolving soccer softbot team coordination with genetic programming. In *Proceedings of the First International Workshop on RoboCup, at the International Joint Conference on Artificial Intelligence*, 1997. Online available at http://citeseer.ist.psu.edu/3898.html and http://www.cs.umd.edu/~seanl/papers/robocup.pdf [accessed 2007-09-09]. In proceedings [1455, 1456]. See also [1325].

[1325] Sean Luke, Charles Hohn, Jonathan Farris, Gary Jackson, and James Hendler. Coevolving soccer softbot team coordination with genetic programming. In Hiroaki Kitano, editor, *RoboCup-97: Robot Soccer World Cup I*, number 1395 in Lecture Notes in Computer Science (LNCS), subseries Lecture Notes in Artificial Intelligence (LNAI). Springer Berlin/Heidelberg, 1998. ISBN: 978-3-54064-473-6. Online available at hhttp://www.cs.umd.edu/~seanl/papers/robocupc.pdf [accessed 2007-09-09]. See also [1324].

[1326] Sean Luke, Conor Ryan, and Una-May O'Reilly, editors. *GECCO 2002: Graduate Student Workshop*, July 9, 2002, New York. AAAI, 445 Burgess Drive, Menlo Park, CA 94025. Part of [154].

[1327] Sean Luke, Liviu Panait, Gabriel Balan, Sean Paus, Zbigniew Skolicki, Jeff Bassett, Robert Hubley, and Alexander Chircop. Ecj: A java-based evolutionary computation research system, 2006. In 2007, ECJ reached version 16. For more information http://cs.gmu.edu/~eclab/projects/ecj/ [accessed 2007-07-10].

[1328] Eduard Lukschandl, Magnus Holmlund, Eric Moden, Mats Nordahl, and Peter Nordin. Induction of Java bytecode with genetic programming. In *Late Breaking Papers at the Genetic Programming 1998 Conference*, pages 135–142, 1998. In proceedings [1198].

[1329] Eduard Lukschandl, Henrik Borgvall, Lars Nohle, Mats G. Nordahl, and Peter Nordin. Evolving routing algorithms with the jbgp-system. In *EvoIASP'99/EuroEcTel'99: Proceedings of the First European Workshops on Evolutionary Image Analysis, Signal Processing and Telecommunications*, pages 193–202, 1999. doi:10.1007/10704703_16. In proceedings [1665].

[1330] Eduard Lukschandl, Henrik Borgvall, Lars Nohle, Mats G. Nordahl, and Peter Nordin. Distributed java bytecode genetic programming with telecom applications. In *European Conference on Genetic Programming, EuroGP*, 2000. doi:10.1007/b75085. In proceedings [1666].

[1331] Eduard Lukschandl, Peter Nordin, and Mats G. Nordahl. Using the java method evolver for load balancing in communication networks. In *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference*, pages 236–239, 2000. In proceedings [2210].

[1332] Changtong Luo. *Low dimensional simplex evolution algorithms and their applications*. PhD thesis, Jilin University, 2007. in Chinese.

[1333] Changtong Luo and Bo Yu. Low dimensional simplex evolution – a hybrid heuristic for global optimization. In *SNPD 2007: Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, volume 2, pages 470–474, July 30–August 1, 2007, Qingdao, Shandong, China. ISBN: 978-0-76952-909-7. INSPEC Accession Number: 9873104. doi:10.1109/SNPD.2007.58.

[1334] Mikulas Luptacik and Rudolf Vetschera, editors. *Proceedings of the First MCDM Winter Conference, 16th International Conference on Multiple Criteria Decision Making (MCDM'2002)*, February 12–22, 2002, Hotel Panhans, Semmering, Austria. See http://orgwww.bwl.univie.ac.at/mcdm2002 [accessed 2007-09-10].

[1335] Jay L. Lush. Progeny test and individual performance as indicators of an animal's breeding value. *Journal of Dairy Science*, 18(1):1–19, January 1935. Online available at http://jds.fass.org/cgi/reprint/18/1/1 [accessed 2007-11-27].

[1336] Evelyne Lutton, Pierre Collet, and Jean Louchet. Easea comparisons on test functions: Galib versus eo. In *Proceedings of the Fifth Conference on Artificial Evolution, Evolution Artificielle (EA-2001)*, pages 219–230, 2001. In proceedings [428]. Online available at http://fractales.inria.fr/evo-lab/EASEAComparisonFinal.ps.gz [accessed 2007-08-24].

M

[1337] Shingo Mabu, Kotaro Hirasawa, and Jinglu Hu. Genetic network programming with reinforcement learning and its performance evaluation. In *Late Breaking Papers at the 2004 Genetic and Evolutionary Computation Conference*, 2004. In proceedings [1113]. Online available at http://www.cs.bham.ac.uk/~wbl/biblio/gecco2004/LBP036.pdf [accessed 2008-06-16].

[1338] Penousal Machado, Jorge Tavares, Francisco B. Pereira, and Ernesto Jorge Fernandes Costa. Vehicle routing problem: Doing it the evolutionary way. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, page 690, 2002. In proceedings [1245]. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.16.8208 and http://osiris.tuwien.ac.at/~wgarn/VehicleRouting/GECCO02_VRPCoEvo.pdf [accessed 2008-10-27].

[1339] P. K. Mackeown. *Stochastic Simulation in Physics*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001. ISBN: 9-8130-8326-3.

[1340] Kenneth J. Mackin and Eiichiro Tazaki. Emergent agent communication in multiagent systems using automatically defined function genetic programming (adf-gp). In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics, IEEE SMC'99*, volume 5, pages 138–142, October 12–15, 1999, Tokyo, Japan. ISBN: 0-7803-5731-0, 978-0-78035-731-0. doi:10.1109/ICSMC.1999.815536.

[1341] Kenneth J. Mackin and Eiichiro Tazaki. Unsupervised training of multiobjective agent communication using genetic programming. In *KES*, pages 738–741, 2000. doi:10.1109/KES.2000.884152. In proceedings [963]. Online available at http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/Mackin00.html and http://www.lania.mx/~ccoello/EMOO/mackin00.pdf.gz [accessed 2008-07-28].

[1342] Kenneth J. Mackin and Eiichiro Tazaki. Multiagent communication combining genetic programming and pheromone communication. *Kybernetes*, 31(6):827–843, 2002. doi:10.1108/03684920210432808. Online available at http://www.emeraldinsight.com/10.1108/03684920210432808 [accessed 2008-07-28].

[1343] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. Berkeley, University of California Press, 1967.

[1344] Ole Lehrmann Madsen. On defining semantics by means of extended attribute grammars. In *Semantics-Directed Compiler Generation, Proceedings of a Workshop*, volume 94 of *Lecture Notes In Computer Science (LNCS)*, pages 259–299, January 14–18, 1980, Aarhus, Denmark. Springer-Verlag, London, UK. ISBN: 3-5401-0250-7.

[1345] Alexander Maedche, Steffen Staab, Claire Nedellec, and Eduard H. Hovy, editors. *IJCAI'2001 Workshop on Ontology Learning, Proceedings of the Second Workshop on Ontology Learning OL'2001*, volume 38 of *CEUR Workshop Proceedings*, August 4, 2001, Seattle, Washington, USA. CEUR-WS.org. Online available at http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-47/ [accessed 2008-04-01]. See also [1507]. Held in conjunction with the 17th International Conference on Artificial Intelligence IJCAI'2001.

[1346] Pattie Maes, Maja J. Mataric, Jean-Arcady Meyer, Jordan B. Pollack, and Stewart W. Wilson, editors. *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior: From animals to animats 4*, September 9-13, 1996, Cape Code, USA. MIT Press, Cambridge, MA, USA. ISBN: 0-2626-3178-4.

[1347] Anne E. Magurran. *Ecological Diversity and Its Measurement*. Princeton University Press, 41 William Street, Princeton, New Jersey 08540, USA, November 1988. ISBN: 978-0-69108-491-6, 0-6910-8485-8, 0-6910-8491-2.

[1348] Anne E. Magurran. Biological diversity. *Current Biology Magazine*, 15(4):R116–R118, February 22, 2005. doi:10.1016/j.cub.2005.02.006. Online available at http://dx.doi.org/10.1016/j.cub.2005.02.006 [accessed 2008-11-10].

[1349] A. Malhotra, J. H. Oliver, and W. Tu. Synthesis of spatially and intrinsically constrained curves using simulated annealing. *ASME Transactions, Journal of Mechanical Design*, 118:53–61, March 1996. DE-vol. 32-1 Alliances in Design Automation, vol. 1 ASME 1991 pp. 145–155.

[1350] Masud Ahmad Malik. Evolution of the high level programming languages: a critical perspective. *ACM SIGPLAN Notices*, 33(12):72–80, December 1998. ISSN: 0362-1340. Online available at http://doi.acm.org/10.1145/307824.307882 [accessed 2007-09-14].

[1351] David Malkin. *The Evolutionary Impact of Gradual Complexification on Complex Systems*. PhD thesis, University College London (UCL), Computer Science, 2008.

[1352] Bernard Manderick and Frans Moyson. The collective behaviour of ants: an example of self-organization in massive parallelism. In *Proceedings of AAAI Spring Symposium on Parallel Models of Intelligence*, 1988, Stanford, California.

[1353] Bernard Manderick and Piet Spiessens. Fine-grained parallel genetic algorithms. In *Proceedings of the third international conference on Genetic algorithms*, pages 428–433, 1989. In proceedings [1820].

[1354] Bernard Manderick, Mark de Weger, and Piet Spiessens. The genetic algorithm and the structure of the fitness landscape. In *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 143–150, 1991. In proceedings [170].

[1355] Vittorio Maniezzo, Luca Maria Gambardella, and Fabio de Luigi. Ant colony optimization. In *New Optimization Techniques in Engineering*, chapter 5, pages 101–117. Springer-Verlag, 2004. In collection [1580]. Online available at http://www.idsia.ch/~luca/aco2004.pdf and http://citeseer.ist.psu.edu/maniezzo04ant.html [accessed 2007-09-13].

[1356] Henry B. Mann and Donald R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics*, 18(1):50–60, March 1947. doi:10.1214/aoms/1177730491. Mathematical Reviews number (MathSciNet): MR22058, Zentralblatt MATH identifier: 0041.26103. Online available at http://projecteuclid.org/euclid.aoms/1177730491 [accessed 2008-10-24].

[1357] Reinhard Männer and Bernard Manderick, editors. *Proceedings of Parallel Problem Solving from Nature 2, PPSN II*, September 28–30, 1992, Brussels, Belgium. Elsevier. See http://ls11-www.informatik.uni-dortmund.de/PPSN/ppsn2/ppsn2.html [accessed 2007-09-05].

[1358] Steven Manos, Leon Poladian, Peter J. Bentley, and Maryanne Large. A genetic algorithm with a variable-length genotype and embryogeny for microstructured optical fibre design. In *GECCO'06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1721–1728, 2006. In proceedings [352]. Online available at http://portal.acm.org/citation.cfm?id=1144278 [accessed 2007-08-17].

[1359] Subbarao Kambhampati Manuela M. Veloso, editor. *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI) and the Seventeenth Innovative Applications of Artificial Intelligence Conference (IAA)*, July 9–13, 2005, Pittsburgh, Pennsylvania, USA. AAAI Press/The MIT Press. ISBN: 1-5773-5236-X. See http://www.aaai.org/Conferences/AAAI/aaai05.php [accessed 2007-09-06] and http://www.aaai.org/Conferences/IAAI/iaai06.php [accessed 2007-09-06].

[1360] Elena Marchiori and Adri G. Steenbeek. An evolutionary algorithm for large scale set covering problems with application to airline crew scheduling. In *Proceedings of Real-World Applications of Evolutionary Computing, EvoWorkshops 2000*, pages 367–381, 2000. In proceedings [320]. Online available at http://citeseer.ist.psu.edu/marchiori00evolutionary.html [accessed 2007-08-27].

[1361] M. H. Marghny and I. E. El-Semman. Extracting fuzzy classification rules with gene expression programming. *ICGST International Journal on Artificial Intelligence and Machine Learning, AIML*, (Special Issue on AI & Specific Applications), 2006. AIML 05 Conference, 19-21 December 2005, CICC, Cairo, Egypt. Online available at http://www.icgst.com/AIML05/papers/P1120535114.pdf [accessed 2007-08-23].

[1362] Eric A. Marks and Michael Bell. *Executive's Guide to Service oriented architecture (SOA): A Planning and Implementation Guide for Business and Technology*. John Wiley & Sons, Inc., Hoboken, NJ, April 2006. ISBN: 978-0-47003-614-3.

[1363] Max Martin, Eric Drucker, and Walter Don Potter. Ga, eo, and pso applied to the discrete network configuration problem. In *International Conference on Genetic and Evolutionary Methods, GEM'08*, 2008. In proceedings [81]. Online available at http://www.cs.uga.edu/~potter/CompIntell/GEM_Martin-et-al.pdf [accessed 2008-08-23].

[1364] Trevor P. Martin and Anca L. Ralescu, editors. *Fuzzy Logic in Artificial Intelligence, Towards Intelligent Systems, IJCAI'95 Selected Papers from Workshop*, volume 1188 of *Lecture Notes in Computer Science (LNCS)*, 1997, Montréal, Québec, Canada. Springer. ISBN: 3-5406-2474-0. See also [1453, 1454, 2190].

[1365] Worthy Neil Martin, Jens Lienig, and James P. Cohoon. Island (migration) models: Evolutionary algorithms based on punctuated equilibria. In *Handbook of Evolutionary Computation*, chapter 6.3. Oxford University Press, 1997. In collection [104]. Online available at http://www.cs.virginia.edu/papers/Island_Migration.pdf [accessed 2007-08-13].

[1366] H. Martinez-Alfaro and D. R. Flugrad. Collision-free path planning for mobile robots and/or agvs using simulated annealing. In *Systems, Man, and Cybernetics 1994. Proceedings of the IEEE International Conference on Humans, Information and Technology*, volume 1, pages 270–275. IEEE, October 1994. doi:10.1109/ICSMC.1994.399849.

[1367] H. Martinez-Alfaro, H. Valdez, and J. Ortega. Linkage synthesis of a four bar mechanism for n precision points using simulated annealing. In *Proceedings of the 1998 ASME Design Engineering Technical Conference*, page 7, September 13–16, 1998, Atlanta, USA.

[1368] Horacio Martínez-Alfaro and Manuel Valenzuela-Rendón. Using simulated annealing for paper cutting optimization. In *MICAI 2004: Advances in Artificial Intelligence*, pages 11–20, 2004. In proceedings [1442].

[1369] Yoshiyuki Matsumura, Kazuhiro Ohkura, and Kanji Ueda. Advantages of global discrete recombination in $(\mu/\mu, \lambda)$-evolution strategies. In *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, volume 2, pages 1848–1853, 2002. doi:10.1109/CEC.2002.1004524. In proceedings [703].

[1370] Friedemann Mattern. *Verteilte Basisalgorithmen*, volume 226 of *Informatik-Fachberichte (IFB)*. W. Bauer im Auftrag der Gesellschaft für Informatik (GI), Springer-Verlag GmbH, Berlin/Heidelberg, Germany, 1989. ISBN: 3-5405-1835-5, 0-3875-1835-5, 978-3-54051-835-8, 978-0-38751-835-0. LCCN: 90140009. Based on his dissertation at Prof. Dr. J. Nehmer's group / Sonderforschungsbereich "VLSI-Entwurf und Parallelität" at the computer science department of the University of Kaiserslautern.

[1371] J. Matyas. Random optimization. *Automation and Remote Control*, 26(2):244–251, 1965.

[1372] J. Matyas. Das zufällige optimierungsverfahren und seine konvergenz. In *Proceedings of the 5th International Analogue Computation Meeting*, pages 540–544. Presses Academiques Europeens, Brussels, 1968, Lausanne.

[1373] Ueli Maurer. Fast generation of prime numbers and secure public-key cryptographic parameters. *Journal of Cryptology*, 8(3):123–155, September 1995. ISSN: 0933-2790 (Print) 1432-1378 (Online). doi:10.1007/BF00202269. Online available at http://www.springerlink.com/content/u3710818146qq153/fulltext.pdf and http://citeseer.ist.psu.edu/186041.html [accessed 2007-09-15].

[1374] Giles Mayley. Guiding or hiding: explorations into the effects of learning on the rate of evolution. In *Fourth European Conference on Artificial Life, ECAL'97*, pages 135–144, 1997. In proceedings [975]. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.50.9241 [accessed 2008-09-10].

[1375] Ernst Mayr. *The Growth of Biological Thought*. Harvard University Press, 1982, Cambridge, MA, USA. ISBN: 0-6743-6446-5.

[1376] A. D. McAulay and Jae C. Oh. Image learning classifier system using genetic algorithms. In *Proceedings of IEEE National Aerospace Electronics Conference NAECON'89*, volume 2, pages 705–710, May 22–26, 1989. an IEEE conference.

[1377] John L. McCarthy. Recursive functions of symbolic expressions and their computation by machine, part i. *Communications of the ACM*, 3(4):184–195, April 1960. ISSN: 0001-0782. Online available at http://doi.acm.org/10.1145/367177.367199 and http://citeseer.ist.psu.edu/mccarthy60recursive.html [accessed 2007-09-15].

[1378] John L. McCarthy. History of lisp. In Richard L. Wexelblat, editor, *History of Programming Languages: Proceedings of the ACM SIGPLAN Conference*, pages 173–197. Academic Press, June 1978. Online available at http://citeseer.ist.psu.edu/

mccarthy78history.html [accessed 2007-09-15]. See also http://www-formal.stanford.edu/jmc/history/lisp/lisp.html [accessed 2007-09-15].

[1379] John L. McCarthy, Paul W. Abrahams, Daniel J. Edwards, Timothy P. Hari, and Michael L. Levin. *LISP 1.5 Programmer's Manual*. The MIT Press, August 1962. ISBN: 978-0-26213-011-0, 0-2621-3011-4. Second Edition, 15th Printing, Online available at http://www.softwarepreservation.org/projects/LISP/book/LISP%201.5%20Programmers%20Manual.pdf [accessed 2007-09-15]. See also [1379].

[1380] John Robert McDonnell, Robert G. Reynolds, and David B. Fogel, editors. *Proceedings of the 4th Annual Conference on Evolutionary Programming*, A Bradford Book, Complex Adaptive Systems, March 1995, San Diego, CA, USA. The MIT Press, Cambridge, Massachusetts. ISBN: 0-2621-3317-2.

[1381] Deborah L. McGuinness and George Ferguson, editors. *Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI) and the Sixteenth Conference on Innovative Applications of Artificial Intelligence (IAAI)*, July 25–29, 2004, San Jose, California, USA. AAAI Press/The MIT Press. ISBN: 0-2625-1183-5. See http://www.aaai.org/Conferences/AAAI/aaai04.php [accessed 2007-09-06] and http://www.aaai.org/Conferences/IAAI/iaai04.php [accessed 2007-09-06].

[1382] Robert Ian McKay, Xuan Hoai Nguyen, Peter Alexander Whigham, and Yin Shan. Grammars in genetic programming: A brief review. In L. Kang, Z. Cai, and Y. Yan, editors, *Progress in Intelligence Computation and Intelligence: Proceedings of the International Symposium on Intelligence, Computation and Applications*, pages 3–18. China University of Geosciences Press, April 2005. Online available at http://sc.snu.ac.kr/PAPERS/isica05.pdf [accessed 2007-08-15].

[1383] K. I. M. McKinnon. Convergence of the Nelder-Mead simplex method to a nonstationary point. *SIAM Journal on Optimization*, 9(1):148–158, 1999. ISSN: 1052-6234 (print) / 1095-7189 (electronic).

[1384] Nicholas Freitag McPhee and Justin Darwin Miller. Accurate replication in genetic programming. In *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pages 303–309, 1995. In proceedings [636]. Online available at http://www.mrs.umn.edu/~mcphee/Research/Accurate_replication.ps and http://citeseer.ist.psu.edu/mcphee95accurate.html [accessed 2007-09-07].

[1385] Nicholas Freitag McPhee and Ricardo Poli. Memory with memory: Soft assignment in genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1235–1242, 2008. In proceedings [1117].

[1386] Michael Meissner, Michael Schmuker, and Gisbert Schneider. Optimized particle swarm optimization (opso) and its application to artificial neural network training. *BMC Bioinformatics*, 7(125), 2006. doi:10.1186/1471-2105-7-125. Online available at http://www.biomedcentral.com/1471-2105/7/125 [accessed 2007-08-21].

[1387] Tommaso Melodia, Dario Pompili, and Ian F. Akyildiz. On the interdependence of distributed topology control and geographical routing in ad hoc and sensor networks. *Journal of Selected Areas in Communications*, 23(3):520–532, March 2005. doi:10.1109/JSAC.2004.842557.

[1388] Jerry M. Mendel, Takashi Omari, and Xin Yao, editors. *The First IEEE Symposium on Foundations of Computational Intelligence (FOCI'07)*, April 1–5, 2007, Hilton Hawaiian Village Beach Resort & Spa, Honolulu, Hawaii, USA. Institute of Electrical and Electronics Engineers (IEEE), Piscataway, N.J., USA. ISBN: 1-4244-0703-6, 978-1-42440-703-3. IEEE catalog number: 07EX1569. Sponsored by IEEE Computational Intelligence Society.

[1389] Roberto R. F. Mendes, Fabricio de B. Voznika, Alex A. Freitas, and Julio C. Nievola. Discovering fuzzy classification rules with genetic programming and co-evolution. In Luc de Raedt and Arno Siebes, editors, *Proceedings of 5th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'01)*, vol-

ume 2168 of *Lecture Notes in Computer Science (LNCS), subseries Lecture Nots in Artificial Intelligence (LNAI)*, pages 314–325. Springer Verlag Berlin/Heidelberg, September 3–5, 2001, Freiburg, Germany. ISBN: 978-3-54042-534-2. See also [1390]. Online available at http://www.cs.kent.ac.uk/people/staff/aaf/pub_papers.dir/PKDD-2001.ps [accessed 2007-09-09].

[1390] Roberto R. F. Mendes, Fabricio de B. Voznika, Alex A. Freitas, and Julio C. Nievola. Discovering fuzzy classification rules with genetic programming and co-evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, 2001. In proceedings [1937] and [1389]. Online available at http://citeseer.ist.psu.edu/521000.html [accessed 2007-09-09].

[1391] Rui Mendes and Arvind S. Mohais. Dynde: a differential evolution for dynamic optimization problems. In *Proceedings of 2005 IEEE Congress on Evolutionary Computation*, volume 3, pages 2808–2815, 2005. In proceedings [449]. Online available at http://www.di.uminho.pt/~rcm/publications/DynDE.pdf and http://en.scientificcommons.org/8412355 [accessed 2007-08-13].

[1392] Daniel Merkle, Martin Middendorf, and Hartmut Schmeck. Ant colony optimization for resource-constrained project scheduling. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2000)*, pages 893–900, 2000. In proceedings [2216]. Online available at http://citeseer.ist.psu.edu/merkle00ant.html and http://pacosy.informatik.uni-leipzig.de/pv/Personen/middendorf/Papers/ [accessed 2007-08-19].

[1393] Laurence D. Merkle and Frank Moore, editors. *Defense Applications of Computational Intelligence (DACI) Workshop 2008*, July 12, 2008, Renaissance Atlanta Hotel Downtown, 590 West Peachtree Street NW, Atlanta, Georgia 30308 USA. ACM Press, New York, NY, USA. ISBN: 978-1-60558-131-6. Part of GECCO 2008, see also [1117].

[1394] Marjan Mernik, Goran Gerlič, Viljem Žumer, and Barrett R. Bryant. Can a parser be generated from examples? In *SAC'03: Proceedings of the 2003 ACM symposium on Applied computing*, pages 1063–1067, 2003, Melbourne, Florida. ACM Press, New York, NY, USA. ISBN: 1-5811-3624-2. doi:10.1145/952532.952740. Online available at http://www.cis.uab.edu/softcom/GenParse/sac.pdf and http://doi.acm.org/10.1145/952532.952740 [accessed 2007-09-09].

[1395] Peter Merz and Bernd Freisleben. A comparison of memetic algorithms, tabu search, and ant colonies for the quadratic assignment problem. In *Proceedings of the Congress on Evolutionary Computation*, volume 3, pages 2063–2070, 1999. In proceedings [69], Online available at http://en.scientificcommons.org/204950 and http://citeseer.ist.psu.edu/merz99comparison.html [accessed 2008-03-28].

[1396] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, June 1953. doi:10.1063/1.1699114. Online available at http://link.aip.org/link/?JCPSA6/21/1087/1 [accessed 2008-03-26].

[1397] Jean-Arcady Meyer and Stewart W. Wilson, editors. *From Animals to Animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior*, September 24–28, 1990, Paris, France. The MIT Press, Cambridge, MA, USA. ISBN: 0-2626-3138-5. Published in 1991.

[1398] Jean-Arcady Meyer, A. Berthoz, D. Floreano, H. Roitblat, and Stewart W. Wilson, editors. *SAB2000: From Animals to Animats 6, Proceedings of the 6th International Conference on Simulation of Adpative Behavior*, August 2000, Paris, France. MIT Press, Cambridge, MA, USA. ISBN: 0-2626-3200-4.

[1399] Thomas Meyer, Daniel Schreckling, Christian Tschudin, and Lidia Yamamoto. Robustness to code and data deletion in autocatalytic quines. In Corrado Priami, Falko Dressler, Ozgur B. Akan, and Alioune Ngom, editors, *Transactions on Computational Systems Biology X*, volume 10/1974 of *Lecture Notes in Computer Science*

(LNCS), Subseries SL 8 – Lecture Notes in Bioinformatics, chapter 2, pages 20–40. Springer Berlin / Heidelberg, 2008. ISBN: 3-5409-2272-5, 978-3-54092-272-8. doi:10.1007/978-3-540-92273-5_2. Online available at http://cn.cs.unibas.ch/people/tm/doc/2008-tcsb.pdf [accessed 2008-11-09]. accepted for publication.

[1400] Silja Meyer-Nieberg and Hans-Georg Beyer. Self-adaptation in evolutionary algorithms. In *Parameter Setting in Evolutionary Algorithms*. Springer, 2007. In collection [1299]. Online available at http://citeseer.ist.psu.edu/741946.html [accessed 2007-07-28].

[1401] Juan C. Meza and M. L. Martinez. On the use of direct search methods for the molecular conformation problem. *Journal of Computational Chemistry*, 15(6):627–632, 1994. Online available at http://crd.lbl.gov/~meza/papers/jcc.pdf [accessed 2008-06-14].

[1402] M. Mezard, Giorgio Parisi, and M. A. Virasoro. *Spin Glass Theory and Beyond – An Introduction to the Replica Method and Its Applications*, volume 9 of *World Scientific Lecture Notes in Physics*. World Scientific Publishing Company, November 1987. ISBN: 978-9-97150-115-0, 9-9715-0115-5, 978-9-97150-116-7, 9-9715-0116-3.

[1403] Efrén Mezura-Montes and Carlos Artemio Coello Coello. Using the evolution strategies' self-adaptation mechanism and tournament selection for global optimization. *Intelligent Engineering Systems through Artificial Neural Networks (ANNIE'2003)*, 13:373–378, November 2003. Theoretical Developments in Computational Intelligence Award, 1st runner up. Online available at http://www.lania.mx/~emezura/documentos/annie03.pdf [accessed 2008-03-20].

[1404] Efrén Mezura-Montes, Jesús Velázquez-Reyes, and Carlos Artemio Coello Coello. A comparative study of differential evolution variants for global optimization. In *GECCO'06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 485–492, 2006. doi:10.1145/1143997.1144086. In proceedings [352]. Online available at http://portal.acm.org/citation.cfm?id=1144086 and http://delta.cs.cinvestav.mx/~ccoello/2006.html [accessed 2007-08-13].

[1405] Zbigniew Michalewicz. A step towards optimal topology of communication networks. In Vibeke Libby, editor, *Proceedings of Data Structures and Target Classification, the SPIE's International Symposium on Optical Engineering and Photonics in Aerospace Sensing*, volume 1470 of *SPIE*, pages 112–122. SPIE – The International Society for Optical Engineering., April 1–5 1991, Orlando, Florida. ISBN: 0-8194-0579-5, 978-0-81940-579-1.

[1406] Zbigniew Michalewicz. Genetic algorithms, numerical optimization, and constraints. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, pages 151–158., 1995. In proceedings [636]. Online available at http://www.cs.adelaide.edu.au/~zbyszek/Papers/p16.pdf and http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.14.1880 [accessed 2009-02-28].

[1407] Zbigniew Michalewicz and David B. Fogel. *How to Solve It: Modern Heuristics*. Springer, second, revised and extended edition, December 2004. ISBN: 978-3-54022-494-5.

[1408] Zbigniew Michalewicz and Girish Nazhiyath. Genocop iii: A co-evolutionary algorithm for numerical optimization with nonlinear constraints. In *Proceedings of the Second IEEE International Conference on Evolutionary Computation*, volume 2, pages 647–651, 1995. doi:10.1109/ICEC.1995.487460. In proceedings [1000]. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.14.2923 and http://www.cs.cinvestav.mx/~constraint/papers/p24.ps [accessed 2009-02-28].

[1409] Zbigniew Michalewicz and Robert G. Reynolds, editors. *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2008*, June1–6, 2007, Hong Kong Convention and Exhibition Centre, Hong Kong, China. IEEE Press, 445 Hoes Lane, P.O.

Box 1331, Piscataway, NJ 08855-1331, USA. ISBN: 978-1-42441-823-7. See http://www.wcci2008.org/ [accessed 2008-07-24].

[1410] Zbigniew Michalewicz and Marc Schoenauer. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1):1–32, Spring 1996. ISSN: 1063-6560, 1530-9304. doi:10.1162/evco.1996.4.1.1. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.13.9279 and http://www.cs.adelaide.edu.au/~zbyszek/Papers/p30.pdf [accessed 2009-02-28].

[1411] Zbigniew Michalewicz, J. David Schaffer, Hans-Paul Schwefel, David B. Fogel, and Hiroaki Kitano, editors. *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, June 27–29, 1994, Orlando, Florida, USA. IEEE Press, Piscataway, New Jersey. ISBN: 0-7803-1899-4. INSPEC Accession Number: 4812337. See http://ieeexplore.ieee.org/servlet/opac?punumber=1125 [accessed 2007-09-06].

[1412] Kaisa Miettinen, Marko M. Mäkelä, Pekka Neittaanmäki, and Jacques Periaux, editors. *Evolutionary Algorithms in Engineering and Computer Science*. John Wiley & Sons, Ltd, Chichester, UK, July 16, 1999. ISBN: 0-4719-9902-4, 978-0-47199-902-7.

[1413] Kaisa Miettinen, Marko M. Mäkelä, Pekka Neittaanmäki, and Jacques Periaux, editors. *European Short Course on Genetic Algorithms and Evolution Strategies, Proceedings of EUROGEN 1999*, May 30–June 3, 1999, University of Jyväskylä, Jyväskylä, Finland.

[1414] Mitsunori Miki, Tomoyuki Hiroyasu, and Jun'ya Wako. Adaptive temperature schedule determined by genetic algorithm for parallel simulated annealing. In *The 2003 Congress on Evolutionary Computation, CEC'03*, 2003. In proceedings [1803]. Online available at http://mikilab.doshisha.ac.jp/dia/research/person/wako/society/cec2003/cec03_wako.pdf [accessed 2007-09-15].

[1415] Brad L. Miller and David E. Goldberg. Genetic algorithms, tournament selection, and the effects of noise. IlliGAL Report 95006, Illinois Genetic Algorithms Laboratory, Department of General Engineering, University of Illinois, Urbana-Champaign, 117 Transportation Building, 104 South Mathews Avenue, Urbana, IL 61801, USA, July 1995. See also [1416]. Online available at http://citeseer.ist.psu.edu/86198.html and http://www.illigal.uiuc.edu/pub/papers/IlliGALs/95006.ps.Z [accessed 2007-07-28].

[1416] Brad L. Miller and David E. Goldberg. Genetic algorithms, selection schemes, and the varying effects of noise. *Evolutionary Computation*, 4(2):113–131, Summer 1996. ISSN: 1063-6560. doi:10.1162/evco.1996.4.2.113. See also [1415]. Online available at http://citeseer.ist.psu.edu/110556.html and http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=4225507A21F8083F025E28499309D1FB?doi=10.1.1.31.3449&rep=rep1&type=pdf [accessed 2008-07-21].

[1417] Brad L. Miller and Michael J. Shaw. Genetic algorithms with dynamic niche sharing for multimodal function optimization. IlliGAL Report 95010, Department of General Engineering, University of Illinois at Urbana-Champaign, 117 Transportation Building, 104 South Mathews Avenue, Urbana, IL 61801, USA, December 1, 1995. Online available at http://citeseer.comp.nus.edu.sg/5466.html [accessed http://www.illigal.uiuc.edu/pub/papers/IlliGALs/95010.ps.Z]2008-03-15.

[1418] Julian Francis Miller. An empirical study of the efficiency of learning boolean functions using a cartesian genetic programming approach. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1135–1142, 1999. In proceedings [142]. Online available at http://citeseer.ist.psu.edu/miller99empirical.html and http://www.elec.york.ac.uk/intsys/users/jfm7/gecco1999b-miller.pdf [accessed 2007-11-02].

[1419] Julian Francis Miller and Peter Thomson. Evolving digital electronic circuits for real-valued function generation using a genetic algorithm. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 863–868, 1998. In proceedings

[1209]. Online available at http://citeseer.ist.psu.edu/miller98evolving.html [accessed 2007-11-03].

[1420] Julian Francis Miller and Peter Thomson. Aspects of digital evolution: Evolvability and architecture. In *Parallel Problem Solving from Nature – PPSN V*, pages 927–936, 1998. In proceedings [624]. Online available at http://citeseer.ist.psu.edu/miller98aspects.html and http://www.elec.york.ac.uk/intsys/users/jfm7/ppsn1998-miller-thomson.pdf [accessed 2007-11-03].

[1421] Julian Francis Miller and Peter Thomson. Aspects of digital evolution: Geometry and learning. In *Evolvable Systems: From Biology to Hardware*, volume 1478/1998 of *Lecture Notes in Computer Science (LNCS)*, pages 25–35. Springer Berlin/Heidelberg, 1998. ISBN: 978-3-54064-954-0. doi:10.1007/BFb0057601. Online available at http://citeseer.ist.psu.edu/40293.html and http://www.elec.york.ac.uk/intsys/users/jfm7/ices1998-miller-thomson.pdf [accessed 2007-11-03].

[1422] Julian Francis Miller and Peter Thomson. Cartesian genetic programming. In *Genetic Programming, Proceedings of EuroGP'2000*, pages 121–132, 2000. In proceedings [1666]. Online available at http://citeseer.ist.psu.edu/424028.html and http://www.elec.york.ac.uk/intsys/users/jfm7/cgp-eurogp2000.pdf [accessed 2007-11-02].

[1423] Julian Francis Miller, Marco Tomassini, Pier Luca Lanzi, Conor Ryan, Andrea Tettamanzi, and William B. Langdon, editors. *Proceedings of the 4th European Conference on Genetic Programming, EuroGP 2001*, volume 2038/2001 of *Lecture Notes in Computer Science (LNCS)*, April 18-20, 2001, Lake Como, Italy. Springer Berlin/Heidelberg. ISBN: 3-5404-1899-7.

[1424] Hokey Min, Tomasz G. Smolinski, and Grzegorz M. Boratyn. A genetic algorithm-based data mining approach to profiling the adopters and non-adopters of e-purchasing. In W. W. Smari, editor, *Information Reuse and Integration, Third International Conference, IRI-2001*, pages 1–6. International Society for Computers and Their Applications (ISCA), 2001. Online available at http://citeseer.ist.psu.edu/min01genetic.html [accessed 2007-07-29].

[1425] Marvin Lee Minsky. Steps toward artificial intelligence. In *Proceedings of the IRE*, volume 49(1), pages 8–30, 1961. Reprint: Feigenbaum & Feldman, Computers and Thought, 1963, luger.

[1426] Marvin Lee Minsky. *Computation: Finite and Infinite Machines*. Prentica Hall Series on Automatic Computation. Prentice Hall, June 1967. ISBN: 978-0-13165-563-8.

[1427] Marvin Lee Minsky. *Berechnung: Endliche und Unendliche Maschinen*. Verlag Berliner Union GmbH, Stuttgart, 1971. ISBN: 978-3-40853-029-4.

[1428] Marvin Lee Minsky. Steps toward artificial intelligence. In Edward A. Feigenbaum and Julian Feldman, editors, *Computers & thought*, pages 406–450. MIT Press, Cambridge, MA, USA, 1995. ISBN: 0-2625-6092-5. Online available at http://web.media.mit.edu/~minsky/papers/steps.html [accessed 2007-08-05].

[1429] Daniele Miorandi, Paolo Dini, Eitan Altman, and Hisao Kameda. *WP 2.2 – Paradigm Applications and Mapping, D2.2.2 Framework for Distributed On-line Evolution of Protocols and Services*. BIOlogically inspired NETwork and Services (BIONETS) and Future and Emerging Technologies (FET) project of the EU, 2^{nd} edition, June 14, 2007. BIONETS/CN/wp2.2/v2.1. Editor: Lidia A. R. Yamamoto. Verification David Linner and Françoise Baude. Status: Final, Public. Online available at http://www.bionets.eu/docs/BIONETS_D2_2_2.pdf [accessed 2008-06-23].

[1430] Boris Mirkin. *Clustering for Data Mining: A Data Recovery Approach*. Computer Science and Data Analysis. Chapman & Hall/CRC, April 2005. ISBN: 978-1-58488-534-4.

[1431] Melanie Mitchell. *An Introduction to Genetic Algorithms*. Complex Adaptive Systems. The MIT Press, reprint edition, February 1998. ISBN: 0-2626-3185-7.

[1432] Melanie Mitchell, Stephanie Forrest, and John Henry Holland. The royal road for genetic algorithms: Fitness landscapes and GA performance. In *Towards a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, pages 245–254, 1991. In proceedings [2108]. Online available at http://citeseer.ist.psu.edu/mitchell91royal.html and http://web.cecs.pdx.edu/~mm/ecal92.pdf [accessed 2007-10-15].

[1433] Tom M. Mitchell. Generalization as search. In Bonnie Lynn Webber and Nils J. Nilsson, editors, *Readings in Artificial Intelligence*, pages 517–542. Tioga Pub. Co. Press / Morgan Kaufmann Publishers / Elsevier Science & Technology Books, 2nd edition, February 1981. ISBN: 0-9353-8203-8, 978-0-93538-203-7, 0-9346-1303-6, 978-0-93461-303-3. See also [1434].

[1434] Tom M. Mitchell. Generalization as search. *Artificial Intelligence*, 18(2):203–226, March 1982. ISSN: 0004-3702. doi:10.1016/0004-3702(82)90040-6. See also [1433].

[1435] Tom M. Mitchell and Reid G. Smith, editors. *Proceedings of the 7th National Conference on Artificial Intelligence, AAAI*, August 21–26, 1988, St. Paul, Minnesota, USA. AAAI Press/The MIT Press. ISBN: 0-2625-1055-3. See http://www.aaai.org/Conferences/AAAI/aaai88.php [accessed 2007-09-06].

[1436] Peter Mitic. Critical values for the wilcoxon signed rank statistic. *The Mathematica Journal*, 6(3):73–77, 1996.

[1437] Debasis Mitra, Fabio Romeo, and Alberto Sangiovanni-Vincentelli. Convergence and finite-time behavior of simulated annealing. In *Proceedings of the 24th IEEE Conference on Decision and Control*, volume 24, pages 761–767. IEEE Computer Society, December 1985. doi:10.1109/CDC.1985.268600.

[1438] Michael Mitzenmacher and Eli Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, January 2005. ISBN: 0-5218-3540-2.

[1439] Jun'ichi Mizoguchi, Hitoshi Hemmi, and Katsunori Shimohara. Production genetic algorithms for automated hardware design through an evolutionary process. In *IEEE World Congresson Computational Intelligence, Proceedings of the First IEEE Conference on Evolutionary Computation*, volume 2, pages 661–664. IEEE Press, June 27–29, 1994. doi:10.1109/ICEC.1994.349980.

[1440] Andreas F. Molisch. *Wireless Communications*. John Wiley & Sons / IEEE Press, November 18, 2005. ISBN: 978-0-47084-887-6, 0-4708-4887-1. Partly online available at http://books.google.com/books?id=2NNCHgAACAAJ [accessed 2008-07-27].

[1441] Nicolas Monmarché, El-Ghazali Talbi, Pierre Collet, Marc Schoenauer, and Evelyne Lutton, editors. *Revised Selected Papers of the 8th International Conference on Artificial Evolution, Evolution Artificielle, EA 2007*, Lecture Notes in Computer Science (LNCS), October 29–31, 2007, Tours, France. Springer Berlin/Heidelberg/New York. ISBN: 978-3-54079-304-5, 3-5407-9304-6. doi:10.1007/978-3-540-79305-2. See http://ea07.hant.li.univ-tours.fr/ [accessed 2007-09-09]. Published in 2008.

[1442] Raul Monroy, Gustavo Arroyo-Figueroa, Luis Enrique Sucar, and Juan Humberto Sossa Azuela, editors. *Proceedings of the MICAI 2004: Advances in Artificial Intelligence, Third Mexican International Conference on Artificial Intelligence*, volume 2972/2004 of *Lecture Notes in Computer Science (LNCS), Subseries Lecture Notes in Artificial Intelligence (LNAI)*, April 26–30, 2004, Mexico City, México. Springer Berlin / Heidelberg. ISBN: 3-5402-1459-3, 978-3-54021-459-5. doi:10.1007/b96521.

[1443] David Montana and Talib Hussain. Adaptive reconfiguration of data networks using genetic algorithms. *Applied Soft Computing*, 4(4):433–444, September 2004. doi:10.1016/j.asoc.2004.02.002. See also [1445]. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.95.1975 [accessed 2008-10-16].

[1444] David Montana and Jason Redi. Optimizing parameters of a mobile ad hoc network protocol with a genetic algorithm. In *GECCO'05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1993–1998, 2005.

doi:10.1145/1068009.1068342. Online available at http://vishnu.bbn.com/papers/erni.pdf and http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.86.5827 [accessed 2008-10-16]. In proceedings [202].

[1445] David Montana, Talib Hussain, and Tushar Saxena. Adaptive reconfiguration of data networks using genetic algorithms. In *GECCO'02: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1141–1149, 2002. In proceedings [1245]. See also [1443].

[1446] David J. Montana. Strongly typed genetic programming. BBN Technical Report #7866, Bolt Beranek and Newman Inc. (BBN), 70 Fawcett Street, Cambridge, MA 02138, dmontana@bbn.com, May 7, 1993. Online available at http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/ftp.io.com/papers/stgp.ps.Z [accessed 2007-10-04]. Superseded by [1447].

[1447] David J. Montana. Strongly typed genetic programming. BBN Technical Report #7866, Bolt Beranek and Newman Inc. (BBN), 70 Fawcett Street, Cambridge, MA 02138, dmontana@bbn.com, March 25, 1994. Online available at http://citeseer.ist.psu.edu/345471.html and http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/ftp.io.com/papers/stgp2.ps.Z [accessed 2007-10-04]. Superseds [1446] and is itself superseded by [1448].

[1448] David J. Montana. Strongly typed genetic programming. *Evolutionary Computation*, 3(2):199–230, 1995. Online available at http://vishnu.bbn.com/papers/stgp.pdf [accessed 2007-10-04] (November 20, 2002 edition).

[1449] *The Seventh Metaheuristics International Conference*, June 25–29, 2007, Montréal, Québec, Canada. See http://www.mic2007.ca/ [accessed 2007-09-12].

[1450] Federico Morán, Alvaro Moreno, Juan J. Merelo Guervós, and Pablo Chacón, editors. *Advances in Artificial Life, Proceedings of the Third European Conference on Artificial Life*, volume 929 of *Lecture Notes in Computer Science (LNCS)*, June 4–6, 1995, Granada, Spain. Springer. ISBN: 3-5405-9496-5.

[1451] Conwy Lloyd Morgan. On modification and variation. *Science*, 4(99): 733–740, November 20, 1896. ISSN: 0036-8075 (print), 1095-9203 (online). doi:10.1126/science.4.99.733. See also [1452].

[1452] Conwy Lloyd Morgan. On modification and variation. In *Adaptive individuals in evolving populations: models and algorithms*, pages 81–90. Addison-Wesley Longman Publishing Co., Inc., 1996. In collection [171]. See also [1451].

[1453] *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, volume 1, August 20–25, 1995, Montréal, Québec, Canada. Morgan Kaufmann, San Francisco, CA, USA. ISBN: 1-5586-0363-8. Online available at http://dli.iiit.ac.in/ijcai/IJCAI-95-VOLt.htm [accessed 2008-04-01]. See also [1454, 2190, 1364].

[1454] *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, volume 2, August 20–25, 1995, Montréal, Québec, Canada. Morgan Kaufmann, San Francisco, CA, USA. ISBN: 1-5586-0363-8. Online available at http://dli.iiit.ac.in/ijcai/IJCAI-95-VOL2/CONTENT/content.htm [accessed 2008-04-01]. See also [1453, 2190, 1364].

[1455] *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI 97*, volume 1, August 23–29, 1997, Nagoya, Japan. Morgan Kaufmann, San Francisco, CA, USA. Online available at http://dli.iiit.ac.in/ijcai/IJCAI-97-VOL1/CONTENT/content.htm [accessed 2008-04-01]. See also [1456, 1703].

[1456] *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, IJCAI 97*, volume 2, August 23–29, 1997, Nagoya, Japan. Morgan Kaufmann, San Francisco, CA, USA. Online available at http://dli.iiit.ac.in/ijcai/IJCAI-97-VOL2/CONTENT/content.htm [accessed 2008-04-01]. See also [1455, 1703].

[1457] Naoki Mori, Hajime Kita, and Yoshikazu Nishikawa. Adaptation to a changing environment by means of the thermodynamical genetic algorithm. In

Parallel Problem Solving from Nature – PPSN IV, pages 513–522, 1996. doi:10.1007/3-540-61723-X_1015. In proceedings [2118].

[1458] Naoki Mori, Seiji Imanishi, Hajime Kita, and Yoshikazu Nishikawa. Adaptation to changing environments by means of the memory based thermodynamical genetic algorithm. In *Proceedings of The Seventh International Conference on Genetic Algorithms, ICGA'97*, pages 299–306, 1997. In proceedings [98].

[1459] Naoki Mori, Hajime Kita, and Yoshikazu Nishikawa. Adaptation to a changing environment by means of the feedback thermodynamical genetic algorithm. In *Parallel Problem Solving from Nature – PPSN V*, pages 149–158, 1998. doi:10.1007/BFb0056858. In proceedings [624].

[1460] David E. Moriarty, Alan C. Schultz, and John J. Grefenstette. Evolutionary algorithms for reinforcement learning. *Journal of Artificial Intelligence Research*, 11:241–276, September 1999. Online available at http://citeseer.ist.psu.edu/moriarty99evolutionary.html and http://www.jair.org/media/613/live-613-1809-jair.pdf [accessed 2007-09-12].

[1461] Ronald Morrison. *On the Development of Algol*. PhD thesis, Department of Computational Science, University of St Andrews, December 1979. Online available at http://www-old.cs.st-andrews.ac.uk/research/publications/Mor79a.php [accessed 2007-07-10].

[1462] Ronald W. Morrison and Kenneth Alan De Jong. Measurement of population diversity. In *Artificial Evolution*, pages 1047–1074, 2001. doi:10.1007/3-540-46033-0_3. In proceedings [428]. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.108.1903 [accessed 2008-11-10].

[1463] Ronald Walter Morrison. *Designing Evolutionary Algorithms for Dynamic Environments*. PhD thesis, George Mason University, USA, 2002. Advisor: Kenneth Alan De Jong. See also [1464].

[1464] Ronald Walter Morrison. *Designing Evolutionary Algorithms for Dynamic Environments*, volume 24(1) of *Natural Computing*. Springer, Berlin, June 2004. ISBN: 978-3-54021-231-7. See also [1463].

[1465] Ronald Walter Morrison and Kenneth Alan De Jong. A test problem generator for non-stationary environments. In *Proceedings of the 1999 Congress on Evolutionary Computation, CEC 99*, volume 3, pages 2047–2053, 1999. doi:10.1109/CEC.1999.785526. In proceedings [69].

[1466] Joel N. Morse. Reducing the size of the nondominated set: Pruning by clustering. *Computers & Operations Research*, 7(1–2):55–66, 1980. doi:10.1016/0305-0548(80)90014-3.

[1467] Joel N. Morse, editor. *Proceedings of the 4th International Conference on Multiple Criteria Decision Making: Organizations, Multiple Agents With Multiple Criteria (MCDM'1980)*, Lecture Notes in Economics and Mathematical Systems, 1980, Newark, Delaware, USA. Springer. ISBN: 978-0-38710-821-6. Published in July 1981.

[1468] Pablo Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical Report C3P 826, Caltech Concurrent Computation Program 158-79, California Institute of Technology, Pasadena, CA 91125, USA, Pasadena, CA, 1989. Online available at http://www.densis.fee.unicamp.br/~moscato/papers/bigone.ps [accessed 2007-08-18].

[1469] Pablo Moscato. Memetic algorithms. In *Handbook of Applied Optimization*, chapter 3.6.4, pages 157–167. Oxford University Press, 2002. In collection [1613].

[1470] Pablo Moscato and Carlos Cotta. A gentle introduction to memetic algorithms. In *Handbook of Metaheuristics*, chapter 5, pages 105–144. Kluwer Academic Publishers, 2003. doi:10.1007/0-306-48056-5_5. In collection [813]. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.77.5300 [accessed 2008-09-11].

[1471] Sanaz Mostaghim. *Multi-objective Evolutionary Algorithms: Data structures, Convergence and, Diversity*. PhD thesis, Fakultät für Elektrotechnik, Informatik und

Mathematik, Universität Paderborn, Deutschland (Germany), Sharker Verlag, 2004. Online available at http://deposit.ddb.de/cgi-bin/dokserv?idn=974405604 and http://ubdata.uni-paderborn.de/ediss/14/2004/mostaghi/disserta.pdf [accessed 2007-08-17].

[1472] Jack Mostow and Chuck Rich, editors. *Proceedings of the Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence Conference, AAAI 98, IAAI 98*, July 26–30, 1998, Madison, Wisconsin, USA. AAAI Press/The MIT Press. ISBN: 0-2625-1098-7. See http://www.aaai.org/Conferences/AAAI/aaai98.php [accessed 2007-09-06] and http://www.aaai.org/Conferences/IAAI/iaai98.php [accessed 2007-09-06].

[1473] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge International Series on Parallel Computation. Cambridge University Press, August 1995. ISBN: 978-0-52147-465-8.

[1474] Rajeev Motwani and Prabhakar Raghavan. Randomized algorithms. *ACM Comput. Surv.*, 28(1):33–37, 1996. ISSN: 0360-0300.

[1475] Gero Mühl. Lecture notes "verteilte systeme", lesson "einführung und motivation, 2006. Online available at http://kbs.cs.tu-berlin.de/teaching/sose2006/vs/folien/02.pdf [accessed 2008-02-09].

[1476] Heinz Mühlenbein. Parallel genetic algorithms, population genetics and combinatorial optimization. In *Parallelism, Learning, Evolution*, pages 398–406, 1989. doi:10.1007/3-540-55027-5_23. In proceedings [164]. See also [1477].

[1477] Heinz Mühlenbein. Parallel genetic algorithms, population genetics and combinatorial optimization. In *Proceedings of the third international conference on Genetic algorithms*, pages 416–421, 1989. In proceedings [1820]. See also [1476].

[1478] Heinz Mühlenbein. Evolution in time and space – the parallel genetic algorithm. In *Foundations of Genetic Algorithms 1*, pages 316–337, 1990. In proceedings [1924]. Online available at http://muehlenbein.org/parallel.PDF and http://citeseer.ist.psu.edu/11201.html [accessed 2007-11-30].

[1479] Heinz Mühlenbein. How genetic algorithms really work – i. mutation and hillclimbing. In *Parallel Problem Solving from Nature 2*, pages 15–26, 1992. In proceedings [1357]. Online available at http://www.iais.fraunhofer.de/fileadmin/images/pics/Abteilungen/INA/muehlenbein/PDFs/technical/how-genetic-algorithms.pdf [accessed 2008-09-12].

[1480] Heinz Mühlenbein. Parallel genetic algorithms in combinatorial optimization. In Osman Balci, Ramesh Sharda, and Stavros A. Zenios, editors, *Selected papers from the Conference Computer Science and Operations Research: New Developments in Their Interfaces*, pages 441–456, January 8–10, 1992, Williamsburg, Virginia. Pergamon Press, Oxford. ISBN: 0-0804-0806-0, 978-0-08040-806-4. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.78.990 [accessed 2008-06-23].

[1481] Heinz Mühlenbein and Dirk Schlierkamp-Voosen. Predictive models for the breeder genetic algorithm i: Continuous parameter optimization. *Evolutionary Computations*, 1(1):25–49, Spring 1993. ISSN: 1063-6560, 2049097. Online available at http://citeseer.ist.psu.edu/mtihlenbein93predictive.html [accessed 2008-07-21].

[1482] Srinivas Mukkamala, Andrew H. Sung, and Ajith Abraham. Modeling intrusion detection systems using linear genetic programming approach. In Robert Orchard, Chunsheng Yang, and Moonis Ali, editors, *Proceedings of the 17th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE 2004*, volume 3029/2004 of *Lecture Notes in Computer Science (LNCS)*, pages 633–642. Springer Springer Verlag Inc, May 17–20, 2004, Ottawa, Canada. ISBN: 3-5402-2007-0, 978-3-54022-007-7. doi:10.1007/b97304. Online available at http://www.rmltech.com/LGP%20Based%20IDS.pdf [accessed 2008-06-17].

[1483] Markus Müller-Olm, David A. Schmidt, and Bernhard Steffen. Model checking: A tutorial introduction. In Agostino Cortesi and Gilberto Filé, editors, *Proceed-*

ings of the 6th Static Analysis Symposium SAS'99, volume 1694 of *Lecture Notes in Computer Science (LNCS)*, pages 330–354. Springer-Verlag, September 22–24 1999, Venice, Italy. ISBN: 0-5436-6459-9, 978-3-54066-459-8. doi:10.1007/3-540-48294-6_22. Online available at http://people.cis.ksu.edu/~schmidt/papers/sas99.ps.gz [accessed 2008-10-02].

[1484] Masaharu Munetomo. Designing genetic algorithms for adaptive routing algorithms in the internet. In *Evolutionary Telecommunications: Past, Present, and Future*, pages 215–216, 1999. In proceedings [154]. Online available at http://citeseer.ist.psu.edu/198441.html and http://uk.geocities.com/markcsinclair/ps/etppf_mun.ps.gz [accessed 2008-06-25]. Presentation online available at http://uk.geocities.com/markcsinclair/ps/etppf_mun_slides.ps.gz and http://assam.cims.hokudai.ac.jp/~munetomo/documents/gecco99-ws.ppt [accessed 2008-06-25].

[1485] Masaharu Munetomo and David E. Goldberg. Linkage identification by non-monotonicity detection for overlapping functions. IlliGAL Report 99005, Illinois Genetic Algorithms Laboratory (IlliGAL), University of Illinois at Urbana-Champaign, 117 Transportation Building, 104 S. Mathews Avenue, Urbana, IL 61801, USA, January 1999. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.33.9322 and http://www.illigal.uiuc.edu/pub/papers/IlliGALs/99005.ps.Z [accessed 2008-10-17]. See also [1486].

[1486] Masaharu Munetomo and David E. Goldberg. Linkage identification by non-monotonicity detection for overlapping functions. *Evolutionary Computation*, 7(4): 377–398, Winter 1999. ISSN: 1063-6560, 1530-9304. doi:10.1162/evco.1999.7.4.377. See also [1485].

[1487] Masaharu Munetomo, Yoshiaki Takai, and Yoshiharu Sato. An adaptive network routing algorithm employing path genetic operators. In *Proceedings of the Seventh International Conference on Genetic Algorithms*, pages 643–649, 1997. In proceedings [98].

[1488] Masaharu Munetomo, Yoshiaki Takai, and Yoshiharu Sato. An adaptive routing algorithm with load balancing by a genetic algorithm. *Transactions of the Information Processing Society of Japan (IPSJ)*, 39(2):219–227, 1998. in Japanese.

[1489] Masaharu Munetomo, Yoshiaki Takai, and Yoshiharu Sato. A migration scheme for the genetic adaptive routing algorithm. In *Proceedings of the 1998 IEEE Conference on Systems, Man, and Cybernetics*, volume 3, pages 2774–2779. IEEE Press, October 11–14, 1998, San Diego, U.S.A. doi:10.1109/ICSMC.1998.725081.

[1490] Tadahiko Murata and Takashi Nakamura. Developing cooperation of multiple agents using genetic network programming with automatically defined groups. In *Late Breaking Papers at the 2004 Genetic and Evolutionary Computation Conference*, 2004. In proceedings [1113].

[1491] Tadahiko Murata and Takashi Nakamura. Multi-agent cooperation using genetic network programming with automatically defined groups. In *Genetic and Evolutionary Computation – GECCO 2004*, pages 712–714, 2004. In proceedings [545].

[1492] Tadahiko Murata and Takashi Nakamura. Genetic network programming with automatically defined groups for assigning proper roles to multiple agents. In *GECCO'05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1705–1712, 2005. doi:10.1145/1068009.1068294. In proceedings [202]. Online available at http://www.cs.bham.ac.uk/~wbl/biblio/gecco2005/docs/p1705.pdf and http://doi.acm.org/10.1145/1068009.1068294 [accessed 2008-06-16].

[1493] M. Musil, M. J. Wilmut, and N. R. Chapman. A hybrid simplex genetic algorithm for estimating geoacoustic parameters using matched-field inversion. *IEEE Journal of Oceanic Engineering*, 24(3):358–369, July 1999. ISSN: 0364-9059. CODEN: IJOEDY. INSPEC Accession Number: 6322331. doi:10.1109/48.775297.

[1494] Nitin Muttil and Shie-Yui Liong. Superior exploration–exploitation balance in shuffled complex evolution. *Journal of Hydraulic Engineering*, 130(12):1202–1205, December 2004.

[1495] John Mylopoulos and Raymond Reiter, editors. *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, volume 1, August 24–30, 1991, Sydney, Australia. Morgan Kaufmann, San Francisco, CA, USA. ISBN: 1-5586-0160-0. Online available at http://dli.iiit.ac.in/ijcai/IJCAI-91-VOL1/CONTENT/content.htm [accessed 2008-04-01]. See also [1496, 1608, 596].

[1496] John Mylopoulos and Raymond Reiter, editors. *Proceedings of the 12th International Joint Conference on Artificial Intelligence*, volume 2, August 24–30, 1991, Sydney, Australia. Morgan Kaufmann, San Francisco, CA, USA. ISBN: 1-5586-0160-0. Online available at http://dli.iiit.ac.in/ijcai/IJCAI-91-VOL2/CONTENT/content.htm [accessed 2008-04-01]. See also [1495, 1608, 596].

N

[1497] Tadashi Nakano and Tatsuya Suda. Adaptive and evolvable network services. In *Genetic and Evolutionary Computation – GECCO 2004*, pages 151–162, 2004. In proceedings [544]. Online available at http://www.cs.york.ac.uk/rts/docs/GECCO_2004/Conferenceroceedings/papers/3102/31020151.pdf [accessed 2008-08-29].

[1498] Tadashi Nakano and Tatsuya Suda. Self-organizing network services with evolutionary adaptation. *IEEE Transactions on Neural Networks*, 16(5):1269–1278, September 2005. ISSN: 1045-9227. doi:10.1109/TNN.2005.853421. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.74.9149 [accessed 2008-06-23].

[1499] Tadashi Nakano and Tatsuya Suda. Applying biological principles to designs of network services. *Applied Soft Computing*, 7(3):870–878, June 2007. ISSN: 1568-4946. doi:10.1016/j.asoc.2006.04.006.

[1500] H. Nakayama and Y. Sawaragi, editors. *Proceedings of the 7th International Conference on Multiple Criteria Decision Making (MCDM'1986)*, 1986, Kyoto, Japan.

[1501] *Collected Abstracts for the First International Workshop on Learning Classifier Systems (IWLCS-92)*, October 6–9, 1992, NASA Johnson Space Center in Houston, Texas, USA.

[1502] Bart Naudts. *Het meten van GA-hardheit – Measuring GA-hardness*. PhD thesis, Universitaire Instelling Antwerpen, Antwerpen, Netherlands, June 1998. Supervisor: Alain Verschoren. Online available at http://citeseer.ist.psu.edu/185231.html and http://libra.msra.cn/paperdetail.aspx?id=322506 [accessed 2007-07-29].

[1503] Bart Naudts and Alain Verschoren. Epistasis on finite and infinite spaces. In *Proceedings of the 8th International Conference on Systems Research, Informatics and Cybernetics*, pages 19–23, 1996. Online available at http://en.scientificcommons.org/155291 and http://citeseer.ist.psu.edu/142750.html [accessed 2007-08-13].

[1504] Bart Naudts and Alain Verschoren. Epistasis and deceptivity. *Bulletin of the Belgian Mathematical Society*, 6(1):147–154, 1999. Zentralblatt Math identifier: 0915.68143, Mathematical Reviews number (MathSciNet): MR1674709. Online available at http://citeseer.ist.psu.edu/14550.html and http://projecteuclid.org/euclid.bbms/1103149975 [accessed 2007-11-05].

[1505] Bart Naudts, Dominique Suys, and Alain Verschoren. Generalized royal road functions and their epistasis. *Computers and Artificial Intelligence*, 19(4), 2000. Original: March 5, 1997. Online available at http://citeseer.ist.psu.edu/111356.html [accessed 2007-08-13].

[1506] Andreas Nearchou and Sotiris Omirou. Differential evolution for sequencing and scheduling optimization. *Journal of Heuristics*, 12(6):395–411(17), December 2006. Online available at http://www.springerlink.com/content/p6761p070470k448/fulltext.pdf [accessed 2007-08-13].

[1507] Bernhard Nebel, editor. *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001*, volume 1 and 2, August 4–10, 2001, Seattle, Washington, USA. Morgan Kaufmann, San Francisco, CA, USA. ISBN: 1-5586-0777-3. Online available at http://dli.iiit.ac.in/ijcai/IJCAI-2001/content/content.htm [accessed 2008-04-01]. See also [1345].

[1508] Nadia Nedjah and Luiza de Macedo Mourelle, editors. *Swarm Intelligent Systems*, volume 26 of *Studies in Computational Intelligence*. Springer, Berlin/Heidelberg/New York, July 28, 2006. ISBN: 3-5403-3868-3, 978-3-54033-868-0. Series editor: Janusz Kacprzyk.

[1509] Nadia Nedjah and Luiza de Macedo Mourelle, editors. *Systems Engineering using Particle Swarm Optimisation*. Nova Publishers, February 28, 2007. ISBN: 1-6002-1119-4, 978-1-60021-119-5. Partly online available at http://books.google.com/books?id=opkyL6fFPrwC [accessed 2008-08-23].

[1510] Nadia Nedjah, Luiza de Macedo Mourelle, Ajith Abraham, and Mario Köppen, editors. *5th International Conference on Hybrid Intelligent Systems (HIS 2005)*, November 6–9, 2005, Pontifical Catholic University of Rio de Janeiro: PUC-Rio, Rio de Janeiro, Brazil. IEEE Computer Society. ISBN: 0-7695-2457-5. see http://his05.hybridsystem.com/ [accessed 2007-09-01].

[1511] Nadia Nedjah, Ajith Abraham, and Luiza de Macedo Mourelle, editors. *Genetic Systems Programming: Theory and Experiences*, volume 13 of *Studies in Computational Intelligence*. Springer Publishing Company, Berlin / Heidelberg, Germany, 2006. ISBN: 3-5402-9849-5, 978-3-54029-849-6. doi:10.1007/11521433. Partly online available at http://books.google.de/books?id=0QO1XE9X3gcC [accessed 2008-09-16].

[1512] Nadia Nedjah, Enrique Alba, and Luiza De Macedo Mourelle, editors. *Parallel Evolutionary Computations*, volume 22/2006 of *Studies in Computational Intelligence*. Springer Berlin / Heidelberg, June 2006. ISBN: 978-3-54032-837-7. doi:10.1007/3-540-32839-4.

[1513] Christopher J. Neely and Paul A. Weller. Predicting exchange rate volatility: genetic programming versus garch and riskmetrics. *Review*, pages 43–54, May 2002. Online available at http://research.stlouisfed.org/publications/review/02/05/43-54NeelyWeller.pdf [accessed 2007-09-09].

[1514] Mircea Gh. Negoita, Robert J. Howlett, and Lakhmi C. Jain, editors. *Proceedings of the 8th International Conference on Knowledge-Based Intelligent Information and Engineering Systems, KES 2004, Part I*, volume 3213 of *Lecture Notes in Computer Science (LNCS)*, September 20–25, 2004, Wellington, New Zealand. Springer. ISBN: 3-5402-3318-0. See http://www.kesinternational.org/kes2004/ [accessed 2008-07-27]. See also [1515, 1516].

[1515] Mircea Gh. Negoita, Robert J. Howlett, and Lakhmi C. Jain, editors. *Proceedings of the 8th International Conference on Knowledge-Based Intelligent Information and Engineering Systems, KES 2004, Part II*, volume 3214 of *Lecture Notes in Computer Science (LNCS)*, September 20–25, 2004, Wellington, New Zealand. Springer. ISBN: 3-5402-3206-0. See http://www.kesinternational.org/kes2004/ [accessed 2008-07-27]. See also [1514, 1516].

[1516] Mircea Gh. Negoita, Robert J. Howlett, and Lakhmi C. Jain, editors. *Proceedings of the 8th International Conference on Knowledge-Based Intelligent Information and Engineering Systems, KES 2004, Part III*, volume 3215 of *Lecture Notes in Computer Science (LNCS)*, September 20–25, 2004, Wellington, New Zealand. Springer. ISBN: 3-5402-3205-2. See http://www.kesinternational.org/kes2004/ [accessed 2008-07-27]. See also [1514, 1515].

[1517] John Ashworth Nelder and Roger A. Mead. A simplex method for function minimization. *Computer Journal*, 7:308–313, 1965.

[1518] Kevin M. Nelson. A comparison of evolutionary programming and genetic algorithms for electronic part placement. In *Evolutionary Programming IV: Proceedings of the*

Fourth Annual Conference on Evolutionary Programming, pages 503–519, 1995. In proceedings [1380].

[1519] Arnold Neumaier. Global optimization and constraint satisfaction. In I. Bomze, I. Emiris, Arnold Neumaier, and L. Wolsey, editors, *Proceedings of GICOLAG workshop (of the research project Global Optimization, Integrating Convexity, Optimization, Logic Programming and Computational Algebraic Geometry)*, December 2006. Online available at http://www.mat.univie.ac.at/~neum/glopt.html [accessed 2007-07-12].

[1520] N. Neumann. Some procedures for calculating the distributions of elementary nonparametric test statistics. *Statistical Software Newsletter*, 14(3), 1988.

[1521] M. E. J. Newman and R. Engelhardt. Effect of neutral selection on the evolution of molecular species. *Proceedings of the Royal Society of London B (Biological Sciences)*, 256(1403):1333–1338, July 22, 1998. ISSN: 0962-8452 (Print) 1471-2954 (Online). doi:10.1098/rspb.1998.0438. Online available at http://citeseer.ist.psu.edu/202669.html and http://journals.royalsociety.org/content/5ttwagyubu88bang/fulltext.pdf [accessed 2008-06-01].

[1522] Jerzy Neyman and Egon Sharpe Pearson. On the use and interpretation of certain test criteria for purposes of statistical inference, part i. *Biometrika*, 20A(1/2):175–240, July 1928. In collection [1523].

[1523] Jerzy Neyman and Egon Sharpe Pearson. *Joint Statistical Papers*. Cambridge University Press / Hodder Arnold / Lubrecht & Cramer, Ltd., Cambridge, UK, January 1967. ISBN: 0-8526-4706-9, 978-0-85264-706-6. ASIN: B000P0O0T2.

[1524] Xuan Hoai Nguyen. Solving trigonometric identities with tree adjunct grammar guided genetic programming. In *Hybrid Information Systems, Proceedings of First International Workshop on Hybrid Intelligent Systems*, pages 339–351, 2001. In proceedings [6]. See also [1531].

[1525] Xuan Hoai Nguyen. *A Flexible Representation for Genetic Programming: Lessons from Natural Language Processing*. PhD thesis, School of Information Technology and Electrical Engineering University College, University of New South Wales, Australian Defence Force Academy, December 2004. Online available at http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/hoai_thesis.html and http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/hoai_thesis.tar.gz [accessed 2007-07-15].

[1526] Xuan Hoai Nguyen and Robert Ian McKay. A framework for tree-adjunct grammar guided genetic programming. In *Proceedings of the Post-graduate ADFA Conference on Computer Science (PACCS'01)*, volume 1 of *ADFA Monographs in Computer Science Series*, pages 93–100, July 14, 2001, Canberra ACT, Australia. ISBN: 0-7317-0507-6. Online available at http://sc.snu.ac.kr/PAPERS/TAG3P.pdf [accessed 2007-09-09].

[1527] Xuan Hoai Nguyen, Robert Ian McKay, and Daryl Leslie Essam. Some experimental results with tree adjunct grammar guided genetic programming. In *EuroGP'02: Proceedings of the 5th European Conference on Genetic Programming*, pages 229–238, 2002. In proceedings [737]. Online available at http://sc.snu.ac.kr/PAPERS/TAGGGP.pdf [accessed 2007-09-09].

[1528] Xuan Hoai Nguyen, Robert Ian McKay, and Daryl Leslie Essam. Can tree adjunct grammar guided genetic programming be good at finding a needle in a haystack? – a case study. In *Proceedings of IEEE International Conference on Communications, Circuits and Systems and West Sino Expositions*, volume 2, pages 1113–1117. IEEE Press, June 29–July 1, 2002. Online available at http://sc.snu.ac.kr/PAPERS/hoaietal.pdf [accessed 2007-09-09].

[1529] Xuan Hoai Nguyen, Robert Ian McKay, Daryl Leslie Essam, and R. Chau. Solving the symbolic regression problem with tree-adjunct grammar guided genetic programming: the comparative results. In *CEC'02: Proceedings of the Congress on Evolutionary*

Computation 2002, pages 1326–1331, 2002. In proceedings [703]. Online available at http://sc.snu.ac.kr/PAPERS/MCEC2002.pdf [accessed 2007-09-09].

[1530] Xuan Hoai Nguyen, Robert Ian McKay, and H. A. Abbass. Tree adjoining grammars, language bias, and genetic programming. In *Genetic Programming: 6th European Conference*, pages 157–183, 2003. In proceedings [1786]. Online available at http://sc.snu.ac.kr/PAPERS/eurogp03.pdf [accessed 2007-09-10].

[1531] Xuan Hoai Nguyen, Robert Ian McKay, and Daryl Leslie Essam. Finding trigonometric identities with tree adjunct grammar guided genetic programming. In Ajith Abraham, Lakhmi C. Jain, and Berend van der Zwaag, editors, *Innovations in Intelligent Systems and Applications*, volume 140 of *Springer Studies in Fuzziness and Soft Computing*, pages 221–236. Springer-Verlag, January 2004. ISBN: 978-3-54020-265-3. See also [1524]. Online available at http://sc.snu.ac.kr/PAPERS/trigonometry.pdf [accessed 2007-09-09].

[1532] Bertram Nickolay, Bernd Schneider, and Stefan Jacob. Parameter optimisation of an image processing system using evolutionary algorithms. In *Computer Analysis of Images and Patterns, Proceedings of the 7th International Conference, CAIP'97*, volume 1296/1997 of *Lecture Notes in Computer Science (LNCS)*, pages 637–644. Springer Berlin / Heidelberg, September 10-12 1997, Kiel, Germany. ISBN: 978-3-54063-460-7. doi:10.1007/3-540-63460-6_173.

[1533] Stefan Niemczyk. Ein modellproblem mit einstellbarer schwierigkeit zur evaluierung evolutionärer algorithmen. Diplom i arbeit (equiv. bachelor thesis), Universität Kassel, Fachgebiet Elektrotechnik/Informatik, Fachbereich Verteilte Systeme, Wilhelmshöher Allee 73, 34121 Kassel, Deutschland, May 5, 2008. Advisor: Thomas Weise. Supervisor: Kurt Geihs. Committee: Albert Zündorf. Online available at http://www.it-weise.de/documents/files/N2008MPMES.pdf [accessed 2009-06-26].

[1534] Pat Niemeyer and Jonathan Knudsen. *Learning Java*. O'Reilly Media, Inc, third edition, June 2005. ISBN: 978-0-59600-873-4.

[1535] Volker Nissen. *Einführung in evolutionäre Algorithmen: Optimierung nach dem Vorbild der Evolution*. Vieweg, Braunschweig, 1997. ISBN: 3-5280-5499-9, 978-3-52805-499-1.

[1536] Volker Nissen and Matthias Krause. Constrained combinatorial optimization with an evolution strategy. In *Proceedings of Fuzzy Logik, Theorie und Praxis, 4. Dortmunder Fuzzy-Tage*, pages 33–40, June 1994, Dortmund, Germany. Springer-Verlag, London, UK. ISBN: 3-5405-8649-0.

[1537] Dusit Niyato, Ekram Hossain, and Afshin Fallahi. Sleep and wakeup strategies in solar-powered wireless sensor/mesh networks: Performance analysis and optimization. *IEEE Transactions on Mobile Computing*, 6(2):221–236, 2007. ISSN: 1536-1233.

[1538] David Noever and Subbiah Baskaran. Steady-state vs. generational genetic algorithms: A comparison of time complexity and convergence properties. Technical Report 92-07-032, Santa Fe Institute, 1992.

[1539] Andreas Nolte and Rainer Schrader. A note on the finite time behaviour of simulated annealing. In U. Zimmermann, U. Derigs, W. Gaul, R. Möhring, and K.-P. Schuster, editors, *Operations Research Proceedings 1996, Selected Papers of the Symposium on Operations Research (SOR 96)*, pages 175–180. Springer, 1996. Published 1997. See also [1540].

[1540] Andreas Nolte and Rainer Schrader. A note on the finite time behaviour of simulated annealing. *Mathematics of OR*, 25(347):476–484, 2000. Online available at http://citeseer.ist.psu.edu/nolte99note.html and http://www.zaik.de/~paper/unzip.html?file=zaik1999-347.ps [accessed 2008-03-26] (revised version from March 1999). See also [1539].

[1541] Peter Nordin. A compiling genetic programming system that directly manipulates the machine code. In *Advances in genetic programming 1*, chapter 14, pages 311–331. MIT Press, 1994. In collection [1140]. Machine code GP Sun Spark and i868.

[1542] Peter Nordin. *Evolutionary Program Induction of Binary Machine Code and its Applications*. PhD thesis, der Universitat Dortmund am Fachereich Informatik, Krehl Verlag, Münster, Germany, 1997. ISBN: 3-9315-4607-1, 978-3-93154-607-6.

[1543] Peter Nordin and Wolfgang Banzhaf. Evolving turing-complete programs for a register machine with self-modifying code. In *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA95)*, pages 318–325, 1995. In proceedings [636]. Online available at http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/Nordin_1995_tcp.html and http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.34.4526 [accessed 2008-09-16].

[1544] Peter Nordin and Wolfgang Banzhaf. Complexity compression and evolution. In *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, pages 310–317, 1995. In proceedings [636]. Online available at http://citeseer.ist.psu.edu/nordin95complexity.html [accessed 2007-09-07].

[1545] Peter Nordin and Wolfgang Banzhaf. Programmatic compression of images and sound. In *Proceedings of the First Annual Conference on Genetic Programming*, pages 345–350, 1996. In proceedings [1207]. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.32.7927 and http://www.cs.mun.ca/~banzhaf/papers/gp96.pdf [accessed 2008-09-16].

[1546] Peter Nordin, Frank Francone, and Wolfgang Banzhaf. Explicitly defined introns and destructive crossover in genetic programming. In *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 6–22, 1995. In proceedings [1757] and also [1547]. Online available at http://citeseer.ist.psu.edu/nordin95explicitly.html and http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/nordin_1995_introns.html [accessed 2007-09-07].

[1547] Peter Nordin, Frank Francone, and Wolfgang Banzhaf. Explicitly defined introns and destructive crossover in genetic programming. In *Advances in genetic programming 2*, pages 111–134. MIT Press, 1996. In collection [61] and also [1546]. SysReport 3/95, System Analysis, Department of Computer Science, University of Dortmund, Germany. Online available at http://citeseer.comp.nus.edu.sg/349513.html and http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/nordin_1995_introns.html [accessed 2008-09-17], ftp://lumpi.informatik.uni-dortmund.de/pub/biocomp/papers/ML95.ps.gz [accessed 2007-08-17].

[1548] Peter Nordin, Wolfgang Banzhaf, and Markus Brameier. Evolution of a world model for a miniature robot using genetic programming. *Robotics and Autonomous Systems*, 25(1–2):105–116, October 31, 1998. ISSN: 0921-8890. doi:10.1016/S0921-8890(98)00004-9. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.54.1211 and http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/Nordin_1998_RAS.html [accessed 2008-09-16].

[1549] Peter Nordin, Wolfgang Banzhaf, and Frank D. Francone. Efficient evolution of machine code for CISC architectures using instruction blocks and homologous crossover. In *Advances in Genetic Programming 3*, chapter 12, pages 275–299. MIT Press, 1999. In collection [1936]. Online available at http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/nordin_1999_aigp3.html and http://www.cs.bham.ac.uk/~wbl/aigp3/ch12.pdf [accessed 2008-09-16].

[1550] Peter Nordin, Frank Hoffmann, Frank D. Francone, and Markus Brameier. Aimgp and parallelism. In *IEEE Congress on Evolutionary Computation*, pages 1059–1066, 1999. In proceedings [69]. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.6.5450 and http://www.cs.mun.ca/~banzhaf/papers/cec_parallel.pdf [accessed 2008-09-16].

[1551] M. G. Norman and Pablo Moscato. A competitive and cooperative approach to complex combinatorial search. In *Proceedings of the 20th Informatics and Operations Research Meeting (20th JAIIO, Jornadas Argentinas e Informática e Investigación Operativa)*, pages 3.15–3.29, August 20–23, 1991, Centro Cultural General San Martn,

Capital Federal, Buenos Aires, Argentina. Also published as Technical Report Caltech Concurrent Computation Program, Report. 790, California Institute of Technology, Pasadena, California, USA, 1989.

[1552] M. Nowak and P. Schuster. Error thresholds of replication in finite populations mutation frequencies and the onset of muller's ratchet. *Journal of Theoretical Biology*, 137(4):375–395, April 20, 1989. ISSN: 0022-5193.

O

[1553] Martin J. Oates and David W. Corne. Global web server load balancing using evolutionary computational techniques. *Soft Computing – A Fusion of Foundations, Methodologies and Applications*, 5(4):297–312, August 2001. ISSN: 1432-7643 (Print) 1433-7479 (Online). doi:10.1007/s005000100103. Online available at http://www.springerlink.com/content/g1u7lygrugwk3nl5/fulltext.pdf [accessed 2008-09-03].

[1554] Shigeru Obayashi. Multidisciplinary design optimization of aircraft wing planform based on evolutionary algorithms. In *Proceedings of the 1998 IEEE International Conference on Systems, Man, and Cybernetics*. IEEE Press, October 11–14, 1998, La Jolla, California, USA. Online available at [accessed 2008-06-14].

[1555] Shigeru Obayashi, Kalyanmoy Deb, Carlo Poloni, Tomoyuki Hiroyasu, and Tadahiko Murata, editors. *Proceedings of the Fourth International Conference on Evolutionary Multi-Criterion Optimization (EMO'2007)*, volume 4403/2007 of *Lecture Notes in Computer Science (LNCS)*, March 5–8, 2007, Matsushima/Sendai, Japan. Springer-Verlag, Berlin. ISBN: 978-3-54070-927-5. See http://www.is.doshisha.ac.jp/emo2007/ [accessed 2007-09-11].

[1556] P. O'Brien, D. Corcoran, and D. Lowry. An evolution strategy to estimate emission source distributions on a regional scale from atmospheric observations. *Atmospheric Chemistry & Physics Discussions*, 3:1333–1366, February 2003. Provided by the Smithsonian/NASA Astrophysics Data System. Online available at http://www.atmos-chem-phys-discuss.net/3/1333/2003/acpd-3-1333-2003.html [accessed 2007-08-27].

[1557] Gabriela Ochoa, Inman Harvey, and Hilary Buxton. Error thresholds and their relation to optimal mutation rates. In *European Conference on Artificial Life*, pages 54–63, 1999. In proceedings [689]. Online available at http://citeseer.ist.psu.edu/ochoa99error.html and ftp://ftp.cogs.susx.ac.uk/pub/users/inmanh/ecal99.ps.gz [accessed 2008-07-20].

[1558] Christopher K. Oei, David E. Goldberg, and Shau-Jin Chang. Tournament selection, niching, and the preservation of diversity. IlliGAl Report 91011, Illinois Genetic Algorithms Laboratory (IlliGAL), Department of Computer Science, Department of General Engineering, University of Illinois at Urbana-Champaign, December 1991. Online available at http://www.illigal.uiuc.edu/pub/papers/IlliGALs/91011.ps.Z [accessed 2008-03-15].

[1559] Euclid of Alexandria, editor. *Stoicheia (Elements)*. Euclid of Alexandria, Alexandria, 300 BC. A series consisting of 13 books, to which two books were added later on. Online available at http://aleph0.clarku.edu/~djoyce/java/elements/elements.html and http://www.gutenberg.org/etext/21076 [accessed 2007-10-05] (in english). Online available at http://farside.ph.utexas.edu/euclid.html and http://en.wikipedia.org/wiki/Image:Euclid-Elements.pdf [accessed 2007-10-05] (bilingual/-greek). See also [913] and http://en.wikipedia.org/wiki/Euclid%27s_Elements [accessed 2007-10-05].

[1560] Anne L. Olsen. Penalty functions and the knapsack problem. In *Proceedings of the First IEEE Conference on Evolutionary Computation*, volume 2, pages 554–558, 1994. doi:10.1109/ICEC.1994.350000. In proceedings [1411].

[1561] Donald M. Olsson and Lloyd S. Nelson. The nelder-mead simplex procedure for function minimization. *Technometrics*, 17(1):45–51, February 1975.

[1562] Mihai Oltean. Evolving evolutionary algorithms for function optimization. In *Proceedings of the 5th International Workshop on Frontiers in Evolutionary Algorithms*, pages 295–298, 2003. In proceedings [639]. Online available at http://www.cs.ubbcluj.ro/~moltean/oltean_fea2003_1.pdf and http://citeseer.ist.psu.edu/640208.html [accessed 2007-08-24].

[1563] Mihai Oltean. Searching for a practical evidence of the no free lunch theorems. In *Biologically Inspired Approaches to Advanced Information Technology, Revised Selected Papers of the First International Workshop, BioADIT 2004*, volume 3141/2004 of *Lecture Notes in Computer Science (LNCS)*, pages 472–483. Springer Berlin / Heidelberg, January 29–30, 2004, Lausanne, Switzerland. ISBN: 978-3-54023-339-8. doi:10.1007/b101281. Online available at http://www.cs.ubbcluj.ro/~moltean/oltean_bioadit_springer2004.pdf [accessed 2009-03-01].

[1564] Kunle Olukotun, Basem A. Nayfeh, Lance Hammond, Ken Wilson, and Kunyung Chang. The case for a single-chip multiprocessor. In *ASPLOS-VII: Proceedings of the seventh international conference on Architectural support for programming languages and operating systems*, pages 2–11, 1996, Cambridge, Massachusetts, United States. ACM Press, New York, NY, USA. ISBN: 0-8979-1767-7. Online available at http://ogun.stanford.edu/~kunle/publications/hydra_ASPLOS_VII.pdf and http://doi.acm.org/10.1145/237090.237140 [accessed 2007-09-11].

[1565] Michael O'Neill. Grammatical evolution. In *Proceedings of the Fifth Research Conference of the Deptartment of Computer Science and Information Systems, University of Limerick*, September 1998.

[1566] Michael O'Neill. *Automatic Programming in an Arbitrary Language: Evolving Programs with Grammatical Evolution*. PhD thesis, University of Limerick, 2001.

[1567] Michael O'Neill and Anthony Brabazon. Grammatical differential evolution. In Hamid R. Arabnia, editor, *Proceedings of the 2006 International Conference on Artificial Intelligence, ICAI 2006*, volume 1, pages 231–236. CSREA Press, June 26–29, 2006, Las Vegas, Nevada, USA. ISBN: 1-9324-1596-3. Online available at http://ww1.ucmss.com/books/LFS/CSREA2006/ICA4864.pdf [accessed 2007-09-09].

[1568] Michael O'Neill and Anthony Brabazon. Grammatical swarm: The generation of programs by social programming. *Natural Computing: an international journal*, 5(4):443–462, November 2006. ISSN: 1567-7818 (Print) 1572-9796 (Online). doi:10.1007/s11047-006-9007-7. Online available at http://dx.doi.org/10.1007/s11047-006-9007-7 and http://www.springerlink.com/content/p3t1923gr7725583/fulltext.pdf [accessed 2007-09-09].

[1569] Michael O'Neill and Conor Ryan. Evolving multi-line compilable c programs. In *Proceedings of the Second European Workshop on Genetic Programming*, pages 83–92, 1999. In proceedings [1664]. Online available at http://citeseer.ist.psu.edu/278127.html and http://www.grammatical-evolution.org/papers/eurogp99.ps.gz [accessed 2007-09-09].

[1570] Michael O'Neill and Conor Ryan. *Grammatical Evolution: Evolutionary Automatic Programming in an Arbitrary Language*, volume 4 of *Genetic Programming*. Kluwer Academic Publishers / Springer, May 2003. ISBN: 1-4020-7444-1, 978-1-40207-444-8. Series Editor: John R. Koza. Foreword from Wolfgang Banzhaf. Partly online available at http://books.google.de/books?id=yVx5dg_opRIC [accessed 2008-09-19].

[1571] Michael O'Neill and Conor Ryan. Grammatical evolution by grammatical evolution: The evolution of grammar and genetic code. In *Proceedings of 7th European Conference on Genetic Programming, EuroGP2004*, pages 138–149, 2004. In proceedings [1115]. Online available at http://ncra.ucd.ie/papers/eurogp2004.pdf [accessed 2007-08-28].

[1572] Michael O'Neill and Conor Ryan, editors. *Grammatical Evolution Workshop (GEWS 2002)*, July 9, 2002, New York, NY, USA. Part of GECCO, see [1245] and also http://www.grammatical-evolution.com/gews2002/ [accessed 2007-09-10].

[1573] Michael O'Neill and Conor Ryan, editors. *The 2nd Grammatical Evolution Workshop (GEWS 2003)*, July 2003, Chicago, IL, USA. Part of GECCO, see [334, 335] and also http://www.grammatical-evolution.com/gews2003/ [accessed 2007-09-10].

[1574] Michael O'Neill and Conor Ryan, editors. *The 3rd Grammatical Evolution Workshop (GEWS 2004)*, June 26, 2004, Seattle, WA, USA. Part of GECCO, see [544, 545] and also http://www.grammatical-evolution.com/gews2004/ [accessed 2007-09-10].

[1575] Michael O'Neill, J. J. Collins, and Conor Ryan. Automatic programming of robots. In *Proceedings of Artificial Intelligence and Cognitive Science AICS 2000*, 2000.

[1576] Michael O'Neill, J. J. Collins, and Conor Ryan. Automatic generation of robot behaviours using grammatical evolution. In *Proceedings of the Fifth International Symposium on Artificial Life and Robotics AROB 2000*, pages 351–354, 2000, Japan.

[1577] Michael O'Neill, Anthony Brabazon, Conor Ryan, and J. J. Collins. Evolving market index trading rules using grammatical evolution. In *Proceedings of the EvoWorkshops on Applications of Evolutionary Computing*, pages 343–352, 2001. In proceedings [235]. Online available at http://citeseer.ist.psu.edu/485032.html [accessed 2007-09-09].

[1578] Michael O'Neill, Finbar Leahy, and Anthony Brabazon. Grammatical swarm: A variable-length particle swarm algorithm. In *Swarm Intelligent Systems*, chapter 5. Springer, 2006. In collection [1508].

[1579] Michael O'Neill, Leonardo Vanneschi, Steven Matt Gustafson, Anna Isabel Esparcia Alcázar, Ivanoe De Falco, Antonio Della Cioppa, and Ernesto Tarantino, editors. *Genetic Programming – Proceedings of the 11th European Conference on Genetic Programming, EuroGP 2008*, volume 4971/2008 of *Lecture Notes in Computer Science (LNCS), Subseries: Theoretical Computer Science and General Issues*, March 26–28 2008, Naples, Italy. Springer-Verlag GmbH, Berlin/Heidelberg, Germany. ISBN: 3-5407-8670-8, 978-3-54078-670-2. doi:10.1007/978-3-540-78671-9. Partly online available at http://books.google.de/books?id=OAHUcHbXoNkC [accessed 2008-12-24].

[1580] Godfrey C. Onwubolu and B. V. Babu, editors. *New Optimization Techniques in Engineering*, volume 141 of *Studies in Fuzziness and Soft Computing*. Springer-Verlag, Berlin/Heidelberg, Germany, March 5, 2004. ISBN: 3-5402-0167-X, 978-3-54020-167-0. Partly online available at http://books.google.de/books?id=YRsq-Pz0kAAC [accessed 2008-06-07].

[1581] Stan Openshaw. Building an automated modelling system to explore a universe of spatial interaction models. *Geographical Analysis – An International Journal of Theoretical Geography*, 26(1):31–46, 1988.

[1582] Stan Openshaw and Ian Turton. Building new spatial interaction models using genetic programming. In *Workshop on Artificial Intelligence and Simulation of Behaviour (AISB), International Workshop on Evolutionary Computing, Selected Papers*, 1994. In proceedings [693]. Online available at http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/openshaw_1994_bnsim.html and http://www.geog.leeds.ac.uk/papers/94-1/94-1.pdf [accessed 2008-09-16].

[1583] Una-May O'Reilly, Gwoing Tina Yu, Rick Riolo, and Bill Worzel, editors. *Genetic Programming Theory and Practice II, Proceedings of the Genetic Programming Theory Practice 2004 Workshop (GPTP-2004)*, volume 8 of *Genetic Programming Series*, May 13-15, 2004, The Center for the Study of Complex Systems (CSCS), University of Michigan, Ann Arbor, Michigan, USA. Springer, New York, USA. ISBN: 978-0-38723-253-9, 978-0-38723-254-6. doi:10.1007/b101112. See http://www.cscs.umich.edu/gptp-workshops/gptp2004/ [accessed 2007-09-28].

[1584] *Late Breaking Papers at Genetic and Evolutionary Computation Conference (GECCO 1999)*, July 13–17, 1999, Orlando, Florida, USA. See also [142, 1889].

[1585] *The Second International Workshop on Learning Classifier Systems (IWLCS-99)*, July 13, 1999, Orlando, Florida, USA. Co-located with GECCO-99 (see also [142]). Proceedings are published in [1252].

[1586] Henry F. Osborn. Oytogenic and phylogenic variation. *Science*, 4(100):786–789, November 27 1896. doi:10.1126/science.4.100.786.

[1587] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory*. The MIT Press, July 1994. ISBN: 0-2626-5040-1.

[1588] Ibrahim H. Osman. An introduction to metaheuristics. In M. Lawrence and C. Wilsdon, editors, *Operational Research Tutorial Papers*, pages 92–122. Stockton Press, Hampshire, UK, 1995. Publication of the Operational Research Society, Birmingham, UK.

[1589] Ibrahim H. Osman and James P. Kelly, editors. *1st Metaheuristics International Conference (MIC'95) – Meta-Heuristics: The Theory and Applications*, July 22–26, 1995, Breckenridge, Colorado, USA. Kluwer Academic Publishers, Boston, USA. ISBN: 0-7923-9700-2. Published in 1996.

[1590] P. Ošmera and R. Matoušek, editors. *Proceedings of the 13th International Conference on Soft Computing, MENDEL'07*, September 5–7, 2007, Prague, Czech Republic. Brno University of Technology, Faculty of Mechanical Engineering. ISBN: 978-8-02143-473-8.

[1591] Pavel Osmera, editor. *Proceedings of the 6th International Conference on Soft Computing, MENDEL 2000*, June 7–9, 2000, Brno University of Technology, Brno, Czech Republic. Ústav Automatizace a Informatiky. ISBN: 8-0214-1609-2.

[1592] Fernando E. B. Otero, Monique M. S. Silvia, and Alex A. Freitas. Genetic programming for attribute construction in data mining. In *GECCO'02: Proceedings of the Genetic and Evolutionary Computation Conference*, 2002. In proceedings [1245] and also [1593].

[1593] Fernando E. B. Otero, Monique M. S. Silva, Alex A. Freitas, and J. C. Nievola. Genetic programming for attribute construction in data mining. In *Genetic Programming: Proc. 6th European Conference (EuroGP-2003)*, pages 384–393, 2003. In proceedings [1786] and also [1592]. Online available at http://www.cs.kent.ac.uk/people/staff/aaf/pub_papers.dir/EuroGP-2003-Fernando.pdf [accessed 2007-09-09].

[1594] Claude Overstreet, Jr. and Richard E. Nance. A random number generator for small word-length computers. In *ACM'73: Proceedings of the annual conference*, pages 219–223, 1973, Atlanta, Georgia, United States. ACM Press/CSC-ER, New York, NY, USA. doi:10.1145/800192.805707. Online available at http://doi.acm.org/10.1145/800192.805707 [accessed 2007-09-15].

[1595] Gultekin Özsoyoglu, Z. Meral Özsoyoglu, and Victor Matos. Extending relational algebra and relational calculus with set-valued attributes and aggregate functions. *ACM Transactions on Database Systems*, 12(4):566–592, 1987. Online available at http://doi.acm.org/10.1145/32204.32219 [accessed 2007-09-12].

P

[1596] Jukka Paakki. Attribute grammar paradigms – a high-level methodology in language implementation. *ACM Computing Surveys (CSUR)*, 27(2):196–255, 1995. ISSN: 0360-0300. doi:10.1145/210376.197409. Online available at http://doi.acm.org/10.1145/210376.197409 [accessed 2007-09-15].

[1597] B. Paechter, Thomas Bäck, Marc Schoenauer, M. Sebag, Ágoston E. Eiben, J. J. Merelo, and T. C. Fogarty. A distributed resource evolutionary algorithm machine (dream). In *Proceedings of the Congress on Evolutionary Computation*, volume 2, pages 951–958, 2000. doi:10.1109/CEC.2000.870746. In proceedings [1002].

[1598] Ingo Paenke, Jürgen Branke, and Yaochu Jin. On the influence of phenotype plasticity on genotype diversity. In *First IEEE Symposium on Foundations of Computational Intelligence (FOCI'07)*, pages 33–40, 2007. doi:10.1109/FOCI.2007.372144. In proceedings [1388] (Best student paper). Online available at http://www.soft-computing.de/S001P005.pdf and http://www.honda-ri.org/intern/Publications/PN%2052-06 [accessed 2008-11-10].

[1599] Vasile Palade, Robert J. Howlett, and Lakhmi C. Jain, editors. *Proceedings of the 7th International Conference on Knowledge-Based Intelligent Information and Engineering Systems, KES 2003, Part I*, volume 2773 of *Lecture Notes in Computer Science (LNCS)*, September 3–5, 2003, Oxford, UK. Springer. ISBN: 3-5404-0803-7. See also [1600].

[1600] Vasile Palade, Robert J. Howlett, and Lakhmi C. Jain, editors. *Proceedings of the 7th International Conference on Knowledge-Based Intelligent Information and Engineering Systems, KES 2003, Part II*, volume 2774 of *Lecture Notes in Computer Science (LNCS)*, September 3–5, 2003, Oxford, UK. Springer. ISBN: 3-5404-0804-5. See also [1599].

[1601] Charles Campbell Palmer. *An approach to a problem in network design using genetic algorithms.* PhD thesis, Polytechnic University, New York, NY, Polytechnic University, New York, NY, April 1994. Supervisors: Aaron Kershenbaum, Susan Flynn Hummel, Richard M. van Slyke, Robert R. Boorstyn. UMI Order No. GAX94-31740. Appears also as article in Wiley InterScience, Networks, Volume 26, Issue 3, Pages 151–163, Online available at http://citeseer.ist.psu.edu/palmer95approach.html [accessed 2007-08-12].

[1602] Charles Campbell Palmer and Aaron Kershenbaum. Representing trees in genetic algorithms. In *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, volume 1, pages 379–384, 1994. In proceedings [1411], see also [1603].

[1603] Charles Campbell Palmer and Aaron Kershenbaum. Representing trees in genetic algorithms. In *Handbook of Evolutionary Computation*. Oxford University Press, 1997. In collection [104], see also [1602]. Online available at http://citeseer.ist.psu.edu/palmer94representing.html and http://www.cs.cinvestav.mx/~constraint/ [accessed 2007-08-12].

[1604] Simonetta Paloscia, Giovanni Macelloni, Paolo Pampaloni, and Emanuele Santi. Retrieval of soil moisture data at global scales with amsr-e. In *XXVIIIth URSI General Assembly*, October 23–29, 2005, New Delhi, India. URSI Secretariat, c/o INTEC, Ghent University, Sint-Pietersnieuwstraat 41, B-9000 Ghent, Belgium. Online available at http://ursiweb.intec.ugent.be/Proceedings/ProcGA05/pdf/F06.2(0590).pdf [accessed 2008-06-14].

[1605] Guanyu Pan, Quansheng Dou, and Xiaohua Liu. Performance of two improved particle swarm optimization in dynamic optimization environments. In *ISDA'06: Proceedings of the Sixth International Conference on Intelligent Systems Design and Applications (ISDA'06)*, volume 2, pages 1024–1028, 2006, Jinan, China. IEEE Computer Society, Washington, DC, USA. ISBN: 0-7695-2528-8.

[1606] Hui Pan, Ling Wang, and Bo Liu. Particle swarm optimization for function optimization in noisy environment. *Applied Mathematics and Computation*, 181(2):908–919, October 15, 2006. ISSN: 0096-3003. CODEN: AMHCBQ. doi:10.1016/j.amc.2006.01.066.

[1607] Michael J. Panik. *Advanced Statistics from an Elementary Point of View.* Academic Press, October 2005. ISBN: 0-1208-8494-1, 978-0-12088-494-0. Partly online available at http://books.google.de/books?id=aHZ75UvBR-0C [accessed 2008-08-18].

[1608] Mike P. Papazoglou and John Zeleznikow, editors. *The Next Generation of Information Systems: From Data to Knowledge – A Selection of Papers Presented at Two*

IJCAI-91 Workshops, volume 611 of *Lecture Notes in Computer Science (LNCS)*, 1992, Sydney, Australia. Springer. ISBN: 3-5405-5616-8. See also [1495, 1496, 596].

[1609] Joseph A. Paradiso. Systems for human-powered mobile computing. In *DAC'06: Proceedings of the 43rd annual conference on Design automation*, pages 645–650, 2006, San Francisco, CA, USA. ACM Press, New York, NY, USA. ISBN: 1-5959-3381-6. Online available at http://www.media.mit.edu/resenv/pubs/papers/2006-07-DAC-Paper.pdf and http://portal.acm.org/citation.cfm?id=1147074 [accessed 2007-08-01].

[1610] Joseph A. Paradiso and Thad Starner. Energy scavenging for mobile and wireless electronics. *IEEE Pervasive Computing*, 04(1):18–27, 2005. ISSN: 1536-1268. doi:10.1109/MPRV.2005.9. Online available at http://www.media.mit.edu/resenv/papers.html and http://whitepapers.silicon.com/0,39024759,60295509p,00.htm [accessed 2007-08-01].

[1611] Abhishek Parakh. A d-sequence based recursive random number generator, 2006. Online available at http://arxiv.org/abs/cs.CR/0603029 [accessed 2007-09-16].

[1612] Panos M. Pardalos and Ding-Zhu Du. *Handbook of Combinatorial Optimization*, volume 1–3 of *Combinatorial Optimization*. Kluwer Academic Publishers, Springer Netherlands, October 31, 1998. ISBN: 0-7923-5019-7, 978-0-79235-019-4.

[1613] Panos M. Pardalos and Mauricio G. C. Resende, editors. *Handbook of Applied Optimization*. Oxford University Press, Inc., 198 Madison Avenue, New York, New York 10016, USA, February 22, 2002. ISBN: 0-1951-2594-0, 978-0-19512-594-8.

[1614] Panos M. Pardalos, Nguyen Van Thoai, and Reiner Horst. *Introduction to Global Optimization*. Nonconvex Optimization and Its Applications. Springer, second edition, December 31, 2000. ISBN: 978-0-79236-756-7. First edition: June 30, 1995, ISBN: 978-0792335566.

[1615] Vilfredo Federico Pareto. *Cours d'Économie Politique*. F. Rouge, Lausanne/Paris, France, 1896–1897. 2 volumes. Partly online available at http://ann.sagepub.com/cgi/reprint/9/3/128.pdf [accessed 2009-04-23] in the ANNALS of the American Academy of Political and Social Science 1897; 9; 128; doi:10.1177/000271629700900314.

[1616] Julia K. Parrish and William M. Hamner, editors. *Animal Groups in Three Dimensions: How Species Aggregate*. Cambridge University Press, December 1997. ISBN: 0-5214-6024-7, 978-0-52146-024-8. Partly online available at http://books.google.de/books?id=zj4I4HNZSR0C [accessed 2008-11-25].

[1617] K. E. Parsopoulos and M. N. Vrahatis. Recent approaches to global optimization problems through particle swarm optimization. *Natural Computing*, 1 (2–3):235–306, June 2002. Online available at http://citeseer.ist.psu.edu/parsopoulos02recent.html [accessed 2007-08-21].

[1618] Emanuel Parzen. On estimation of a probability density function and mode. *Annals of Mathematical Statistics*, 33:1065–1076, August 1962. Online available at http://citeseer.ist.psu.edu/parzen62estimation.html [accessed 2007-08-11].

[1619] Norman R. Paterson. *Genetic programming with context-sensitive grammars*. PhD thesis, Saint Andrew's University, September 2002. Online available at ftp://ftp.dcs.st-and.ac.uk/pub/norman/GPwCSG.ps.gz [accessed 2007-09-09].

[1620] Norman R. Paterson and Mike Livesey. Distinguishing genotype and phenotype in genetic programming. In *Late Breaking Papers at the Genetic Programming 1996 Conference*, pages 141–150, 1996. In proceedings [1197]. Online available at http://citeseer.ist.psu.edu/82479.html and ftp://ftp.dcs.st-and.ac.uk/pub/norman/GADS.ps.gz [accessed 2007-09-07].

[1621] Norman R. Paterson and Mike Livesey. Evolving caching algorithms in c by gp. In *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 262–267, 1997. In proceedings [1208].

[1622] David A. Patterson and John L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface*. The Morgan Kaufmann Series in Computer Architecture and Design. Morgan Kaufmann, third edition, August 2004. ISBN: 978-1-55860-604-3.

[1623] Richard E. Pattis. Ebnf: A notation to describe syntax, the late 1980s. Online available at http://www.cs.cmu.edu/~pattis/misc/ebnf.pdf [accessed 2007-07-03].

[1624] Diane Paul. Fitness: Historical perspectives. In Evelyn Fox Keller and Elisabeth A. Lloyd, editors, *Keywords in Evolutionary Biology*, pages 112–114. Hardvard University Press, Cambridge, MA, USA, November 1992. ISBN: 978-0-67450-312-0, 0-6745-0312-0.

[1625] Linus Pauling. *The Nature of the Chemical Bond*. Cornell Univ. Press, Ithaca, New York, 1960. ISBN: 0-8014-0333-2. Online available at http://osulibrary.oregonstate.edu/specialcollections/coll/pauling/ [accessed 2007-07-12].

[1626] Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. The Addison-Wesley series in artificial intelligence. Addison-Wesley Pub (Sd), April 1984. ISBN: 978-0-20105-594-8.

[1627] David W. Pearson, Nigel C. Steele, and F. Albrecht, editors. *Proceedings of the 2nd International Conference on Artificial Neural Nets and Genetic Algorithms, ICANNGA*, April 18–21, 1995, École des Mines d'Alès, Alès, France. Springer Verlag. ISBN: 3-2118-2692-0.

[1628] David W. Pearson, Nigel C. Steele, and Rudolf F. Albrecht, editors. *Proceedings of the 6th International Conference on Artificial Neural Nets and Genetic Algorithms, ICANNGA*, 2003, Roanne, France. Springer Verlag. ISBN: 978-3-21100-743-3.

[1629] Witold Pedrycz and Athanasios V. Vasilakos. Computational intelligence: A development environment for telecommunications networks. In *Computational Intelligence in Telecommunications Networks*, chapter 1, pages 1–27. CRC Press, 2000. In collection [1630].

[1630] Witold Pedrycz and Athanasios V. Vasilakos, editors. *Computational Intelligence in Telecommunications Networks*. CRC Press, September 15, 2000. ISBN: 0-8493-1075-X, 978-0-84931-075-1. Partly online available at http://books.google.de/books?id=dE6d-jPoDEEC [accessed 2008-09-03].

[1631] Radek Pelánek. Beem: Benchmarks for explicit model checkers. In *Proceedings of SPIN 2007, the 14th International SPIN Workshop on Model Checking Software*, pages 263–267, 2007. doi:10.1007/978-3-540-73370-6_17. In proceedings [256]. Online available at http://spinroot.com/spin/Workshops/ws07/Pelanek.pdf [accessed 2008-10-02].

[1632] Martin Pelikan. Analysis of estimation of distribution algorithms and genetic algorithms on nk landscapes. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2008*, pages 1033–1040, 2008. doi:10.1145/1389095.1389287. In proceedings [1117] Session: Genetic Algorithms Papers. Online available at http://arxiv.org/abs/0801.3111 and http://doi.acm.org/10.1145/1389095.1389287 [accessed 2009-02-26].

[1633] Martin Pelikan, David E. Goldberg, and Erick Cantú-Paz. Boa: The bayesian optimization algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 1999)*, pages 525–532, 1999. In proceedings [142]. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.46.8131 and https://eprints.kfupm.edu.sa/28537/ [accessed 2008-10-18]. Also: IlliGAL Report 99003, January 1999, Illinois Genetic Algorithms Laboratory (IlliGAL), Department of General Engineering, University of Illinois at Urbana-Champaign, 117 Transportation Building, 104 S. Mathews Avenue, Urbana-Champaign, Illinois, 61801-2996, USA, online available at http://www.illigal.uiuc.edu/pub/papers/IlliGALs/99003.ps.Z and http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.45.934 [accessed 2008-10-18].

[1634] Parag C. Pendharkar and Gary J. Koehler. A general steady state distribution based stopping criteria for finite length genetic algorithms. *European Journal of Operational Research*, 176(3):1436–1451, 2007. Online available at http://linkinghub.elsevier.com/retrieve/pii/S0377221705008581 [accessed 2007-07-28].

[1635] Francisco Baptista Pereira and Jorge Tavares, editors. *Bio-inspired Algorithms for the Vehicle Routing Problem*, volume 161 of *Studies in Computational Intelligence*. Springer Berlin / Heidelberg, 2009. ISBN: 978-3-54085-151-6. doi:10.1007/978-3-540-85152-3.

[1636] Timothy Perkis. Stack based genetic programming. In *IEEE World Congress on Computational Intelligence*, volume 1, pages 148–153, 1994. In proceedings [1411]. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.53.2622 [accessed 2008-09-17].

[1637] Adrian Perrig and Dawn Xiaodong Song. A first step towards the automatic generation of security protocols. In *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS)*, pages 73–83. The Internet Society (ISOC), February 2–4, 2000, Catamaran Resort Hotel, San Diego, California, USA. ISBN: 1-8915-6207-X, 1-8915-6208-8. Online available at http://www.cs.berkeley.edu/~dawnsong/papers/apg.pdf and http://www.isoc.org/isoc/conferences/ndss/2000/proceedings/031.pdf [accessed 2008-06-23].

[1638] Alan Pétrowski. A clearing procedure as a niching method for genetic algorithms. In *Proceedings of IEEE International Conference on Evolutionary Computation*, pages 798–803, 1996. INSPEC Accession Number: 5375492. doi:10.1109/ICEC.1996.542703. In proceedings [1006]. Online available at http://sci2s.ugr.es/docencia/cursoMieres/clearing-96.pdf and http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.33.8027 [accessed 2008-12-16].

[1639] Alan Pétrowski. An efficient hierarchical clustering technique for speciation. Technical Report, Institut National des Télécommunications, 9 rue Charles Fourier, 91011 Evry Cedex, France, 1997. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.54.1131 [accessed 2009-02-28].

[1640] Patrick C. Phillips. The language of gene interaction. *Genetics*, 149(3):1167–1171, July 7, 1998. ISSN: 0016-6731. Online available at http://www.genetics.org/cgi/content/full/149/3/1167 [accessed 2008-03-02].

[1641] Samuel Pierre and Ali Elgibaoui. Improving communication network topologies using tabu search. In *LCN: Proceedings of the 22nd IEEE Conference on Local Computer Networks (LCN '97)*, pages 44–53. IEEE Computer Society, November 2–5 1997, Minneapolis, Minnesota, USA. ISBN: 0-8186-8141-1.

[1642] Samuel Pierre and Fabien Houéto. Assigning cells to switches in cellular mobile networks using taboo search. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 32(3):351–356, June 2002. doi:10.1109/TSMCB.2002.999810.

[1643] Samuel Pierre and Fabien Houéto. A tabu search approach for assigning cells to switches in cellular mobile networks. *Computer Communications*, 25(5):464–477, March 2002. doi:10.1016/S0140-3664(01)00371-1. Online available at http://dx.doi.org/10.1016/S0140-3664(01)00371-1 [accessed 2008-08-01].

[1644] Samuel Pierre and Gisèle Legault. An evolutionary approach for configuring economical packet switched computer networks. *AI in Engineering*, 10(2):127–134, 1996. doi:10.1016/0954-1810(95)00022-4. Online available at http://dx.doi.org/10.1016/0954-1810(95)00022-4 [accessed 2008-08-01].

[1645] Samuel Pierre and Gisèle Legault. A genetic algorithm for designing distributed computer network topologies. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 28(2):249–258, April 1998. doi:10.1109/3477.662766.

[1646] Massimo Pigliucci, Courtney J. Murren, and Carl D. Schlichting. Review article: Phenotypic plasticity in evolution. *Journal of Experimental Biology*, 209(12):2362–2367, June 2006. ISSN: 0022-0949 (Print) 1477-9145 (Online). doi:10.1242/jeb.02070.

Online available at http://jeb.biologists.org/cgi/reprint/209/12/2362.pdf [accessed 2008-09-11].

[1647] Ying ping Chen. *Extending the Scalability of Linkage Learning Genetic Algorithms – Theory & Practice*, volume 190 of *Studies in Fuzziness and Soft Computing*. Springer Berlin / Heidelberg, 2006. ISBN: 978-3-54028-459-8, 3-5402-8459-1. doi:10.1007/b102053. PhD Thesis, University of Illinois at Urbana-Champaign at Champaign, IL, USA, Adviser: David E. Goldberg, 2004, Order Number: AAI3130894. Partly online available at http://books.google.de/books?id=kKr3rKhPU7oC [accessed 2008-10-30].

[1648] Leonidas S. Pitsoulis and Mauricio G.C. Resende. Greedy randomized adaptive search procedures. In *Handbook of Applied Optimization*, chapter 3.6.5, pages 168–183. Oxford University Press, 2002. In collection [1613]. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.26.2170 and http://www.research.att.com/~mgcr/doc/grasp-hao.pdf [accessed 2008-11-06].

[1649] R. L. Plackett. Some theorems in least squares. *Biometrika*, 37(12):149–157, 1950. Online available at http://biomet.oxfordjournals.org/cgi/reprint/37/1-2/149 [accessed 2007-09-15].

[1650] Alexander Podlich. Intelligente planung und optimierung des güterverkehrs auf straße und schiene mit evolutionären algorithmen. Master's thesis, University of Kassel, FB-16, Distributed Systems Group, Wilhelmshöher Allee 73, 34121 Kassel, Germany, March 2009. Advisor: Thomas Weise, Christian Gorldt, Manfred Menze. Supervisor: Prof. Dr. Kurt Geihs, Prof. Dr.-Ing. B. Scholz-Reiter.

[1651] Hartmut Pohlheim. Geatbx introduction – evolutionary algorithms: Overview, methods and operators. Technical Report, http://www.GEATbx.com [accessed 2007-07-03], http://www.GEATbx.com [accessed 2007-07-03], November 2005. Documentation for GEATbx version 3.7 (Genetic and Evolutionary Algorithm Toolbox for use with Matlab). Online available at http://www.geatbx.com/download/GEATbx_Intro_Algorithmen_v38.pdf [accessed 2007-07-03].

[1652] Wolfgang Polasek. *Schließende Statistik – Einführung in die Schätz- und Testtheorie für Wirtschaftswissenschaftler*. Springer-Lehrbuch. Springer, 1997. ISBN: 3-5406-1731-0, 978-3-54061-731-0.

[1653] Riccardo Poli. Evolution of recursive transition networks for natural language recognition with parallel distributed genetic programming. Technical Report CSRP-96-19, School of Computer Science, The University of Birmingham, Birmingham B15 2TT, United Kingdom, December 1996. Online available at http://citeseer.ist.psu.edu/90834.html [accessed 2007-11-04]. See also [1656].

[1654] Riccardo Poli. Parallel distributed genetic programming. Technical Report CSRP-96-15, School of Computer Science, The University of Birmingham, Birmingham B15 2TT, United Kingdom, September 1996. See [1659]. Online available at http://citeseer.ist.psu.edu/86223.html [accessed 2007-11-04].

[1655] Riccardo Poli. Some steps towards a form of parallel distributed genetic programming. In *The 1st Online Workshop on Soft Computing (WSC1)*. Nagoya University, Japan, August 19-30, 1996. Originally published as technical report, CSRP-96-14, University of Birmingham, School of Computer Science, 1996. Online available at http://cswww.essex.ac.uk/staff/poli/papers/Poli-WSC1-1996.pdf and http://citeseer.ist.psu.edu/156274.html [accessed 2007-11-04].

[1656] Riccardo Poli. Evolution of recursive transistion networks for natural language recognition with parallel distributed genetic programming. In *Proceedings of the Workshop on Artificial Intelligence and Simulation of Behaviour (AISB), International Workshop on Evolutionary Computing, Selected Papers*, pages 163–177, 1997. In proceedings [447]. Online available at http://cswww.essex.ac.uk/staff/rpoli/papers/Poli-AISB-1997.pdf and http://citeseer.ist.psu.edu/355686.html [accessed 2007-11-04]. See also [1653].

[1657] Riccardo Poli. Discovery of symbolic, neuro-symbolic and neural networks with parallel distributed genetic programming. In *3rd International Conference on Artificial Neural Networks and Genetic Algorithms, ICANNGA'97*, 1997. In proceedings [1902]. Online available at http://cswww.essex.ac.uk/staff/rpoli/papers/Poli-ICANNGA1997.pdf and http://citeseer.ist.psu.edu/poli97discovery.html [accessed 2007-11-04].

[1658] Riccardo Poli. Evolution of graph-like programs with parallel distributed genetic programming. In *Genetic Algorithms: Proceedings of the Seventh International Conference*, pages 346–353, 1997. In proceedings [98]. Online available at http://citeseer.ist.psu.edu/578015.html and http://cswww.essex.ac.uk/staff/poli/papers/Poli-ICGA1997-PDGP.pdf [accessed 2007-11-04].

[1659] Riccardo Poli. Parallel distributed genetic programming. In *New Ideas in Optimization*, pages 403–432. McGraw-Hill Education, 1999. In collection [448], see also [1654]. Online available at http://cswww.essex.ac.uk/staff/rpoli/papers/Poli-NIO-1999-PDGP.pdf [accessed 2007-11-04].

[1660] Riccardo Poli. A simple but theoretically-motivated method to control bloat in genetic programming. In *Proceedings of 6th European Conference on Genetic Programming, EuroGP 2003*, pages 204–217, 2003. In proceedings [1786].

[1661] Riccardo Poli and William B. Langdon. A new schema theory for genetic programming with one-point crossover and point mutation. In *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 278–285, 1997. In proceedings [1208]. Online available at http://citeseer.ist.psu.edu/327495.html [accessed 2008-06-14]. See also [1662].

[1662] Riccardo Poli and William B. Langdon. A new schema theory for genetic programming with one-point crossover and point mutation. Technical Report CSRP-97-3, The University of Birmingham, School of Computer Science, B15 2TT, UK, January 1997. Revised August 1997. Online available at http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/poli_1997_schemaTR.html and ftp://ftp.cs.bham.ac.uk/pub/tech-reports/1997/CSRP-97-03.ps.gz [accessed 2008-06-17]. See also [1661].

[1663] Riccardo Poli, William B. Langdon, Marc Schoenauer, Terry Fogarty, and Wolfgang Banzhaf, editors. *Late Breaking Papers at EuroGP'98: The First European Workshop on Genetic Programming*, April 14-15, 1998, Paris, France. Distributed at the workshop. See alse [141].

[1664] Riccardo Poli, Peter Nordin, William B. Langdon, and Terence C. Fogarty, editors. *Proceedings of the Second European Workshop on Genetic Programming*, volume 1598/1999 of *Lecture Notes in Computer Science (LNCS)*, May 26-27, 1999, Göteborg, Sweden. Springer Berlin/Heidelberg. ISBN: 3-5406-5899-8.

[1665] Riccardo Poli, Hans-Michael Voigt, Stefano Cagnoni, David Corne, George D. Smith, and Terence C. Fogarty, editors. *Proceedings of the First European Workshops on Evolutionary Image Analysis, Signal Processing and Telecommunications, EvoIASP'99 and EuroEcTel'99*, volume 1596/1999 of *Lecture Notes in Computer Science (LNCS)*, May 26–27, 1999, Göteborg, Sweden. Springer. ISBN: 3-5406-5837-8, 978-3-54065-837-5. doi:10.1007/10704703. Partly online available at http://books.google.de/books?id=CYDLxvi0-SYC [accessed 2008-09-16].

[1666] Riccardo Poli, Wolfgang Banzhaf, William B. Langdon, Julian Francis Miller, Peter Nordin, and Terence C. Fogarty, editors. *Proceedings of the European Conference on Genetic Programming*, volume 1802/2000 of *Lecture Notes in Computer Science (LNCS)*, April 15-16, 2000, Edinburgh, Scotland, UK. Springer Berlin/Heidelberg. ISBN: 3-5406-7339-3. doi:10.1007/b75085. Partly online available at http://books.google.de/books?id=KhaR0XZHedEC [accessed 2008-09-24].

[1667] Riccardo Poli, William B. Langdon, and Nicholas Freitag McPhee. *A Field Guide to Genetic Programming*. Published via http://lulu.com [accessed 2008-03-26] and freely available at http://www.gp-field-guide.org.uk [accessed 2008-03-26], 2008. ISBN:

978-1-40920-073-4. Online available at http://www.lulu.com/items/volume_63/2167000/2167025/2/print/book.pdf and http://www.gp-field-guide.org.uk/ [accessed 2008-03-27]. (With contributions by John R. Koza).

[1668] H. Vincent Poor. *An Introduction to Signal Detection and Estimation*. Springer, second edition, June 2005. ISBN: 978-0-38794-173-8.

[1669] Robert L. Popp, David J. Montana, Richard R. Gassner, Gordon Vidaver, and Suraj Iyer. Automated hardware design using genetic programming, VHDL, and FPGAs. In *IEEE International Conference on Systems, Man, and Cybernetics*, volume 3, pages 2184–2189. IEEE, October 11–14, 1998, San Diego, CA USA. INSPEC Accession Number: 6189463.

[1670] V. William Porto, N. Saravanan, D. Waagen, and Ágoston E. Eiben, editors. *Proceedings of the 7th International Conference on Evolutionary Programming VII, EP98, in cooperation with IEEE Neural Networks Council*, volume 1447/1998 of *Lecture Notes in Computer Science (LNCS)*, March 25–27, 1998, Mission Valley Marriott, San Diego, California, USA. Springer Verlag, Heidelberg, Germany. ISBN: 978-3-54064-891-8.

[1671] Vincent W. Porto, David B. Fogel, and Lawrence Jerome Fogel. Alternative neural network training methods. *IEEE Expert: Intelligent Systems and Their Applications*, 10(3):16–22, 1995. ISSN: 0885-9000. doi:10.1109/64.393138.

[1672] Emil Leon Post. Formal reductions of the general combinatorial decision problem. *American Journal of Mathematics*, 65(2):197–215, 1943, Baltimore, MD, USA. ISSN: 0002-9327, 1080-6377. doi:10.2307/2371809.

[1673] Mitchell A. Potter and Kenneth Alan De Jong. A cooperative coevolutionary approach to function optimization. In *Parallel Problem Solving from Nature - PPSN III, International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature*, pages 249–257, 1994. In proceedings [492]. Online available at http://citeseer.ist.psu.edu/potter94cooperative.html [accessed 2007-08-24].

[1674] Jean-Yves Potvin, Patrick Soriano, and Maxime Vallée. Generating trading rules on the stock markets with genetic programming. *Computers & Operations Research*, 31 (7):1033–1047, June 2004. ISSN: 0305-0548. doi:10.1016/S0305-0548(03)00063-7.

[1675] William H. Press, Saul A. Teukolsky, William T. Vettering, and Brian P. Flannery. *Numerical Recipes in C++. Example Book. The Art of Scientific Computing*, chapter 10. Cambridge University Press, second edition, February 7, 2002. ISBN: 0-5217-5034-2, 978-0-52175-034-9. Partly online available at http://books.google.de/books?id=gwijz-OyIYEC [accessed 2008-06-14].

[1676] Kenneth V. Price, Rainer M. Storn, and Jouni A. Lampinen. *Differential Evolution – A Practical Approach to Global Optimization*. Natural Computing Series. Springer, New York, October 2005. ISBN: 978-3-54020-950-8. Series editors: G. Rozenberg, Thomas Bäck, Ágoston E. Eiben, J.N. Kok, and H.P. Spaink. Partly online available at http://books.google.de/books?id=S67vX-KqVqUC [accessed 2008-07-23].

[1677] Adam Prügel-Bennett and Alex Rogers. Modelling ga dynamics. In *Theoretical Aspects of Evolutionary Computing*, pages 59–86. Springer, 2001. Original: 1999, Online available at http://eprints.ecs.soton.ac.uk/13205/ [accessed 2007-07-28]. In collection [1083].

[1678] William F. Punch, Douglas Zongker, and Erik D. Goodman. The royal tree problem, a benchmark for single and multiple population genetic programming. In *Advances in Genetic Programming 2*, pages 299–316. MIT Press, 1996. In collection [61]. Online available at http://citeseer.ist.psu.edu/147908.html [accessed 2007-10-14].

[1679] Robin C. Purshouse and Peter J. Fleming. The Multi-Objective Genetic Algorithm Applied to Benchmark Problems – An Analysis. Research Report 796, Department of Automatic Control and Systems Engineering, University of Sheffield, Sheffield, S1 3JD, UK, August 2001. Online available at http://citeseer.ist.psu.edu/

499210.html and http://www.lania.mx/~ccoello/EMOO/purshouse01.pdf.gz [accessed 2008-04-16].

Q

[1680] Yuhui Qiu and Fang Wang. Improving particle swarm optimizer using the nonlinear simplex method at late stage. In Richard T. Hurley and Wenying Feng, editors, *Proceedings of the ISCA, the 14th International Conference on Intelligent and Adaptive Systems and Software Engineering*, pages 25–30. ISCA, July 20–22, 2005, Novotel Toronto Centre, Toronto, Canada. ISBN: 1-8808-4355-2.

[1681] D. Quagliarella, J. Périaux, C. Poloni, and G. Winter, editors. *Genetic Algorithms and Evolution Strategy in Engineering and Computer Science: Recent Advances and Industrial Applications – Proceedings of the 2nd European Short Course on Genetic Algorithms and Evolution Strategies, EUROGEN 1997*, Recent Advances in Industrial Applications, November 28–December 4, 1997, Triest, Italy. Wiley & Sons. ISBN: 978-0-47197-710-0.

[1682] R. J. Quick, Victor J. Rayward-Smith, and George D. Smith. The royal road functions: description, intent and experimentation. In *Evolutionary Computing – Selected Papers from the AISB Workshop*, pages 223–235, 1996. doi:10.1007/BFb0032786. In proceedings [695]. Partly online available at http://books.google.de/books?id=cJi2MRos9cMC [accessed 2008-04-09].

[1683] Alejandro Quintero and Samuel Pierre. Assigning cells to switches in cellular mobile networks: a comparative study. *Computer Communications*, 26(9):950–960, June 2003. doi:10.1016/S0140-3664(02)00224-4. Online available at http://dx.doi.org/10.1016/S0140-3664(02)00224-4 [accessed 2008-08-01].

[1684] Alejandro Quintero and Samuel Pierre. Evolutionary approach to optimize the assignment of cells to switches in personal communication networks. *Computer Communications*, 26(9):927–938, June 2, 2003. doi:10.1016/S0140-3664(02)00238-4. Online available at http://dx.doi.org/10.1016/S0140-3664(02)00238-4 [accessed 2008-08-01].

[1685] Alejandro Quintero and Samuel Pierre. Sequential and multi-population memetic algorithms for assigning cells to switches in mobile networks. *Computer Networks*, 43 (3):247–261, October 22, 2003. doi:10.1016/S1389-1286(03)00270-6. Online available at http://dx.doi.org/10.1016/S1389-1286(03)00270-6 [accessed 2008-08-01].

[1686] Mohammad Adil Qureshi. Evolving agents. In *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 369–374, 1996. In proceedings [1207] and also [1687]. Online available at http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/AQ.gp96.ps.gz [accessed 2007-09-17].

[1687] Mohammad Adil Qureshi. Evolving agents. Research Note RN/96/4, UCL, Gower Street, London, WC1E 6BT, UK, January 1996. Online available at http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/AQ.gp96.ps.gz and http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/qureshi_1996_eaRN.html [accessed 2008-09-02]. See also [1686].

[1688] Mohammad Adil Qureshi. *The Evolution of Agents*. PhD thesis, University College, London, London, UK, July 2001. Online available at http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/qureshi_thesis.html and http://citeseer.ist.psu.edu/759376.html [accessed 2007-09-14]. Supervisor: Jon Crowcroft.

R

[1689] Pablo Manuel Rabanal Basalo, Ismael Rodríguez Laguna, and Fernando Rubio Díez. Using river formation dynamics to design heuristic algorithms. In Selim G. Akl, Cristian S. Calude, Michael J. Dinneen, Grzegorz Rozenberg, and Todd Wareham, editors,

Proceedings of the 6th International Conference on Unconventional Computation, UC 2007, volume 4618/2007 of *Lecture Notes in Computer Science (LNCS)*, pages 163–177. Springer, August 13–17, 2007, Kingston, Canada. ISBN: 978-3-54073-553-3. doi:10.1007/978-3-540-73554-0_16.

[1690] Pablo Manuel Rabanal Basalo, Ismael Rodríguez Laguna, and Fernando Rubio Díez. Finding minimum spanning/distance trees by using river formation dynamics. In *Sixth International Conference on Ant Colony Optimization and Swarm Intelligence – ANTS 2008*, 2008. In proceedings [1947].

[1691] Nicholas J. Radcliffe. Equivalence class analysis of genetic algorithms. *Complex Systems*, 5(2):183–205, 1991. Online available at http://citeseer.ist.psu.edu/608745.html [accessed 2007-07-28].

[1692] Nicholas J. Radcliffe. Non-linear genetic representations. In *Parallel problem solving from nature 2*, pages 259–268, 1992. In proceedings [1357]. Online available at http://citeseer.ist.psu.edu/radcliffe92nonlinear.html and http://users.breathe.com/njr/papers/ppsn92.pdf [accessed 2007-09-05].

[1693] Nicholas J. Radcliffe and Patrick D. Surry. Formal memetic algorithms. In *Proceedings of the Workshop on Artificial Intelligence and Simulation of Behaviour (AISB), International Workshop on Evolutionary Computing, Selected Papers*, pages 1–16, 1994. doi:10.1007/3-540-58483-8_1. In proceedings [693]. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.38.9885 [accessed 2008-09-10].

[1694] Nicholas J. Radcliffe and Patrick D. Surry. Fundamental limitations on search algorithms: Evolutionary computing in perspective. In Jan van Leeuwen, editor, *Computer Science Today – Recent Trends and Developments*, volume 1000 of *Lecture Notes in Computer Science (LNCS)*, pages 275–291. Springer-Verlag, 1995. ISBN: 978-3-54060-105-0, 3-5406-0105-8. doi:10.1007/BFb0015249. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.49.2376 [accessed 2008-08-12].

[1695] Nicholas J. Radcliffe and Patrick David Surry. Fitness variance of formae and performance prediction. In *Foundations of Genetic Algorithms 3*, pages 51–72, 1994. In proceedings [2214]. Online available at http://citeseer.comp.nus.edu.sg/radcliffe94fitness.html [accessed 2007-07-28].

[1696] Nicolas J. Radcliffe. The algebra of genetic algorithms. *Annals of Mathematics and Artificial Intelligence*, 10(4):339–384, December 1994. ISSN: 1012-2443 (print version), 1573-7470 (electronic version). doi:10.1007/BF01531276. Online available at http://www.springerlink.com/content/w20613t556788tr0/fulltext.pdf [accessed 2008-08-10]. Also technical report TR92-11, 1992 from Edinburgh Parallel Computing Centre, University of Edinburgh, King's Buildings, EH9 3JZ, Scotland. Online available at http://citeseer.ist.psu.edu/radcliffe94algebra.html and http://users.breathe.com/njr/formaPapers.html [accessed 2007-07-28].

[1697] Cauligi S. Raghavendra, Krishna M. Sivalingam, and Taieb Znati, editors. *Wireless Sensor Networks (ERCOFTAC)*. Springer Netherlands, second edition, June 1, 2004. ISBN: 978-1-40207-883-5.

[1698] Mohammad Rahimi, Hardik Shah, Gaurav Sukhatme, John Heidemann, and Deborah Estrin. Studying the feasibility of energy harvesting in a mobile sensor network. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 19–24, May 2003. IEEE, Taipai, Taiwan. Online available at http://www.isi.edu/~johnh/PAPERS/Rahimi03a.pdf and http://www.citeulike.org/user/cri06/article/935917 [accessed 2007-08-01].

[1699] Günther R. Raidl. A hybrid gp approach for numerically robust symbolic regression. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 323–328, 22–25 1998. In proceedings [1209]. Online available at http://citeseer.ist.psu.edu/549723.html and http://www.ads.tuwien.ac.at/publications/bib/pdf/raidl-98c.pdf [accessed 2007-09-14].

[1700] Günther R. Raidl and Jens Gottlieb, editors. *Proceedings of the 5th European Conference on Evolutionary Computation in Combinatorial Optimization, EvoCOP 2005*, volume 3448/2005 of *Lecture Notes in Computer Science (LNCS)*, March 30–April 1, 2005, Lausanne, Switzerland. Springer. ISBN: 3-5402-5337-8.

[1701] Günther R. Raidl, Jean-Arcady Meyer, Martin Middendorf, Stefano Cagnoni, Juan J. Romero Cardalda, David Corne, Jens Gottlieb, Agnès Guillot, Emma Hart, Colin G. Johnson, and Elena Marchiori, editors. *Applications of Evolutionary Computing, Proceedings of EvoWorkshop 2003: EvoBIO, EvoCOP, EvoIASP, EvoMUSART, EvoROB, and EvoSTIM*, volume 2611/2003 of *Lecture Notes in Computer Science (LNCS)*, April 14–16, 2003, University of Essex,Essex, UK. Springer, Berlin / Heidelberg. ISBN: 3-5400-0976-0.

[1702] Günther R. Raidl, Stefano Cagnoni, Jürgen Branke, David Corne, Rolf Drechsler, Yaochu Jin, Colin G. Johnson, Penousal Machado, Elena Marchiori, Franz Rothlauf, George D. Smith, and Giovanni Squillero, editors. *Applications of Evolutionary Computing, Proceedings of EvoWorkshops 2004: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoMUSART, and EvoSTOC*, volume 3005/2004 of *Lecture Notes in Computer Science (LNCS)*, April 5–7, 2004, Coimbra, Portugal. Springer, Berlin / Heidelberg. ISBN: 3-5402-1378-3, 978-3-54021-378-9. doi:10.1007/b96500.

[1703] Anca L. Ralescu and James G. Shanahan, editors. *Fuzzy Logic in Artificial Intelligence, IJCAI'97 Selected and Invited Workshop Papers*, volume 1566 of *Lecture Notes in Computer Science (LNCS)*, 1999, Nagoya, Japan. Springer. ISBN: 3-5406-6374-6. See also [1455, 1456].

[1704] Owen Rambow and Aravind K. Joshi. A formal look at dependency grammars and phrase-structure grammars, with special consideration of word-order phenomena. *Meaning-Text Theory*, 1997. Online available at http://arxiv.org/abs/cmp-lg/9410007v1 [accessed 2007-09-15].

[1705] William Rand, Sevan G. Ficici, and Rick Riolo, editors. *Evolutionary Computation and Multi-Agent Systems and Simulation (ECoMASS) Workshop 2008*, July 12, 2008, Renaissance Atlanta Hotel Downtown, 590 West Peachtree Street NW, Atlanta, Georgia 30308 USA. ACM Press, New York, NY, USA. ISBN: 978-1-60558-131-6. Part of GECCO 2008, see also [1117] and http://www.eecs.harvard.edu/~sevan/ecomass08/ [accessed 2008-07-21].

[1706] Alain T. Rappaport and Reid G. Smith, editors. *Proceedings of the The Second Conference on Innovative Applications of Artificial Intelligence (IAAI-90)*, May 1–3, 1990, Washington, DC, USA. AAAI. ISBN: 0-2626-8068-8. Published in 1991. See http://www.aaai.org/Conferences/IAAI/iaai90.php [accessed 2007-09-06].

[1707] Theodore S. Rappaport. *Wireless Communications: Principles and Practice*. Prentice Hall PTR, 2001, (2nd ed: January 2002). ISBN: 978-0-13042-232-3, 0-1304-2232-0. Partly online available at http://books.google.com/books?id=8ZXdHQAACAAJ [accessed 2008-07-27].

[1708] Thomas Kiel Rasmussen and Thiemo Krink. Improved hidden markov model training for multiple sequence alignment by a particle swarm optimization-evolutionary algorithm hybrid. *BioSystems*, 72(1–2):5–17, November 2003. Online available at http://dx.doi.org/10.1016/S0303-2647(03)00131-X [accessed 2007-08-21].

[1709] Leonard A. Rastrigin. The convergence of the random search method in the extremal control of many-parameter system. *Automation and Remote Control*, 24:1337–1342, 1963.

[1710] Victor J. Rayward-Smith. A unified approach to tabu search, simulated annealing and genetic algorithms. In Victor J. Rayward-Smith, editor, *Applications of Modern Heuristic Methods – Proceedings of the UNICOM Seminar on Adaptive Computing and Information Processing*, volume I, pages 55–78, January 25–27, 1994, Brunel University Conference Centre, London, UK. Alfred Waller Ltd / Nelson

Thornes Ltd / Unicom Seminars Ltd, Henley-on-Thames, UK. ISBN: 1-8724-7428-4, 978-1-87247-428-1.

[1711] Victor J. Rayward-Smith, Ibrahim H. Osman, Colin R. Reeves, and George D. Smith, editors. *Modern Heuristic Search Methods*. Wiley, December 1996. ISBN: 978-0-47196-280-9.

[1712] Ingo Rechenberg. *Cybernetic Solution Path of an Experimental Problem*. Royal Aircraft Establishment, August 1965. Library Translation 1122, Farnborough. Reprinted in [696].

[1713] Ingo Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog Verlag, Stuttgart, Germany, 1973. ISBN: 978-3-77280-374-1. His dissertation from 1970.

[1714] Ingo Rechenberg. *Evolutionsstrategie '94*, volume 1 of *Werkstatt Bionik und Evolutionstechnik*. Frommann-Holzboog Verlag, Stuttgart, Germany, September 1994. ISBN: 978-3-77281-642-0.

[1715] Lynne M. Reder, editor. *Implicit Memory and Metacognition*. Carnegie Mellon Symposia on Cognition. Lawrence Erlbaum (Routledge, Tailor & Francis Group), 1996. ISBN: 0-8058-1860-X, 978-0-80581-860-4, 0-8058-1859-6, 978-0-80581-859-8. Partly online available at http://books.google.de/books?id=g5Bi3dIrpjkC [accessed 2008-12-01].

[1716] Colin R. Reeves, editor. *Modern Heuristic Techniques for Combinatorial Problems*. John Wiley & Sons, Inc., Blackwell Scientific Publishing, Halsted Press, New York, NY, USA / Oxford, England, April 1993. ISBN: 0-4702-2079-1, 978-0-47022-079-5.

[1717] Colin R. Reeves and Christine Wright. An experimental design perspective on genetic algorithms. In *Foundations of Genetic Algorithms 3*, pages 7–22, 1994. In proceedings [2214]. Online available at http://citeseer.ist.psu.edu/reeves95experimental.html [accessed 2008-07-20].

[1718] Christian M. Reidys and Peter F. Stadler. Neutrality in fitness landscapes. *Applied Mathematics and Computation*, 117(2–3):321–350, January 25, 2001. ISSN: 0096-3003. doi:10.1016/S0096-3003(99)00166-6. Online available at http://dx.doi.org/10.1016/S0096-3003(99)00166-6 and http://citeseer.ist.psu.edu/reidys98neutrality.html [accessed 2008-06-01].

[1719] Jean-Michel Renders and Hugues Bersini. Hybridizing genetic algorithms with hill-climbing methods for global optimization: two possible ways. In *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, pages 312–317, 1994. INSPEC Accession Number: 4818657. doi:10.1109/ICEC.1994.349948. In proceedings [1411].

[1720] Alfréd Rényi. *Probability Theory*. Dover Publications, May 2007. ISBN: 0-4864-5867-9, 978-0-48645-867-0.

[1721] Mauricio G. C. Resende and Jorge Pinho de Sousa, editors. *4th Metaheuristics International Conference (MIC2001) – Metaheuristics: Computer Decision-Making*, volume 86 of *Applied Optimization*, July 16–20, 2001, Porto, Portugal. Kluwer Academic Publishers, Boston, USA. ISBN: 1-4020-7653-3. Published in November 30, 2003. See http://paginas.fe.up.pt/~gauti/MIC2001/ [accessed 2007-09-12].

[1722] Mauricio G.C. Resende and Celso C. Ribeiro. Greedy randomized adaptive search procedures. In *Handbook of Metaheuristics*, chapter 8, pages 219–242. Kluwer Academic Publishers / Springer, 2003. doi:10.1007/0-306-48056-5_8. In collection [813] and also AT&T Labs Research Technical Report, September 2001, revision 2, August 29, 2002. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.18.2716 and http://www.research.att.com/~mgcr/doc/sgrasp-hmetah.pdf [accessed 2008-11-06].

[1723] Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. *SIGGRAPH Computer Graphics*, 4(10):25–34, July 1987. ISSN: 0097-8930. Online available at http://graphics.ucmerced.edu/~mkallmann/

courses/papers/Reynolds1987-flocks.pdf and http://www.cs.toronto.edu/~dt/siggraph97-course/cwr87/ [accessed 2008-07-27]. See also [1724].

[1724] Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *SIGGRAPH'87, Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 25–34, July 27–31, 1987, Anaheim, California, USA. Association for Computing Machinery (ACM), New York, NY, USA. ISBN: 0-8979-1227-6. doi:10.1145/37401.37406. Online available at http://citeseer.ist.psu.edu/reynolds-flocks.html and http://doi.acm.org/10.1145/37401.37406 [accessed 2008-07-27]. See also [1723].

[1725] Bernadete Ribeiro, Rudolf F. Albrecht, Andrej Dobnikar, David W. Pearson, and Nigel C. Steele, editors. *Proceedings of the 7^{th} International Conference on Adaptive and Natural Computing Algorithms, ICANNGA 2005*, March 21–23, 2005, Department of Informatics Engineering, Faculty of Science and Technology, University of Coimbra, Coimbra, Portugal. Springer, Wien. ISBN: 978-3-21124-934-5, 978-3-21127-389-0. see http://icannga05.dei.uc.pt/ [accessed 2007-08-31].

[1726] Celso C. Ribeiro and Pierre Hansen, editors. *3rd Metaheuristics International Conference (MIC'99) – Essays and Surveys in Metaheuristics*, volume 15 of *Operations Research/Computer Science Interfaces*, July 19–23, 1999, Angra dos Reis, Brazil. Kluwer Academic Publishers, Boston, USA. ISBN: 0-7923-7520-3. Published in 2001.

[1727] John A. Rice. *Mathematical Statistics and Data Analysis*. Statistics. Duxbury Press, third edition, April 2006. ISBN: 978-0-53439-942-9, 978-0-53420-934-6.

[1728] Jon T. Richardson, Mark R. Palmer, Gunar E. Liepins, and Mike R. Hilliard. Some guidelines for genetic algorithms with penalty functions. In *Proceedings of the third International Conference on Genetic Algorithms ICGA*, pages 191–197, 1989. In proceedings [1820].

[1729] Hendrik Richter. Behavior of evolutionary algorithms in chaotically changing fitness landscapes. In *Proceedings of 8th International Conference on Parallel Problem Solving from Nature, PPSN VIII*, pages 111–120, 2004. In proceedings [2285].

[1730] Mark Ridley. *Evolution*. Blackwell Publishing Limited, eighth (october 1, 2003) edition, 1996. ISBN: 978-1-40510-345-9.

[1731] John Riedl and Randall W. Hill Jr., editors. *Proceedings of the Fifteenth Conference on Innovative Applications of Artificial Intelligence*, August 12–14, 2003, Acapulco, México. AAAI. ISBN: 1-5773-5188-6. See http://www.aaai.org/Conferences/IAAI/iaai03.php [accessed 2007-09-06].

[1732] Rupert J. Riedl. A systems-analytical approach to macroevolutionary phenomena. *Quarterly Review of Biology*, (52):351–370, 1977.

[1733] Bernhard Riemann. Über die anzahl der primzahlen unter einer gegebenen größe. *Monatsberichte der Königlich Preußischen Akadademie der Wissenschaften zu Berlin*, November 1859. Online available at http://www.claymath.org/millennium/Riemann_Hypothesis/1859_manuscript/zeta.pdf and http://www.maths.tcd.ie/pub/HistMath/People/Riemann/Zeta/EZeta.pdf [accessed 2008-08-24].

[1734] Rick Riolo and Bill Worzel, editors. *Genetic Programming Theory and Practice, Proceedings of the Genetic Programming Theory Practice 2003 Workshop (GPTP-2003)*, Genetic Programming Series, May 15-17, 2003, The Center for the Study of Complex Systems (CSCS), University of Michigan, Ann Arbor, UMichigan, SA. Springer, Kluwer Publishers, Boston, MA. ISBN: 1-4020-7581-2, 978-1-40207-581-0. See http://www.cscs.umich.edu/gptp-workshops/gptp2003/ [accessed 2007-09-28].

[1735] Rick Riolo, Terence Soul, and Bill Worzel, editors. *Genetic Programming Theory and Practice IV, Proceedings of the Genetic Programming Theory Practice 2006 Workshop (GPTP-2006)*, Genetic and Evolutionary Computation, May 11-13, 2006, The Center for the Study of Complex Systems (CSCS), University of Michigan, Ann Arbor, Michigan, USA. Springer. ISBN: 978-0-38733-375-5. See http://www.cscs.umich.edu/gptp-workshops/gptp2006/ [accessed 2007-09-28].

[1736] Rick L. Riolo. Letseq: An implementation of the cfs-c classifier system in a task-domain that involves learning to predict letter. Technical Report, Logic of computers group, Division of computer science and engineering, University of Michigan, 1986, revised 1988. Online available at http://citeseer.ist.psu.edu/345595.html [accessed 2007-09-11].

[1737] Rick L. Riolo. Bucket brigade performance: Ii. default hierarchies. In *Proceedings of the Second International Conference on Genetic algorithms and their application*, pages 196–201, 1987. In proceedings [857].

[1738] Rick L. Riolo. Bucket brigade performance: I. long sequences of classifiers. In *Proceedings of the Second International Conference on Genetic algorithms and their application*, pages 184–195, 1987. In proceedings [857].

[1739] Rick L. Riolo. The emergence of default hierarchies in learning classifier systems. In *Proceedings of the 3rd International Conference on Genetic Algorithms (ICGA89)*, pages 322–327, 1989. In proceedings [1820].

[1740] Matei Ripeanu, Ian Foster, and Adriana Iamnitchi. Mapping the gnutella network: Properties of large-scale peer-to-peer systems and implications for system design. *IEEE Internet Computing Journal*, 6(1):50–57, February 2002. Online available at http://people.cs.uchicago.edu/~matei/papers/ic.pdf and http://citeseer.ist.psu.edu/ripeanu02mapping.html [accessed 2008-06-24].

[1741] Irina Rish. An empirical study of the naive bayes classifier. In *Proceedings of IJCAI-01 workshop on Empirical Methods in AI, International Joint Conference on Artificial Intelligence*, pages 41–46. American Association for Artificial Intelligence, 2001. In proceedings [1507]. Online available at http://www.cc.gatech.edu/~isbell/classes/reading/papers/Rish.pdf [accessed 2007-08-11].

[1742] Ron Rivest. S-expressions, May 4, 1997. draft-rivest-sexp-00.txt. Online available at http://people.csail.mit.edu/rivest/Sexp.txt [accessed 2007-07-03]. Network Working Group, Internet Draft, Expires November 4, 1997.

[1743] Herbert Robbins and Sutton Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22(3):400–407, September 1951. doi:10.1214/aoms/1177729586. Online available at http://projecteuclid.org/euclid.aoms/1177729586 [accessed 2008-10-11], Mathematical Reviews number (MathSciNet): MR42668, Zentralblatt MATH identifier: 0054.05901.

[1744] Christian P. Robert and George Casella. *Monte Carlo Statistical Methods*. Springer Texts in Statistics. Springer-Verlag New York, Inc., Secaucus, NJ, USA, second edition, 2004. ISBN: 0-3872-1239-6, 978-0-38721-239-5. Partly online available at http://books.google.de/books?id=HfhGAxn5GugC [accessed 2008-08-16].

[1745] Alan Robinson. Genetic programming: Theory, implementation, and the evolution of unconstrained solutions. Master's thesis, Cognitive Science, Hampshire College, Amherst, MA 01002, May 2003. Division III Thesis. Committee Lee Spector, Jaime Davila, Mark Feinstein. Online available at http://hampshire.edu/lspector/robinson-div3.pdf and http://citeseer.ist.psu.edu/498673.html [accessed 2007-12-25].

[1746] Alex Rogers and Adam Prügel-Bennett. Modelling the dynamics of a steady-state genetic algorithm. In *Foundations of Genetic Algorithms 5*, pages 57–68, 1998. In proceedings [139]. Online available at http://eprints.ecs.soton.ac.uk/451/02/FOGA.ps and http://citeseer.ist.psu.edu/rogers99modelling.html [accessed 2007-08-28].

[1747] Daniel S. Rokhsar, Philip Warren Anderson, and Daniel L. Stein. Self-organization in prebiological systems: Simulations of a model for the origin of genetic information. *Journal of Molecular Evolution*, 23(2):119–126, June 1986. ISSN: 0022-2844 (print), 1432-1432 (online). doi:10.1007/BF02099906. Online available at http://www.springerlink.com/content/m268540162062735/fulltext.pdf [accessed 2008-07-02].

[1748] Dumitru Roman, Uwe Keller, and Holger Lausen. *WSMO – Web Service Modeling Ontology*. Digital Enterprise Research Institute (DERI), February 2004. Online available at http://www.wsmo.org/2004/d2/v0.1/20040214/ [accessed 2007-09-02]. See also http://www.wsmo.org/ [accessed 2007-09-02] and [1749].

[1749] Dumitru Roman, Uwe Keller, Holger Lausen, Jos de Bruijn, Ruben Lara, Michael Stollberg, Axel Polleres, Cristina Feier, Christoph Bussler, and Dieter Fensel. Web service modeling ontology. *Applied Ontology*, 1(1):77–106, 2005. See also http://www.wsmo.org/ [accessed 2007-09-02] and [1748].

[1750] Simon Ronald. Preventing diversity loss in a routing genetic algorithm with hash tagging. *Complexity International*, 2, April 1995. ISSN: 1320-0683. Online available at http://www.complexity.org.au/ci/vol02/sr_hash/ [accessed 2007-07-28].

[1751] Simon Ronald. *Genetic Algorithms and Permutation-Encoded Problems. Diversity Preservation and a Study of Multimodality*. PhD thesis, University Of South Australia. Department of Computer and Information Science, 1996.

[1752] Simon Ronald. Robust encodings in genetic algorithms: A survey of encoding issues. In *IEEE Forth International Conference on Evolutionary Computation (IEEE/ICEC'97)*, pages 43–48, 1997. doi:10.1109/ICEC.1997.592265. In proceedings [106].

[1753] Min rong Chen, Yong zai Lu, and Gen ke Yang. Multiobjective extremal optimization with applications to engineering design. *Journal of Zhejiang University – Science A*, 8(12):1905–1911, November 2007. ISSN: 1673-565X (Print) 1862-1775 (Online). doi:10.1631/jzus.2007.A1905.

[1754] Wan rong Jih and Jane Yung jen Hsu. Dynamic vehicle routing using hybrid genetic algorithms. In *IEEE International Conference on Robotics and Automation*, volume 1, pages 453–458, May 10–15, 1999, Detroid, Michigan. ISBN: 0-7803-5180-0. INSPEC Accession Number: 6345736. doi:10.1109/ROBOT.1999.770019. Online available at http://neo.lcc.uma.es/radi-aeb/WebVRP/data/articles/DVRP.pdf [accessed 2008-10-27].

[1755] Peter Roosen and Fred Meyer. Determination of chemical equilibria by means of an evolution strategy. In *PPSN-II, Parallel problem solving from nature 2*, pages 411–420, 1992. In proceedings [1357].

[1756] Justinian Rosca. Generality versus size in genetic programming. In *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 381–387, 1996. In proceedings [1207]. Online available at http://citeseer.ist.psu.edu/75165.html and ftp://ftp.cs.rochester.edu/pub/u/rosca/gp/96.gp.ps.gz [accessed 2007-09-07].

[1757] Justinian P. Rosca. Proceedings of the workshop on genetic programming: From theory to real-world applications. Technical Report 95.2, University of Rochester, National Resource Laboratory for the Study of Brain and Behavior, Morgan Kaufmann, San Mateo, California, Rochseter, New York, USA, July 9, 1995, Tahoe City, California, USA. Held in conjunction with the twelfth International Conference on Machine Learning.

[1758] Justinian P. Rosca. An analysis of hierarchical genetic programming. Technical Report TR566, The University of Rochester, Computer Science Department, Rochester, New York, 1527, USA, March 1995. Online available at ftp://ftp.cs.rochester.edu/pub/u/rosca/gp/95.tr566.ps.gz and http://citeseer.ist.psu.edu/rosca95analysis.html [accessed 2008-02-24].

[1759] Justinian P. Rosca and Dana H. Ballard. Causality in genetic programming. In *Proceedings of the 6th International Conference on Genetic Algorithms (ICGA95)*, pages 256–263, 1995. In proceedings [636]. Online available at http://citeseer.ist.psu.edu/rosca95causality.html [accessed 2007-08-12].

[1760] Richard S. Rosenberg. *Simulation of genetic populations with biochemical properties*. PhD thesis, The University of Michigan, College of Literature, Science, and the Arts, Computer and Communication Sciences Department, Ann Arbor, MI, USA, June

1967. Online available at http://hdl.handle.net/2027.42/7321 [accessed 2008-10-17]. Other Identifiers: UMR3370, ORA Project 08333, ID: bad1552.0001.001.

[1761] Paul L. Rosin and Freddy Fierens. Improving neural network generalisation. In *Proceedings of the International Geoscience and Remote Sensing Symposium, "Quantitative Remote Sensing for Science and Applications", IGARSS'95*, volume 2, pages 1255–1257. IEEE, July 10–14, 1995, Florenz, Italy. ISBN: 0-7803-2567-2. INSPEC Accession Number: 5112834. doi:10.1109/IGARSS.1995.521718. Online available at http://citeseer.ist.psu.edu/165458.html and http://users.cs.cf.ac.uk/Paul.Rosin/resources/papers/overfitting.pdf [accessed 2007-09-13].

[1762] Claudio Rossi, Elena Marchiori, and Joost N. Kok. An adaptive evolutionary algorithm for the satisfiability problem. In *SAC'00: Proceedings of the 2000 ACM symposium on Applied computing*, volume 1, pages 463–469, 2000, Como, Italy. ACM Press, New York, NY, USA. ISBN: 1-5811-3240-9. doi:10.1145/335603.335912. Online available at http://doi.acm.org/10.1145/335603.335912 and http://citeseer.ist.psu.edu/335101.html [accessed 2007-08-24].

[1763] Gerald P. Roston. *A Genetic Methodology for Configuration Design*. PhD thesis, Department of Mechanical Engineering of Carnegie Mellon University, Pittsburgh, PA 15213-3891, USA, December 1994. Advisors: Rober Sturges, Jr. and William "Red" Wittaker. Online available at http://citeseer.ist.psu.edu/213003.html and http://www.ri.cmu.edu/pubs/pub_3335.html [accessed 2007-08-15].

[1764] Franz Rothlauf, editor. *Late Breaking Papers at Genetic and Evolutionary Computation Conference, GECCO 2005*, June 25–29, 2005, Loews L'Enfant Plaza Hotel, 480 L'enfant Plaza Sw, Washington, D.C. 20024, USA. See also [202, 199, 1766]. Also distributed on CD-ROM at GECCO-2005.

[1765] Franz Rothlauf. *Representations for Genetic and Evolutionary Algorithms*. Physica-Verlag, second edition, August 2002 (1st ed.), 2006 (2nd ed.). ISBN: 978-3-79081-496-5, 3-5402-5059-X, 978-3-54025-059-3. Foreword by David E. Goldberg. Partly online available at http://books.google.de/books?id=fQrSUwop4JkC [accessed 2008-02-27].

[1766] Franz Rothlauf, Misty Blowers, Jürgen Branke, Stefano Cagnoni, Ivan I. Garibay, Ozlem Garibay, Jörn Grahl, Gregory Hornby, Edwin D. de Jong, Tim Kovacs, Sanjeev Kumar, Claudio F. Lima, Xavier Llorà, Fernando Lobo, Laurence D. Merkle, Julian Francis Miller, Jason H. Moore, Michael ONeill, Martin Pelikan, Terry P. Riopka, Marylyn D. Ritchie, Kumara Sastry, Stephen L. Smith, Hal Stringer, Keiki Takadama, Marc Toussaint, Stephen C. Upton, Alden H. Wright, and Shengxiang Yang, editors. *Proceedings of the 2005 workshops on Genetic and evolutionary computation*, June 25–26, 2005, Loews L'Enfant Plaza Hotel, 480 L'enfant Plaza Sw, Washington, D.C. 20024, USA. ACM, New York, NY, USA. See also [202, 199, 1764].

[1767] Franz Rothlauf, Jürgen Branke, Stefano Cagnoni, David W. Corne, Rolf Drechsler, Yaochu Jin, Penousal Machado, Elena Marchiori, Juan Romero, George D. Smith, and Giovanni Squillero, editors. *Applications of Evolutionary Computing, Proceedings of EvoWorkshops 2005: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoMUSART, and EvoSTOC*, volume 3449/2005 of *Lecture Notes in Computer Science (LNCS)*, March 30–April 1, 2005, Lausanne, Switzerland. Springer, Berlin / Heidelberg. ISBN: 3-5402-5396-3, 978-3-54025-396-9.

[1768] Franz Rothlauf, Jürgen Branke, Stefano Cagnoni, Ernesto Costa, Carlos Cotta, Rolf Drechsler, Evelyne Lutton, Penousal Machado, Jason H. Moore, Juan Romero, George D. Smith, Giovanni Squillero, and Hideyuki Takagi, editors. *Applications of Evolutionary Computing, Proceedings of EvoWorkshops 2006: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoINTERACTION, EvoMUSART, and EvoSTOC*, volume 3907/2006 of *Lecture Notes in Computer Science (LNCS)*, April 10–12, 2006, Budapest, Hungary. Springer, Berlin / Heidelberg. ISBN: 3-5403-3237-5.

[1769] Shad Roundy, Dan Steingart, Luc Frechette, Paul Wright, and Jan Rabaey. Power sources for wireless sensor networks. In *EWSN 2004: European workshop on wireless sensor networks No1*, volume 2920/2004 of *Lecture Notes in Computer Science (LNCS)*, pages 1–17. Springer Berlin / Heidelberg, 2004, Berlin, Germany. ISBN: 978-3-54020-825-9. Online available at http://www.eureka.gme.usherb.ca/memslab/docs/PowerReview-2.pdf [accessed 2007-08-01].

[1770] Dmitri Roussinov and Hsinchun Chen. Information navigation on the web by clustering and summarizing query results. *Information Processing & Management*, 37(6):789–816, 2001. Online available at http://dx.doi.org/10.1016/S0306-4573(00)00062-5 [accessed 2007-08-11].

[1771] R. D. Routledge. Diversity indices: Which ones are admissible? *Journal of Theoretical Biology*, 76(4):503–515, February 21, 1979. doi:10.1016/0022-5193(79)90015-8. Online available at http://dx.doi.org/10.1016/0022-5193(79)90015-8 [accessed 2008-11-10].

[1772] J. P. Royston. Some techniques for assessing multivariate normality based on the Shapiro-Wilk W. *Applied Statistics*, 32(2):121–133, 1983. ISSN: 0035-9254.

[1773] William Michael Rudnick. *Genetic algorithms and fitness variance with an application to the automated design of artificial neural networks*. PhD thesis, Oregon Graduate Institute of Science & Technology, Beaverton, OR, USA, 1992. UMI Order No. GAX92-22642.

[1774] Günter Rudolph. How mutation and selection solve long-path problems in polynomial expected time. *Evolutionary Computation*, 4(2):195–205, Summer 1996. ISSN: 1063-6560, 1530-9304. doi:10.1162/evco.1996.4.2.195. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.42.2612 [accessed 2008-08-13].

[1775] Günter Rudolph. *Convergence Properties of Evolutionary Algorithms*, volume 35 of *Forschungsergebnisse zur Informatik*. Verlag Dr. Kovač, Fachverlag für wissenschaftliche Literatur, Hamburg, Germany, 1997. ISBN: 978-3-86064-554-3, 3-8606-4554-4. PhD Thesis.

[1776] Günter Rudolph. Self-adaptation and global convergence: A counter-example. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC99*, volume 1, pages 646–651, 1999. In proceedings [69]. Online available at http://ls11-www.cs.uni-dortmund.de/people/rudolph/publications/papers/CEC99.pdf and http://citeseer.comp.nus.edu.sg/272402.html [accessed 2008-02-24].

[1777] Günter Rudolph. Self-adaptive mutations may lead to premature convergence. *IEEE Transactions on Evolutionary Computation*, 5(4):410–414, 2001. See also [1778].

[1778] Günter Rudolph. Self-adaptive mutations may lead to premature convergence. Technical Report CI–73/99, Fachbereich Informatik, Universität Dortmund, 44221 Dortmund, Germany, April 30, 2001. Online available at http://citeseer.ist.psu.edu/444363.html [accessed 2007-07-28]. See also [1777].

[1779] Thomas Philip Runarsson, Hans-Georg Beyer, Edmund K. Burke, Juan J. Merelo Guervós, L. Darrell Whitley, and Xin Yao, editors. *Proceedings of 9th International Conference on Parallel Problem Solving from Nature – PPSN IX*, volume 4193/2006 of *Lecture Notes in Computer Science (LNCS)*, September 9–13, 2006, University of Iceland, Reykjavik, Iceland. Springer. ISBN: 3-5403-8990-3, 978-3-54038-990-3. doi:10.1007/11844297. See http://ppsn2006.raunvis.hi.is/ [accessed 2007-09-05].

[1780] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, second edition, December 2002. ISBN: 0-1379-0395-2.

[1781] R. A. Rutenbar. Simulated annealing algorithms: an overview. *IEEE Circuits and Devices Magazine*, 5(1):19–26, January 1989. doi:10.1109/101.17235.

[1782] Conor Ryan. Grammatical evolution tutorial. In *Proceedings of Genetic and Evolutionary Computation Conference, GECCO 2006*, 2006. In proceedings [352]. Online available at http://www.grammaticalevolution.org/tutorial.pdf [accessed 2007-09-09].

[1783] Conor Ryan and Michael O'Neill. Grammatical evolution: A steady state approach. In *Late Breaking Papers at the Genetic Programming 1998 Conference*, 1998. In proceedings [1198]. Online available at http://citeseer.ist.psu.edu/260828.html and http://www.grammatical-evolution.org/papers/gp98/index.html [accessed 2007-09-09].

[1784] Conor Ryan, J. J. Collins, and Michael O'Neill. Grammatical evolution: Evolving programs for an arbitrary language. In *Proceedings of the First European Workshop on Genetic Programming*, pages 83–95, 1998. In proceedings [141]. Online available at http://www.grammatical-evolution.org/papers/eurogp98.ps and http://citeseer.ist.psu.edu/ryan98grammatical.html [accessed 2007-09-09].

[1785] Conor Ryan, Michael O'Neill, and J. J. Collins. Grammatical evolution: Solving trigonometric identities. In *Proceedings of Mendel 1998: 4th International Mendel Conference on Genetic Algorithms, Optimisation Problems, Fuzzy Logic, Neural Networks, Rough Sets*, pages 111–119, 1998. In proceedings [2079]. Online available at http://citeseer.ist.psu.edu/360445.html [accessed 2007-11-12].

[1786] Conor Ryan, Terence Soule, Maarten Keijzer, Edward P. K. Tsang, Riccardo Poli, and Ernesto Costa, editors. *Proceedings of the 6th European Conference on Genetic Programming, EuroGP 2003*, volume 2610/2003 of *Lecture Notes in Computer Science (LNCS)*, April 14-16, 2003, Essex, UK. Springer Berlin/Heidelberg. ISBN: 3-5400-0971-X.

S

[1787] Nikolaos V. Sahinidis and Mohit Tawarmalani. Applications of global optimization to process and molecular design. *Computers and Chemical Engineering*, 24(9-10): 2157–2169, October 1, 2000. Online available at http://citeseer.ist.psu.edu/397733.html [accessed 2007-09-01].

[1788] Sancho Salcedo-sanz and Xin Yao. A hybrid hopfield network-genetic algorithm approach for the terminal assignment problem. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, 34(6):2343–2353, December 2004. ISSN: 1083-4419. INSPEC Accession Number: 8207612. doi:10.1109/TSMCB.2004.836471. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.62.7879 [accessed 2008-09-07].

[1789] Sancho Salcedo-Sanz and Xin Yao. Assignment of cells to switches in a cellular mobile network using a hybrid hopfield network-genetic algorithm approach. *Applied Soft Computing*, 8(1):216–224, January 2008. ISSN: 1568-4946. doi:10.1016/j.asoc.2007.01.002.

[1790] Sancho Salcedo-Sanz, Jose A. Portilla-Figueras, Emilio G. Ortiz-García, Angel M. Pérez-Bellido, Christopher Thraves, Antonio Fernández-Anta, and Xin Yao. Optimal switch location in mobile communication networks using hybrid genetic algorithms. *Applied Soft Computing*, 8(4):1486–1497, September 2008. ISSN: 1568-4946. doi:10.1016/j.asoc.2007.10.021. Soft Computing for Dynamic Data Mining. Online available at http://nical.ustc.edu.cn/papers/asc2007b.pdf and http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.87.8731 [accessed 2008-09-01].

[1791] Abdellah Salhi, Hugh Glaser, and David De Roure. Parallel implementation of a genetic-programming based tool for symbolic regression. Technical Report DSSE Technical Reports: DSSE-TR-97-3, Declarative Systems & Software Engineering Group, Department of Electronics and Computer Science, University of Southampton, Highfield, Southampton SO17 1BJ, United Kingdom, July 18, 1997. See also [1792]. Online available at http://www.dsse.ecs.soton.ac.uk/techreports/95-03/97-3.html [accessed 2007-09-09].

[1792] Abdellah Salhi, Hugh Glaser, and David De Roure. Parallel implementation of a genetic-programming based tool for symbolic regression. *Information Processing Let-*

ters, 66(6):299–307, June 30, 1998. ISSN: 0020-0190. CODEN: IFPLAT. Online available at http://dx.doi.org/10.1016/S0020-0190(98)00056-8 [accessed 2007-09-09]. See also [1791].

[1793] David Salomon. *Assemblers and loaders*. Ellis Horwood Series in Computers and Their Applications. Ellis Horwood, Upper Saddle River, NJ, USA, February 1993. ISBN: 978-0-13052-564-2, 0-1305-2564-2.

[1794] *Proceedings of the Eighth Joint Conference on Information Science (JCIS 2005), Section: The Sixth International Workshop on Frontiers in Evolutionary Algorithms (FEA 2005)*, July 21–16, 2005, Salt Lake City Mariott City Center, Salt Lake City, Utah, USA. Workshop held in conjunction with Eighth Joint Conference on Information Sciences. See http://fs.mis.kuas.edu.tw/~cobol/JCIS2005/jcis05/Track4.html [accessed 2007-09-16].

[1795] Arthur L. Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3(3):210–229, July 1959. See also [1796, 1797]. Online available at http://www.research.ibm.com/journal/rd/033/ibmrd0303B.pdf [accessed 2008-10-14].

[1796] Arthur L. Samuel. Some studies in machine learning using the game of checkers. In Edward A. Feigenbaum and Julian Feldman, editors, *Computers and Thought*, pages 71–105. MIT Press, Cambridge, MA, USA, 1963 (1st ed), 1995 (reprint). ISBN: 0-2625-6092-5, 978-0-26256-092-4. See also [1795, 1797].

[1797] Arthur L. Samuel. Some studies in machine learning using the game of checkers ii – recent progress. *IBM Journal of Research and Development*, 11(6):601–617, November 1967. See also [1795]. Online available at http://www.research.ibm.com/journal/rd/116/ibmrd1106C.pdf and http://pages.cs.wisc.edu/~jerryzhu/cs540/handouts/samuel-checkers.pdf [accessed 2008-10-14].

[1798] Harilaos G. Sandalidis, Constandinos X. Mavromoustakis, and Peter P. Stavroulakis. Performance measures of an ant based decentralised routing scheme for circuit switching communication networks. *Soft Computing – A Fusion of Foundations, Methodologies and Applications*, 5(4):313–317, August 2001. ISSN: 1432-7643 (Print) 1433-7479 (Online). doi:10.1007/s005000100104. Online available at http://www.springerlink.com/content/g1u7lygrugwk3nl5/fulltext.pdf [accessed 2008-09-03].

[1799] Yasuhito Sano and Hajime Kita. Optimization of noisy fitness functions by means of genetic algorithms using history of search. In *PPSN VI: Proceedings of the 6th International Conference on Parallel Problem Solving from Nature*, pages 571–580, 2000. doi:10.1007/3-540-45356-3_56. In proceedings [1830].

[1800] Yasuhito Sano and Hajime Kita. Optimization of noisy fitness functions by means of genetic algorithms using history of search with test of estimation. In *Congress on Evolutionary Computation*, pages 360–365, 2002. In proceedings [703].

[1801] Bruno Sareni and Laurent Krähenbühl. Fitness sharing and niching methods revisited. *IEEE Transactions on Evolutionary Computation*, 2(3):97–106, September 1998. ISSN: 1089-778X. CODEN: ITEVF5. INSPEC Accession Number: 6114042. doi:10.1109/4235.735432.

[1802] Manish Sarkar. Evolutionary programming-based fuzzy clustering. In *Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming*, pages 247–256, 1996. In proceedings [709].

[1803] Ruhul Sarker, Robert G. Reynolds, Hussein Abbass, Kay Chen Tan, Bob McKay, Daryl Leslie Essam, and Tom Gedeon, editors. *Proceedings of the IEEE Congress on Evolutionary Computation, CEC2003*, December 8–12, 2003, Canberra, Australia. IEEE Press, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA. ISBN: 0-7803-7804-0. CEC 2003 – A joint meeting of the IEEE, the IEAust, the EPS, and the IEE.

[1804] Ruhul A. Sarker, Hussein Abbas, and Samin Karim. An evolutionary algorithm for constrained multiobjective optimization problems. In *Proceedings*

of the 5th Australia-Japan Joint Workshop on Intelligent & Evolutionary Systems (AJWIS'2001), pages 113–122, November 2001, The University of Otago, Dunedin, New Zealand. Online available at http://www.lania.mx/~ccoello/EMOO/sarker01a.pdf.gz and http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.8.995 [accessed 2008-11-14].

[1805] Warren Sarle. What is overfitting and how can i avoid it? *Usenet FAQs: comp.ai.neural-nets FAQ*, 3: Generalization(3), May 13, 2007. Online available at http://www.faqs.org/faqs/ai-faq/neural-nets/part3/section-3.html [accessed 2008-02-29].

[1806] Warren S. Sarle. Stopped training and other remedies for overfitting. In *Proceedings of the 27th Symposium on the Interface: Computing Science and Statistics*, pages 352–360, June 23, 1995, Pittsburgh, Pennsylvania. Online available at http://citeseer.ist.psu.edu/71530.html and ftp://ftp.sas.com/pub/neural/inter95.ps.Z [accessed 2007-09-13].

[1807] Stefan Saroiu, P. Krishna Gummadi, and Steven Gribble. A measurement study of peer-to-peer file sharing systems. In *SPIE Multimedia Computing and Networking (MMCN2002)*, January 2002, San Jose, CA, USA. Online available at http://www.cs.washington.edu/homes/gribble/papers/mmcn.pdf and http://citeseer.ist.psu.edu/saroiu02measurement.html [accessed 2007-08-13].

[1808] Takahiro Sasaki and Mario Tokoro. Comparison between lamarckian and darwinian evolution on a model using neural networks and genetic algorithms. *Knowledge and Information Systems*, 2(2):201–222, June 2000. ISSN: 0219-1377 (Print) 0219-3116 (Online). doi:10.1007/s101150050011. Online available at http://www.springerlink.com/content/08frl0vbnxj05tak/fulltext.pdf [accessed 2008-11-10].

[1809] Kumara Sastry and David E. Goldberg. Modeling tournament selection with replacement using apparent added noise. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, page 781, 2001. In proceedings [1937]. Also: IlliGAL report 2001014, January 2001, Illinois Genetic Algorithms Laboratory (IlliGAL), Department of General Engineering, University of Illinois at Urbana-Champaign, Online available at http://citeseer.ist.psu.edu/439887.html and http://citeseer.ist.psu.edu/501944.html [accessed 2007-08-25].

[1810] Kumara Sastry and David E. Goldberg. Let's get ready to rumble redux: crossover versus mutation head to head on exponentially scaled problems. In *GECCO'07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1380–1387, 2007. doi:10.1145/1276958.1277215. In proceedings [2037]. Online available at http://doi.acm.org/10.1145/1276958.1277215 [accessed 2008-07-21]. See also [1811].

[1811] Kumara Sastry and David E. Goldberg. Let's get ready to rumble redux: Crossover versus mutation head to head on exponentially scaled problems. IlliGAL Report 2007005, Illinois Genetic Algorithms Laboratory (IlliGAL), University of Illinois, Urbana-Champaign, Urbana IL, USA, February 11, 2007. Online available at http://www.illigal.uiuc.edu/pub/papers/IlliGALs/2007005.pdf [accessed 2008-07-21]. See also [1810].

[1812] Suresh Chandra Satapathy, Jvr Murthy, P. V. G. D. Prasada Reddy, Venkatesh Katari, Satish Malireddi, and V. N. K. Srujan Kollisetty. An efficient hybrid algorithm for data clustering using improved genetic algorithm and nelder mead simplex search. In *ICCIMA'07: Proceedings of the International Conference on Computational Intelligence and Multimedia Applications (ICCIMA 2007)*, pages 498–510, December 13–15, 2007, Sivakasi, Tamil Nadu, India. IEEE Computer Society, Washington, DC, USA. ISBN: 0-7695-3050-8. INSPEC Accession Number: 9895376. doi:10.1109/ICCIMA.2007.68.

[1813] Shinji Sato. Simulated quenching: a new placement method for module generation. In *ICCAD'97: Proceedings of the 1997 IEEE/ACM international*

conference on Computer-aided design, pages 538–541, 1997, San Jose, California, United States. IEEE Computer Society, Washington, DC, USA. ISBN: 0-8186-8200-0. doi:10.1109/ICCAD.1997.643591. Online available at http://www.sigda.org/Archives/ProceedingArchives/Iccad/Iccad97/papers/1997/iccad97/pdffiles/09c_2.pdf and http://portal.acm.org/ft_gateway.cfm?id=266547&type=pdf&coll=GUIDE&dl=GUIDE&CFID=61186331&CFTOKEN=74233409 [accessed 2008-03-27].

[1814] Toshiyuki Satoh, Tadashi Ishihara, and Hikaru Inooka. Systematic design via the method of inequalities. *Control Systems Magazine, IEEE*, 16(5):57–65, October 1996. ISSN: 0272-1708. CODEN: ISMAD7. doi:10.1109/37.537209.

[1815] F. E. Satterthwaite. Random balance experimentation. *Technometrics*, 1(2):111–137, May 1959.

[1816] F. E. Satterthwaite. Revop or random evolutionary operation. Merrimack College Technical Report 10-10-59, Merriam College, North Andover, Massachusetts, USA, 1959.

[1817] Bruce J. Schachter, Larry S. Davis, and Azriel Rosenfeld. Some experiments in image segmentation by clustering of local feature values. *Pattern Recognition*, 11(1):19–28, 1979.

[1818] Daniel L. Schacter. Critical review – implicit memory: history and current status. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 13(3):501–518, July 1987. ISSN: 0278-7393. Online available at http://pages.pomona.edu/~rt004747/lgcs11read/Schacter87.pdf [accessed 2008-12-01].

[1819] Robert Schaefer and Henryk Telega. *Foundations of Global Genetic Optimization*, volume 74/2007 of *Studies in Computational Intelligence*. Springer Berlin / Heidelberg, 2007. ISBN: 978-3-54073-191-7. doi:10.1007/978-3-540-73192-4. Series editor: Janusz Kacprzyk.

[1820] J. David Schaffer, editor. *Proceedings of the 3rd International Conference on Genetic Algorithms*, June 1989, George Mason University, Fairfax, Virginia, USA. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA. ISBN: 1-5586-0066-3.

[1821] J. David Schaffer. *Multiple Objective Optimization with Vector Evaluated Genetic Algorithms*. PhD thesis, Vanderbilt University, 1984.

[1822] J. David Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms and Their Applications*, pages 93–100, 1985. In proceedings [856].

[1823] J. David Schaffer and Darell Whitley, editors. *Proceedings of COGANN-92. International Workshop on Combinations of GeneticAlgorithms and Neural Networks*, June 6, 1992, Baltimore, Maryland, USA. IEEE Computer Society Press. ISBN: 0-8186-2787-5, 978-0-81862-787-3. INSPEC Accession Number: 4487943. doi:10.1109/COGANN.1992.273951. Cat. No.92TH0435-8.

[1824] J. David Schaffer, Larry J. Eshelman, and Daniel Offutt. Spurious correlations and premature convergence in genetic algorithms. In *Proceedings of the First Workshop on Foundations of Genetic Algorithms (FOGA)*, pages 102–112, 1990. In proceedings [1924].

[1825] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5:197–227, June 1990. ISSN: 0885-6125 (Print) 1573-0565 (Online). doi:10.1007/BF00116037. Online available at http://www.springerlink.com/content/x02406w7q5038735/fulltext.pdf and http://citeseer.ist.psu.edu/schapire90strength.html [accessed 2007-09-15].

[1826] Robert E. Schapire, Yoav Freund, Peter Bartlett, and Wee Sun Lee. Boosting the margin: a new explanation for the effectiveness of voting methods. In *Proceedings 14th International Conference on Machine Learning*, pages 322–330. Morgan Kaufmann, July 1997, Nashville, Tennessee, USA. ISBN: 1-5586-0486-3. Online available

at http://citeseer.ist.psu.edu/schapire97boosting.html [accessed 2007-09-15]. See also [156].

[1827] R. Schilling, W. Haase, J. Periaux, and H. Baier, editors. *Proceedings of the Sixth Conference on Evolutionary and Deterministic Mehtods for Design, Optimization and Control with Applications to Indutsrial and Societal Problems, EUROGEN2005*, September 12–14, 2005, TU München, Munich, Germany. TU München. in association with ECCOMAS and ERCOFTAC. See http://www.flm.mw.tum.de/EUROGEN05/ [accessed 2007-09-16].

[1828] Jürgen Schmidhuber. Evolutionary principles in self-referential learning. (on learning how to learn: The meta-meta-... hook.). Master's thesis, Institut für Informatik, Technische Universität München, Munich, Germany, May 14, 1987. Online available at http://www.idsia.ch/~juergen/diploma.html [accessed 2007-11-01].

[1829] Johannes J. Schneider and Scott Kirkpatrick, editors. *Tabu Search Applied to TSP*, chapter part II (Applications), chapter 1, pages 441–447. Scientific Computation. Springer Berlin Heidelberg, 2006. ISBN: 978-3-54034-559-6, 978-3-54034-560-2. doi:10.1007/978-3-540-34560-2_42.

[1830] Marc Schoenauer, Kalyanmoy Deb, Günter Rudolph, Xin Yao, Evelyne Lutton, Juan J. Merelo Guervós, and Hans-Paul Schwefel, editors. *Proceedings of the 6th International Conference on Parallel Problem Solving from Nature – PPSN VI*, volume 1917/2000 of *Lecture Notes in Computer Science (LNCS)*, September 18–20, 2000, Paris, France. Springer. ISBN: 3-5404-1056-2, 978-3-54041-056-0. doi:10.1007/3-540-45356-3. Partly online available at http://books.google.de/books?id=gI26Cld2BY0C [accessed 2008-09-10].

[1831] Eberhard Schöneburg, Frank Heinzmann, and Sven Feddersen. *Genetische Algorithmen und Evolutionsstrategien*. Addison Wesley Verlag, 1993. ISBN: 3-8931-9493-2, 978-3-89319-493-3.

[1832] Ruud Schoonderwoerd. Collective intelligence for network control. Master's thesis, Delft University of Technology, Faculty of Technical Mathematics and Informatics, May 30, 1996. Online available at http://staff.washington.edu/paymana/swarm/schoon96-thesis.pdf [accessed 2008-06-12]. Graduation committee: Prof. dr. H. Koppelaar, Prof. dr. ir. E.J.H. Kerckhoffs, Drs. dr. L.J.M. Rothkrantz, and J.L. Bruten M.Sc.

[1833] Ruud Schoonderwoerd, Owen E. Holland, Janet L. Bruten, and Leon J. M. Rothkrantz. Ant-based load balancing in telecommunications networks. *Adaptive Behavior*, 5(2):169–207, Fall 1996. Online available at http://citeseer.ist.psu.edu/schoonderwoerd96antbased.html and http://www.ifi.unizh.ch/ailab/teaching/FG06/ [accessed 2007-08-19].

[1834] Ruud Schoonderwoerd, Owen E. Holland, and Janet L. Bruten. Ant-like agents for load balancing in telecommunications networks. In Jörg Mller, editor, *AGENTS'97: Proceedings of the first international conference on Autonomous agents*, pages 209–216, February 5–8 1997, Marina del Rey, California, United States. ACM, New York, NY, USA. ISBN: 0-8979-1877-0. doi:10.1145/267658.267718. Online available at http://staff.washington.edu/paymana/swarm/schoon97-icaa.pdf and http://doi.acm.org/10.1145/267658.267718 [accessed 2008-07-26]. Slides online available at [accessed 2008-07-26]. See also http://sigart.acm.org/proceedings/agents97/ [accessed 2008-07-26].

[1835] Herb Schorr and Alain T. Rappaport, editors. *Proceedings of the The First Conference on Innovative Applications of Artificial Intelligence (IAAI-89)*, March 28–30, 1989, Stanford University, Stanford, California, USA. AAAI. ISBN: 978-0-26269-137-6. Published in 1991. See http://www.aaai.org/Conferences/IAAI/iaai89.php [accessed 2007-09-06].

[1836] Nicol N. Schraudolph and Richard K. Belew. Dynamic parameter encoding for genetic algorithms. *Machine Learning*, 9:9–21, June 1992. ISSN: 0885-6125 (Print) 1573-

0565 (Online). doi:10.1023/A:1022624728869. Online available at http://citeseer.ist.psu.edu/schraudolph92dynamic.html and http://www.springerlink.com/content/u684071468r0x423/fulltext.pdf [accessed 2007-08-29]. UCSD Technical Report CS90-175, LANL Technical Report LAUR 90-2795.

[1837] Crystal Schuil, Matthew Valente, Justin Werfel, and Radhika Nagpal. Collective construction using lego robots. In *Annual National Conference on Artificial Intelligence (AAAI), Robot Exhibition*, 2006. In proceedings [805]. Online available at http://www.eecs.harvard.edu/~rad/ssr/papers/aaai06-schuil.pdf [accessed 2008-06-12].

[1838] Michael A. Schumer. *Optimization by adaptive random search.* PhD thesis, Princeton University, NJ, November 1967. Supervisor Kenneth Steiglitz.

[1839] Michael A. Schumer and Kenneth Steiglitz. Adaptive step size random search. *IEEE Transactions on Automatic Control*, AC-13(3):270–276, June 1968.

[1840] Hans-Paul Schwefel. *Numerical Optimization of Computer Models.* John Wiley and Sons Ltd, New York, NY, USA, June 17, 1981. ISBN: 0-4710-9988-0, 978-0-47109-988-8.

[1841] Hans-Paul Schwefel. *Evolution and Optimum Seeking: The Sixth Generation.* John Wiley & Sons, Inc., Wiley Interscience, New York, NY, USA, 1993. ISBN: 0-4715-7148-2, 978-0-47157-148-3. Har/Dsk edition, January 1995.

[1842] Hans-Paul Schwefel and Reinhard Männer, editors. *Proceedings of the 1st Workshop on Parallel Problem Solving from Nature, PPSN I*, volume 496/1991 of *Lecture Notes in Computer Science (LNCS)*, October 1–3, 1990, FRG, Dortmund, Germany. Springer. ISBN: 3-5405-4148-9, 978-3-54054-148-6. doi:10.1007/BFb0029723. Published 1991. See http://ls11-www.informatik.uni-dortmund.de/PPSN/ppsn1/ppsn1.html [accessed 2007-09-05].

[1843] Katja Schwieger, Heinrich Nuszkowski, and Gerhard Fettweis. Analysis of node energy consumption in sensor networks. In Holger Karl, Andreas Willig, and Adam Wolisz, editors, *Proceedings of Wireless Sensor Networks, First European Workshop (EWSN)*, volume 2920 of *Lecture Notes in Computer Science (LNCS)*, pages 94–105. Springer, January 19–21, 2004, Berlin, Germany. ISBN: 3-5402-0825-9. Online available at http://citeseer.ist.psu.edu/634903.html [accessed 2007-08-01].

[1844] A. Carlisle Scott and Philip Klahr, editors. *Proceedings of the The Fourth Conference on Innovative Applications of Artificial Intelligence (IAAI-92)*, July 12–16, 1992, San Jose, California, USA. AAAI. ISBN: 0-2626-9155-8. See http://www.aaai.org/Conferences/IAAI/iaai912.php [accessed 2007-09-06].

[1845] David W. Scott. *Multivariate Density Estimation: Theory, Practice, and Visualization.* Wiley Series in Probability and Statistics. Wiley-Interscience, John Wiley & Sons, August 1992. ISBN: 0-4715-4770-0.

[1846] Colin C. Seaton and Maryjane Tremayne. Differential evolution: crystal structure determination of a triclinic polymorph of adipamide from powder diffraction data. *Chemical Communications (Camb).*, 880(8):1, April 2002. Online available at http://www.rsc.org/Publishing/Journals/CC/article.asp?doi=b200436d [accessed 2007-08-13].

[1847] *Ninth International Workshop on Learning Classifier Systems (IWLCS 2006)*, July 8–9, 2006, Seattle, WA, USA. Held during GECCO-2006 (see also [352]).

[1848] *Seventh International Workshop on Learning Classifier Systems (IWLCS-2004)*, June 26, 2004, Seattle, Washington, USA. Held during GECCO-2004 (see also [544, 545]). See also [1181].

[1849] A.V. Sebald and Lawrence Jerome Fogel, editors. *Proceedings of the 3rd Annual Conference on Evolutionary Programming*, 1994. World Scientific Publishing, River Edge, NJ.

[1850] Robert Sedgewick. *Algorithms in Java, Parts 1-4.* Addison Wesley, third edition, September 2002. ISBN: 0-2013-6120-5.

[1851] Y. Ahmet Şekercioğlu, Andreas Pitsilides, and Athanasios V. Vasilakos. Computational intelligence in management of atm networks: a survey of current state of research. In *ESIT'99, European Symposium on Intelligent Techniques*, June 3–4, 1999, Orthodox Academy of Crete, Chania-Crete, Greece. Online available at http://www.erudit.de/erudit/events/esit99/12525_P.pdf and http://www.cs.ucy.ac.cy/networksgroup/pubs/published/1999/CI-ATM-London99.pdf [accessed 2008-09-03].

[1852] Walter Selke, L.N. Shchur, and A.L. Talapov. Cluster–flipping monte carlo algorithm and correlations in "good" random number generators. *JETP Letters*, 58(8):684–686, 1993.

[1853] Mrinal K. Sen and Paul L. Stoffa. Nonlinear one-dimensional seismic waveform inversion using simulated annealing. *Geophysics*, 56(10):1624–1638, October 1991.

[1854] Bernhard Sendhoff, Martin Kreutz, and Werner von Seelen. A condition for the genotype-phenotype mapping: Causality. In *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA97)*, pages 73–80, 1997. In proceedings [98]. Online available at http://citeseer.ist.psu.edu/sendhoff97condition.html and http://arxiv.org/abs/adap-org/9711001 [accessed 2007-08-12].

[1855] Randall S. Sexton, Bahram Alidaee, Robert E. Dorsey, and John D. Johnson. Global optimization for artificial neural networks: A tabu search application. *European Journal of Operational Research*, 106(2–3):570–584, April 1998. doi:10.1016/S0377-2217(97)00292-0. Online available at http://dx.doi.org/10.1016/S0377-2217(97)00292-0 [accessed 2007-08-25].

[1856] Mark Shackleton, Rob Shipman, and Marc Ebner. An investigation of redundant genotype-phenotype mappings and their role in evolutionary search. In *Proceedings of the 2000 Congress on Evolutionary Computation CEC00*, pages 493–500, 2000. In proceedings [1002]. Online available at http://citeseer.ist.psu.edu/409243.html and http://wwwi2.informatik.uni-wuerzburg.de/mitarbeiter/ebner/research/publications/uniWu/redundant.ps.gz [accessed 2007-07-29].

[1857] S. H. Shami, I. M. A. Kirkwood, and Mark C. Sinclair. Evolving simple fault-tolerant routing rules using genetic programming. *Electronics Letters*, 33(17):1440–1441, August 14, 1997. ISSN: Print: 0013-5194, Online: 1350-911X. doi:10.1049/el:19970996. See also [1143].

[1858] Claude Elwood Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423/623–656, July/October 1948. Online available at http://plan9.bell-labs.com/cm/ms/what/shannonday/paper.html [accessed 2007-09-15]. Also published in D. Slepian, editor, Key Papers in the Development of Information Theory, New York: IEEE Press, 1974; N. J. A. Sloane and A. D. Wyner, editors, Claude Elwood Shannon: Collected Papers, New York: IEEE Press, 1993; W. Weaver and Claude Elwood Shannon, The Mathematical Theory of Communication, Urbana, Illinois: University of Illinois Press, 1949, republished in paperback 1963.

[1859] Oliver John Sharpe. *Towards a Rational Methodology for Using Evolutionary Search Algorithms*. PhD thesis, University of Sussex, January 2000. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.36.1667 [accessed 2008-08-10].

[1860] Nicholas Peter Sharples. Evolutionary approaches to adaptive protocol design. In Darren Pearce, editor, *CSRP 512: The 12th White House Papers Graduate Research in Cognitive and Computing Sciences at Sussex*, Cognitive Science Research Papers, pages 60–62. University of Sussex at Brighton, November 1999. Online available at ftp://ftp.informatics.sussex.ac.uk/pub/reports/csrp/csrp512.ps.Z [accessed 2008-06-23].

[1861] Nicholas Peter Sharples. *Evolutionary Approaches to Adaptive Protocol Design*. PhD thesis, School of Cognitive & Computing Sciences of the University of Sussex, August 2001. ASIN: B001ABO1DK.

[1862] Nicholas Peter Sharples and Ian Wakeman. Protocol construction using genetic search techniques. In *Real-World Applications of Evolutionary Computing, EvoWorkshops 2000: EvoIASP, EvoSCONDI, EvoTel, EvoSTIM, EvoROB, and EvoFlight*, pages 73–95, 2000. doi:10.1007/3-540-45561-2_23. In proceedings [320].

[1863] J. Shekel. Test functions for multimodal search techniques. In *Proceedings of the Fifth Annual Princeton Conference on Information Science and Systems*, pages 354–359. Princeton University Press, 1971, Princeton, NJ, USA. See also http://en.wikipedia.org/wiki/Shekel_function [accessed 2007-11-06] and http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO_files/Page2354.htm [accessed 2007-11-06].

[1864] Qi Shen, Jian-Hui Jiang, Chen-Xu Jiao, Shuang-Yan Huan, Guo li Shen, , and Ru-Qin Yu. Optimized partition of minimum spanning tree for piecewise modeling by particle swarm algorithm. qsar studies of antagonism of angiotensin ii antagonists. *Journal of Chemical Information and Modeling / J. Chem. Inf. Comput. Sci.*, 44(6): 2027–2031, November 2004. doi:10.1021/ci034292+ S0095-2338(03)04292-6. Online available at http://dx.doi.org/10.1021/ci034292+ [accessed 2007-08-21].

[1865] *Proceedings of the Third Joint Conference on Information Science (JCIS 1997), Section: The First International Workshop on Frontiers in Evolutionary Algorithms (FEA 1998)*, March 1–5, 1997, Sheraton Imperial Hotel & Convention Center, Research Triangle Park, North Carolina, USA. Workshop held in conjunction with Third Joint Conference on Information Sciences.

[1866] David J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. Chapman & Hall/CRC, 2nd edition, February 24, 2000. ISBN: 1-5848-8133-X, 978-1-58488-133-9, 1-5848-8440-1, 978-1-58488-440-8, 978-1-58488-814-7. Partly online available at http://books.google.com/books?id=bmwhcJqq01cC [accessed 2008-08-07] (3rd ed).

[1867] Yuhui Shi and Marco Dorigo, editors. *Proceedings of the 2007 IEEE Swarm Intelligence Symposium. SIS'07*, IEEE International Symposia on Swarm Intelligence, April 1–5, 2007, Hilton Hawaiian Village Beach Resort & Spa, Hawaii, USA. Institute of Electrical & Electronics Enginee (IEEE). http://www.computelligence.org/sis/2007/ [accessed 2007-08-26].

[1868] Jung Eun Shim and Won Suk Lee. A landmark extraction method for protein 2de gel images based on multi-dimensional clustering. *Artificial Intelligence in Medicine*, 35(1-2):157–170, 2005. Online available at http://linkinghub.elsevier.com/retrieve/pii/S093336570500076X and http://dx.doi.org/10.1016/j.artmed.2005.07.002 [accessed 2007-08-11].

[1869] Masaya Shinkai, Hern'an Aguirre, and Kiyoshi Tanaka. Mutation strategy improves gas performance on epistatic problems. In *Proceedings of the 2002 Congress on Evolutionary Computation, CEC'02*, volume 1, pages 968–973, 2002. doi:10.1109/CEC.2002.1007056. In proceedings [703].

[1870] Rob Shipman. Genetic redundancy: Desirable or problematic for evolutionary adaptation? In *Proceedings of the 4^{th} International Conference on Artificial Neural Nets and Genetic Algorithms*, pages 1–11, 1999. In proceedings [576]. Online available at http://citeseer.ist.psu.edu/shipman99genetic.html [accessed 2007-07-29].

[1871] Rob Shipman, Marc Shackleton, Marc Ebner, and R. Watson. Neutral search spaces for artificial evolution: a lesson from life. In Mark Bedau, John S. McCaskill, Norman H. Packard, Steen Rasmussen, John McCaskill, and Norman Packard, editors, *Artificial Life VII: Proceedings of the Seventh International Conference on Artificial Life*, Bradford Books, Complex Adaptive Systems. The MIT Press, July 2000. ISBN: 0-2625-2290-X, 978-0-26252-290-8. Online available at http://citeseer.ist.psu.edu/shipman00neutral.html and http://books.google.com/books?id=-xLGm7KFGy4C [accessed 2007-07-29].

[1872] Rob Shipman, Mark Shackleton, and I. Harvey. The use of neutral genotype-phenotype mappings for improved evolutionary search. *BT Technology Journal*, 18(4): 103–111, 2000. Online available at http://citeseer.ist.psu.edu/shipman00use.html [accessed 2007-07-29].

[1873] Rajeev Shorey, A. Ananda, Mun Choon Chan, and Wei Tsang Ooi. *Mobile, Wireless, and Sensor Networks: Technology, Applications, and Future Directions*. Wiley-IEEE Press, March 23, 2006. ISBN: 978-0-47171-816-1.

[1874] John N. Shutt. Recursive adaptable grammars. Master's thesis, Computer Science Department, Worcester Polytechnic Institute, Worcester Massachusetts, August 10, 1993. Approved by Roy S. Rubinstein and Robert E. Kinicki. Online available at http://libra.msra.cn/paperdetail.aspx?id=256609 and http://en.scientificcommons.org/234810 [accessed 2007-08-17].

[1875] Patrick Siarry and Zbigniew Michalewicz, editors. *Advances in Metaheuristics for Hard Optimization*. Natural Computing Series. Springer-Verlag, LLC, New York, January 2008. ISBN: 978-3-54072-959-4. Partly online available at http://books.google.de/books?id=TkMR1b-VCR4C [accessed 2008-08-01]. Series editors: G. Rozenberg, Thomas Bäck, Ágoston E. Eiben, J.N. Kok, and H.P. Spaink.

[1876] Wojciech Wladyslaw Siedlecki and Jack Sklansky. Constrained genetic optimization via dynamic reward-penalty balancing and its use in pattern recognition. In *Proceedings of the third international conference on Genetic algorithms*, pages 141–150, 1989. In proceedings [1820]. See also [1877].

[1877] Wojciech Wladyslaw Siedlecki and Jack Sklansky. Constrained genetic optimization via dynamic reward-penalty balancing and its use in pattern recognition. In C. H. Chen, L. F. Pau, and P. S. P. Wang, editors, *Handbook of Pattern Recognition and Computer Vision*, chapter 1.3.3, pages 108–124. World Scientific, 1993. ISBN: 9-8102-2276-9, 978-9-81022-276-5. Partly online available at http://books.google.de/books?id=5J6J7PP0-ZMC [accessed 2008-11-15]. See also [1876].

[1878] Sidney Siegel and N. John Castellan Jr. *Nonparametric Statistics for The Behavioral Sciences*. Humanities/Social Sciences/Languages. McGraw-Hill Book Company, New York, NY, USA, 1st: 1956, 2nd: 1988. ISBN: 0-0705-7357-3, 978-0-07057-357-4.

[1879] Bernard. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman & Hall/CRC, Bristol, England, April 1986. ISBN: 0-4122-4620-1.

[1880] Kwang Mon Sim and Weng Hong Sun. Multiple ant-colony optimization for network routing. In *First International Symposium on Cyber Worlds: Theory and Practices (CW'02)*, pages 277–281, November 6–8, 2002. IEEE Computer Society, Los Alamitos, CA, USA. ISBN: 0-7695-1862-1. doi:10.1109/CW.2002.1180890.

[1881] Farhan Simjee and Pai H. Chou. Everlast: long-life, supercapacitor-operated wireless sensor node. In *ISLPED'06: Proceedings of the 2006 international symposium on Low power electronics and design*, pages 197–202, October 4–6, 2006, Tegernsee, Bavaria, Germany. ACM Press, New York, NY, USA. ISBN: 1-5959-3462-6. Online available at http://portal.acm.org/citation.cfm?id=1098918.1098980 [accessed 2007-08-01].

[1882] Dan Simon. *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*. Wiley-Interscience, June 2006. ISBN: 978-0-47170-858-2.

[1883] George Gaylord Simpson. The baldwin effect. *Evolution*, 7(2):110–117, June 1953. ISSN: 0014-3820.

[1884] Karl Sims. Artificial evolution for computer graphics. In *SIGGRAPH'91: Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 319–328. ACM, New York, NY, USA, 1991. ISBN: 0-8979-1436-8. doi:10.1145/122718.122752. Online available at http://doi.acm.org/10.1145/122718.122752 [accessed 2008-03-22].

[1885] Mark C. Sinclair. Minimum cost topology optimisation of the cost 239 european optical network. In *Proceedings of the International Conference on Artificial Neural*

Networks and Genetic Algorithms, pages 26–29, 1995. In proceedings [1627]. Online available at http://citeseer.ist.psu.edu/63056.html [accessed 2008-07-29].

[1886] Mark C. Sinclair. Evolutionary telecommunications: A summary. In *Proceedings of Evolutionary Telecommunications: Past, Present and Future – A Bird-of-a-feather Workshop at GECCO 99*, 1999. In proceedings [1889]. Online available at http://uk.geocities.com/markcsinclair/ps/etppf_sin_sum.ps.gz and http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.38.6797 [accessed 2008-08-01], presentation online available at http://uk.geocities.com/markcsinclair/ps/etppf_sin_slides.ps.gz [accessed 2008-08-01].

[1887] Mark C. Sinclair. Optical mesh network topology design using node-pair encoding genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1192–1197, 1999. In proceedings [142]. Online available at http://citeseer.ist.psu.edu/177545.html and http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/sinclair_1999_OMNTDNEGP.html [accessed 2008-07-29].

[1888] Mark C. Sinclair. Node-pair encoding genetic programming for optical mesh network topology design. In *Telecommunications Optimization: Heuristic and Adaptive Techniques*, chapter 6, pages 99–114. John Wiley & Sons, Ltd., 2000. In collection [450].

[1889] Mark C. Sinclair, David Corne, and George D. Smith, editors. *Proceedings of Evolutionary Telecommunications: Past, Present and Future – A Bird-of-a-feather Workshop at GECCO 99*, July 13, 1999, Orlando, Florida, USA. See also [142, 1584]. Online available at http://uk.geocities.com/markcsinclair/etppf.html [accessed 2008-06-25].

[1890] Mark C. Sinclair, David Corne, and George D. Smith. Evolutionary telecommunications: Past, present and future. In *Proceedings of Evolutionary Telecommunications: Past, Present and Future – A Bird-of-a-feather Workshop at GECCO 99*, 1999. In proceedings [1889]. Online available at http://uk.geocities.com/markcsinclair/ps/etppf_sin_ws.ps.gz and http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.53.6326&rep=rep1&type=pdf [accessed 2008-08-01].

[1891] Michael Singer. Intel first to ship dual core x86 chip. *internetnews.com*, April 2005. Online available at http://www.internetnews.com/ent-news/article.php/3496926 [accessed 2007-07-16]. See also http://www.internetnews.com/ [accessed 2007-09-11].

[1892] Gulshan Singh and Kalyanmoy Deb. Comparison of multi-modal optimization algorithms based on evolutionary algorithms. In *GECCO'06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1305–1312, 2006. doi:10.1145/1143997.1144200. In proceedings [352]. Session: Genetic algorithms: papers. Online available at http://doi.acm.org/10.1145/1143997.1144200 [accessed 2007-09-10].

[1893] Abhishek Sinha and David E. Goldberg. A survey of hybrid genetic and evolutionary algorithms. IlliGAL Report 2003004, Illinois Genetic Algorithms Laboratory (IlliGAL), Department of General Engineering, University of Illinois at Urbana-Champaign, 117 Transportation Building, 104 S. Mathews Avenue, Urbana, IL 61801, January 2003. Online available at http://citeseer.ist.psu.edu/659509.html [accessed 2008-06-05].

[1894] Michael Sipser. *Introduction to the Theory of Computation*. PWS Publishing, second edition, 2006. ISBN: 0-5349-4728-X.

[1895] Zbigniew Skolicki. An analysis of island models in evolutionary computation. In *Workshop Proceedings of Genetic and Evolutionary Computation Conference, GECCO 2005*, pages 386–389, 2005. In proceedings [199]. Online available at http://cs.gmu.edu/~eclab/papers/skolicki05analysis.pdf and http://portal.acm.org/citation.cfm?id=1102256.1102343 [accessed 2007-08-14].

[1896] Zbigniew Skolicki and Kenneth Alan De Jong. The influence of migration sizes and intervals on island models. In *GECCO'05: Proceedings of the 2005*

conference on Genetic and evolutionary computation, pages 1295–1302, 2005. doi:http://doi.acm.org/10.1145/1068009.1068219. In proceedings [202]. Online available at http://portal.acm.org/citation.cfm?id=1068009.1068219 [accessed 2007-08-14].

[1897] Hendrik Skubch. Hierarchical strategy learning for flux agents. Master's thesis, Technische Universität Dresden, Dresden, Germany, February 18, 2007. Supervisor: Prof. Michael Thielscher. Online available at http://www.phenomene.de/hs/hslfa-dipl-hs.pdf [accessed 2007-11-27]. See also [1898].

[1898] Hendrik Skubch. *Hierarchical Strategy Learning for FLUX Agents*. An Applied Technique. VDM Verlag Dr. Müller, December 12, 2007. ISBN: 978-3-83645-271-7. See also [1897].

[1899] N. J. H. Small. Miscellanea: Marginal skewness and kurtosis in testing multivariate normality. *Applied Statistics*, 29(1):85–87, 1980. ISSN: 0035-9254.

[1900] Rafal Smigrodzki, Ben Goertzel, Cassio Pennachin, Lucio Coelho, Francisco Prosdocimi, and W. Davis Parker Jr. Genetic algorithm for analysis of mutations in parkinson's disease. *Artificial Intelligence in Medicine*, 35:227–241, November 2005. Online available at http://dx.doi.org/10.1016/j.artmed.2004.11.006 [accessed 2007-08-05].

[1901] Alice E. Smith and David W. Coit. Penalty functions. In *Handbook of Evolutionary Computation*, chapter C 5.2. Oxford University Press in cooperation with the Institute of Physics Publishing, 1997. In collection [104]. Online available at http://www.cs.cinvestav.mx/~constraint/papers/chapter.pdf [accessed 2008-11-15].

[1902] George D. Smith, Nigel C. Steele, and Rudolf F. Albrecht, editors. *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms, ICANNGA*, April 1–4, 1997, University of East Anglia, Norwich, England, U.K. Springer, Vienna, Austria. ISBN: 978-3-21183-087-1, 3-2118-3087-1. Published in 1998.

[1903] John Maynard Smith. Natural selection and the concept of a protein space. *Nature*, 225(5232):563–564, February 7, 1970. ISSN: 0028-0836. doi:10.1038/225563a0. Received November 7, 1969.

[1904] John Miles Smith and Diane C. P. Smith. Database abstractions: Aggregation and generalization. *ACM Transaction on Database Systems*, 2(2):105–133, June 1977. ISSN: 0362-5915. Online available at http://portal.acm.org/citation.cfm?doid=320544.320546 and http://libra.msra.cn/paperDetail.aspx?id=794625& [accessed 2007-08-11].

[1905] Murray Smith. *Neural Networks for Statistical Modeling*. John Wiley & Sons, Inc., New York, NY, USA / International Thomson Computer Press, Boston, USA, April 1993/1996. ISBN: 0-4420-1310-8, 1-8503-2842-0, 978-1-85032-842-1.

[1906] Peter W. H. Smith and Kim Harries. Code growth, explicitly defined introns, and alternative selection schemes. *Evolutionary Computation*, 6(4):339–360, Winter 1998. ISSN: 1063-6560, 1530-9304. doi:10.1162/evco.1998.6.4.339. Online available at http://citeseer.ist.psu.edu/harries98code.html [accessed 2008-07-22].

[1907] Reid G. Smith and A. Carlisle Scott, editors. *Proceedings of the The Third Conference on Innovative Applications of Artificial Intelligence (IAAI-91)*, June 14–19, 1991, Anaheim, California, USA. AAAI. ISBN: 0-2626-9148-5. See http://www.aaai.org/Conferences/IAAI/iaai91.php [accessed 2007-09-06].

[1908] Robert Elliott Smith. *Default hierarchy formation and memory exploitation in learning classifier systems*. PhD thesis, University of Alabama, Tuscaloosa, AL, USA, 1991. UMI Order No. GAX91-30265.

[1909] Robert Elliott Smith. A report on the first international workshop on learning classifier systems (IWLCS-92). Online available at http://en.scientificcommons.org/70956 and http://libra.msra.cn/paperDetail.aspx?id=352868 [accessed 2007-08-18]. See also [1501], 1992.

[1910] Shana Shiang-Fong Smith. Using multiple genetic operators to reduce premature convergence in genetic assembly planning. *Computers in Industry*, 54(1):35–49, May 2004. ISSN: 0166-3615. doi:10.1016/j.compind.2003.08.001.

[1911] Stephen Smith and Stefano Cagnoni, editors. *Medical Applications of Genetic and Evolutionary Computation (MedGEC) Workshop 2008*, July 12, 2008, Renaissance Atlanta Hotel Downtown, 590 West Peachtree Street NW, Atlanta, Georgia 30308 USA. ACM Press, New York, NY, USA. ISBN: 978-1-60558-131-6. Part of GECCO 2008, see also [1117] and http://www.elec.york.ac.uk/intsys/events/MedGEC2008/home.htm [accessed 2008-07-21].

[1912] Stephen Frederick Smith. *A Learning System based on Genetic Adaptive Algorithms*. PhD thesis, University of Pittsburgh, Pittsburgh, PA, USA, 1980. Order No. AAI8112638, University Microfilms No. 81-12638.

[1913] Tom Smith, Phil Husbands, Paul Layzell, and Michael O'Shea. Fitness landscapes and evolvability. *Evolutionary Computation*, 10(1):1–34, Spring 2002. ISSN: 1063-6560. doi:10.1162/106365602317301754. Online available at http://wotan.liu.edu/docis/dbl/evocom/2002_10_1_1_FLAE.html [accessed 2007-07-28].

[1914] Donald L. Snyder and Michael I. Miller. *Random Point Processes in Time and Space*. Springer, second edition, June 19, 1991. ISBN: 978-0-38797-577-1.

[1915] Elliott Sober. The two faces of fitness. In Rama Shankar Singh, Costas B. Krimbas, Diane B. Paul, and John Beatty, editors, *Thinking about Evolution: Historical, Philosophical, and Political Perspectives*, chapter 15, pages 309–321. Cambridge University Press, 2001. ISBN: 0-5216-2070-8, 978-0-52162-070-3. Online available at http://philosophy.wisc.edu/sober/tff.pdf [accessed 2008-08-11], Book partly online available at http://books.google.de/books?id=HmVbYJ93d-AC [accessed 2008-08-11].

[1916] Richard Soland, Ambrose Coicoechea, L. Duckstein, and Stanley Zionts, editors. *Proceedings of the 9th International Conference on Multiple Criteria Decision Making: Theory and Applications in Business, Industry, and Government (MCDM'1990)*, 1990, Fairfax, USA. Springer. ISBN: 978-0-38797-805-5. Published in July 1992.

[1917] Francisco J. Solis and Roger J-B. Wets. Minimization by random search techniques. *Mathematics of operations research*, 6(1):19–30, February 1981. ISSN: 0364765X.

[1918] Dawn Xiaodong Song, Adrian Perrig, and Doantam Phan. Agvi – automatic generation, verification, and implementation of security protocols. In *Computer Aided Verification – CAV'01: Proceedings of the 13th International Conference on Computer Aided Verification*, volume 2102/2001 of *Lecture Notes in Computer Science (LNCS)*, pages 241–245. Springer Berlin / Heidelberg, July 18–22, 2001, Paris, France. ISBN: 3-5404-2345-1, 978-3-54042-345-4. doi:10.1007/3-540-44585-4_21.

[1919] Dong Song. A linear genetic programming approach to intrusion detection. Master's thesis, Dalhousie University, Halifax, Nova Scotia, Canada, March 2003. Advisors: Malcolm Heywood and Nur Zincir-Heywood. Online available at http://users.cs.dal.ca/~mheywood/Thesis/DSong.pdf [accessed 2008-06-16].

[1920] Dong Song, Malcom I. Heywood, and A. Nur Zincir-Heywood. A linear genetic programming approach to intrusion detection. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2003. In proceedings [335]. Online available at http://users.cs.dal.ca/~mheywood/X-files/Publications/27242325.pdf [accessed 2008-06-16].

[1921] Branko Souček and The IRIS Group, editors. *Dynamic, Genetic, and Chaotic Programming: The Sixth-Generation*. Sixth Generation Computer Technologies. John Wiley Interscience, New York, April 1992. ISBN: 0-4715-5717-X, 978-0-47155-717-3. Partly online available at http://www.amazon.com/gp/reader/047155717X/ref=sib_dp_pt/002-6076954-4198445#reader-link [accessed 2008-05-29].

[1922] Terence Soule and James A. Foster. Removal bias: a new cause of code growth in tree based evolutionary programming. In *1998 IEEE International Conference*

on Evolutionary Computation, pages 781–186, 1998. In proceedings [1001]. Online available at http://citeseer.ist.psu.edu/313655.html [accessed 2007-09-07].

[1923] James C. Spall. *Introduction to Stochastic Search and Optimization*. Estimation, Simulation, and Control – Wiley-Interscience Series in Discrete Mathematics and Optimization. John Wiley & Sons, first edition, June 2003. ISBN: 978-0-47133-052-3.

[1924] Bruce M. Spatz and Gregory J. E. Rawlins, editors. *Proceedings of the First Workshop on Foundations of Genetic Algorithms (FOGA)*, July 15–18, 1990, Indiana University, Bloomington Campus, Indiana, USA. Morgan Kaufmann Publishers, Inc., 2929 Campus Drive, Suite 260, San Mateo, CA 94403, USA. ISBN: 1-5586-0170-8. Published July 1, 1991. Partly online available at http://books.google.de/books?id=Df12yLrlUZYC [accessed 2008-05-29].

[1925] William M. Spears. *Evolutionary Algorithms – The Role of Mutation and Recombination*. Natural Computing Series. Springer, Berlin, 2000. ISBN: 978-3-54066-950-0.

[1926] William M. Spears and Kenneth Alan De Jong. Using genetic algorithms for supervised concept learning. In *Proceedings of the 2nd International IEEE Conference on Tools for Artificial Intelligence*, pages 335–341, 6-9 1990, Herndon, VA, USA. IEEE Computer Society Press, Los Alamitos, CA, USA. IEEE Cat. No. 90CH2915-7? Online available at http://citeseer.ist.psu.edu/11061.html and http://www.cs.uwyo.edu/~wspears/papers/ieee90.pdf [accessed 2007-09-12].

[1927] William M. Spears and Worthy N. Martin, editors. *Proceedings of the Sixth Workshop on Foundations of Genetic Algorithms (FOGA)*, July 21–23, 2000, Charlottesville, VA, USA. Morgan Kaufmann, San Mateo, CA, USA. ISBN: 1-5586-0734-X. see http://www.cs.uwyo.edu/~wspears/foga00/index.html [accessed 2007-09-01].

[1928] Lee Spector. Evolving control structures with automatically defined macros. In Eric V. Siegel and John R. Koza, editors, *Working Notes for the AAAI Symposium on Genetic Programming*, pages 99–105, November 10–12, 1995, MIT, Cambridge, MA, USA. AAAI Press, 445 Burgess Drive, Menlo Park, CA 94025, USA. Technical Report FS-95-01. Online available at http://citeseer.ist.psu.edu/spector95evolving.html and http://helios.hampshire.edu/lspector/pubs/ADM-fallsymp-e.pdf [accessed 2008-06-12].

[1929] Lee Spector. Simultaneous evolution of programs and their control structures. In *Advances in Genetic Programming 2*, chapter 7, pages 137–154. MIT Press, 1996. In collection [61]. Online available at http://citeseer.comp.nus.edu.sg/33885.html and http://helios.hampshire.edu/lspector/pubs/AiGP2-post-final-e.pdf [accessed 2008-06-20].

[1930] Lee Spector. Autoconstructive evolution: Push, pushgp, and pushpop. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2001*, pages 137–146, 2001. In proceedings [1937]. Online available at http://citeseer.ist.psu.edu/445431.html and http://hampshire.edu/lspector/pubs/ace.pdf [accessed 2007-12-24].

[1931] Lee Spector. Adaptive populations of endogenously diversifying pushpop organisms are reliably diverse. In Standish, Abbass, and Bedau, editors, *ICAL 2003: Proceedings of the eighth international conference on Artificial life*, pages 142–145. MIT Press, Cambridge, MA, USA, December 9–12, 2002, University of New South Wales, Sydney, NSW, Australia. ISBN: 0-2626-9281-3. Online available at http://www.alife.org/alife8/proceedings/sub962.pdf and http://citeseer.ist.psu.edu/627720.html [accessed 2007-12-25].

[1932] Lee Spector. *Automatic Quantum Computer Programming – A Genetic Programming Approach*. Genetic Programming. Kluwer Academic Publishers / Springer Science+Business Media, New York, paperback edition 2007 edition, June 2004. ISBN: 0-3873-6496-X, 0-3873-6791-8, 978-0-38736-496-4, 978-0-38736-791-0. Library of Congress Control Number: 2006931640. Series editor: John Koza. Partly online available at http://books.google.de/books?id=mEKfvtxmn9MC [accessed 2008-05-04].

[1933] Lee Spector and Alan Robinson. Multi-type, self-adaptive genetic programming as an agent creation tool. In *GECCO 2002: Proceedings of the Bird of a Feather Workshops, Genetic and Evolutionary Computation Conference*, pages 73–80, 2002. In proceedings [154]. Online available at http://citeseer.ist.psu.edu/614297.html and http://hampshire.edu/lspector/pubs/ecomas2002-spector-toappear.pdf [accessed 2007-12-25].

[1934] Lee Spector and Alan Robinson. Genetic programming and autoconstructive evolution with the push programming language. *Genetic Programming and Evolvable Machines*, 3(1):7–40, 2002. Received August 15, 2001; Revised November 26, 2001. Online available at http://citeseer.ist.psu.edu/619906.html and http://hampshire.edu/lspector/pubs/push-gpem-final.pdf [accessed 2007-12-25].

[1935] Lee Spector and Kilian Stoffel. Ontogenetic programming. In *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 394–399, 1996. In proceedings [1207]. Online available at http://hampshire.edu/lspector/pubs/onto-gp96-e.ps [accessed http://citeseer.comp.nus.edu.sg/2137.html]2008-06-20.

[1936] Lee Spector, William B. Langdon, Una-May O'Reilly, and Peter John Angeline, editors. *Advances in Genetic Programming*, volume 3 of *Complex Adaptive Systems*. MIT Press, Cambridge, MA, USA, July 16, 1999. ISBN: 0-2621-9423-6, 978-0-26219-423-5. Partly online available at http://books.google.de/books?id=5Qwbal3AY6oC [accessed 2008-09-16].

[1937] Lee Spector, Erik D. Goodman, Annie Wu, William B. Langdon, Hans-Michael Voigt, Mitsuo Gen, Sandip Sen, Marco Dorigo, Shahram Pezeshk, Max H. Garzon, and Edmund Burke, editors. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'01)*, July 7–11, 2001, Holiday Inn Golden Gateway Hotel, San Francisco, California, USA. Morgan Kaufmann. ISBN: 978-1-55860-774-3, 1-5586-0774-9. See also [833].

[1938] Lee Spector, Chris Perry, Jon Klein, and Maarten Keijzer. Push 3.0 programming language description. Technical Report HC-CSTR-2004-02, School of Cognitive Science, Hampshire College, Amherst , Massachusetts 01002, USA, September 9, 2004. Successor to the Push 2.0 Programming Language Description (http://hampshire.edu/lspector/push2-description.html [accessed 2007-12-25]), from which it borrows large chunks of text. Online available at http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/Spector_push3tr.html and http://www.hampshire.edu/cms_PDF/HC-CSTR-2004-02.pdf [accessed 2007-12-25].

[1939] Lee Spector, Jon Klein, and Maarten Keijzer. The push3 execution stack and the evolution of control. In *GECCO 2005: Proceedings of the 2005 conference on Genetic and evolutionary computation*, volume 2, pages 1689–1696, 2005. In proceedings [199]. Online available at http://doi.acm.org/10.1145/1068009.1068292 and http://hampshire.edu/lspector/pubs/push3-gecco2005.pdf [accessed 2007-12-25].

[1940] Herbert Spencer. *The Principles of Biology*, volume 1. London & Edinburgh: Williams and Norgate, first edition, 1864 and 1867. Online available at http://www.archive.org/details/ThePrinciplesOfBiology [accessed 2007-08-05].

[1941] W. Spendley, G. R. Hext, and F. R. Himsworth. Sequential application of simplex designs in optimisation and evolutionary operation. *Technometrics*, 4(4):441–461, November 1962.

[1942] Piet Spiessens. Pcs: A classifier system that builds a predictive internal world model. In Luigia Carlucci Aiello, editor, *ECAI 90 9th European Conference on Artificial Intelligence*, pages 622–627, August 6–10 1990, Stockholm, Sweden. Pitman Publishing, London, UK.

[1943] Christian Spieth, Felix Streichert, Nora Speer, and Andreas Zell. Utilizing an island model for ea to preserve solution diversity for inferring gene regulatory networks. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2004)*, volume 1, pages 146–151, 2004. In proceedings [1004]. Online available at http://

www-ra.informatik.uni-tuebingen.de/software/JCell/publications.html [accessed 2007-08-24].

[1944] *Advances in Learning Classifier Systems, Revices Papers of the Fourth International Workshop on Learning Classifier Systems (IWLCS-2001)*, volume 2321/2002 of *Lecture Notes in Computer Science (LNCS)*, July 7–8, 2001, San Francisco, CA, USA. Springer. ISBN: 3-5404-3793-2. Held during GECCO-2001 (see also [1937]), published in 2002.

[1945] *Genetic Programming Theory and Practice V, Proceedings of the Genetic Programming Theory Practice 2007 Workshop (GPTP-2007)*, Genetic and Evolutionary Computation, May 17-19, 2007, The Center for the Study of Complex Systems (CSCS), University of Michigan, Ann Arbor, Michigan, USA. Springer. See http://www.cscs.umich.edu/gptp-workshops/gptp2007/ [accessed 2007-09-28].

[1946] *The Tenth International Workshop on Learning Classifier Systems (IWLCS 2007)*, Lecture Notes in Computer Science (LNCS) subseries Lecture Notes in Artificial Intelligence series (LNAI), July 8, 2007, University College London, in London, England. Springer. Held in association with GECCO-2007 (see also [2037, 2038]).

[1947] *Sixth International Conference on Ant Colony Optimization and Swarm Intelligence – ANTS 2008*, Lecture Notes in Computer Science (LNCS), September 22–28, 2008, Brussels, Belgium. Springer.

[1948] *Proceedings of 10th International Conference on Parallel Problem Solving from Nature – PPSN X*, Lecture Notes in Computer Science (LNCS), September 13–17, 2008, Technische Universität Dortmund / Kongresszentrum Westfalenhallen, Dortmund, Germany. Springer. See http://ls11-www.cs.uni-dortmund.de/ppsn/ppsn10/ [accessed 2008-07-20].

[1949] Giovanni Squillero. Microgp – an evolutionary assembly program generator. *Genetic Programming and Evolvable Machines*, 6(3):247–263, September 2005. ISSN: 1389-2576 (Print) 1573-7632 (Online). doi:10.1007/s10710-005-2985-x. Online available at http://www.springerlink.com/content/q043w03275526m36/fulltext.pdf [accessed 2008-09-17].

[1950] N. S. Sridharan, editor. *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, volume 1, August 1989, Detroit, MI, USA. Morgan Kaufmann, San Francisco, CA, USA. ISBN: 1-5586-0094-9. Online available at http://dli.iiit.ac.in/ijcai/IJCAI-89-VOL1/CONTENT/content.htm [accessed 2008-04-01]. See also [1951].

[1951] N. S. Sridharan, editor. *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, volume 2, August 1989, Detroit, MI, USA. Morgan Kaufmann, San Francisco, CA, USA. ISBN: 1-5586-0094-9. Online available at http://dli.iiit.ac.in/ijcai/IJCAI-89-VOL2/CONTENT/content.htm [accessed 2008-04-01]. See also [1950].

[1952] Marc St-Hilaire, Steven Chamberland, and Samuel Pierre. A local search heuristic for the global planning of umts networks. In Seizo Onoe, Mohsen Guizani, Hsiao-Hwa Chen, and Mamoru Sawahashi, editors, *Proceedings of the International Conference on Wireless Communications and Mobile Computing, IWCMC 2006*, pages 1405–1410, July 3–6 2006, Vancouver, British Columbia, Canada. ACM, New York, NY, USA. ISBN: 1-5959-3306-9. doi:10.1145/1143549.1143831. SESSION: R2-D: general symposium. Online available at http://doi.acm.org/10.1145/1143549.1143831 [accessed 2008-08-01].

[1953] Marc St-Hilaire, Steven Chamberland, and Samuel Pierre. A tabu search heuristic for the global planning of umts networks. In *Proceedings of the IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob'2006)*, pages 148–151, June 19–21, 2006, Montréal, Québec, Canada. ISBN: 1-4244-0494-0. INSPEC Accession Number: 9132969. doi:10.1109/WIMOB.2006.1696339.

[1954] Peter F. Stadler. Fitness landscapes. In Michael Lässig and Angelo Valleriani, editors, *Biological Evolution and Statistical Physics*, Lecture Notes in Physics, pages 187–207. Springer-Verlag, Berlin, 2002. ISBN: 3-5404-3188-8, 978-3-54043-188-6. doi:10.1007/3-540-45692-9. Based on his talk *The Structure of Fitness Landscapes* given at the International Workshop on Biological Evolution and Statistical Physics, May 10–14, 2000, located at the Max-Planck-Institut für Physik komplexer Systeme, Nöthnitzer Str. 38, D-01187 Dresden, Germany. Online available at http://www.bioinf.uni-leipzig.de/~studla/Publications/PREPRINTS/01-pfs-004.pdf and http://citeseer.ist.psu.edu/stadler02fitness.html [accessed 2008-03-24].

[1955] Peter Stagge and Christian Igel. Structure optimization and isomorphisms. In *Theoretical Aspects of Evolutionary Computing*, pages 409–422. Springer, 2000. In collection [1083]. Online available at http://citeseer.ist.psu.edu/520775.html [accessed 2007-08-13].

[1956] *Late Breaking Papers at the 1997 Genetic Programming Conference*, July 13-16, 1997, Stanford University, CA, USA. Stanford Bookstore. ISBN: 0-1820-6995-8. See also [1208].

[1957] Kenneth O. Stanley and Risto Miikkulainen. A taxonomy for artificial embryogeny. *Artificial Life*, 9(2):93–130, 2003. ISSN: 1064-5462. Online available at http://nn.cs.utexas.edu/downloads/papers/stanley.alife03.pdf [accessed 2007-08-17].

[1958] Pawel A. Stefanski. Genetic programming using abstract syntax trees, 1993. Notes from Genetic Programming Workshop at ICGA-93 [730]. Online available at http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/icga93-gp_stefanski.html [accessed 2007-08-15] (The file is actually a zip-archive.).

[1959] Ralf Steinmetz and Klaus Wehrle. Peer-to-peer-networking & -computing – aktuelles schlagwort. *Informatik Spektrum*, 27(1):51–54, 2004. Online available at http://www.springerlink.com/content/up3vdx3cnu1a4wb3/fulltext.pdf [accessed 2007-08-13].

[1960] Christopher R. Stephens, Marc Toussaint, Darrell L. Whitley, and Peter F. Stadler, editors. *Revised Selected Papers of the 9th International Workshop on Foundations of Genetic Algorithms IX, FOGA 2007*, volume 4436/2007 of *Lecture Notes in Computer Science (LNCS)*, January 8–11, 2007, Ciudad Universitaria ("University City"), Universidad Nacional Autonoma de Mexico, Mexico City, México. Springer, Berlin Heidelberg. ISBN: 978-3-54073-479-6. doi:10.1007/978-3-540-73482-6. see http://www.sigevo.org/foga-2007/index.html [accessed 2007-09-01].

[1961] Ralph E. Steuer. *Multiple Criteria Optimization: Theory, Computation and Application*. Krieger Pub Co, reprint edition, August 1989. ISBN: 978-0-89464-393-4.

[1962] Terry Stewart. Extrema selection: accelerated evolution on neutral networks. In *Congress on Evolutionary Computation*, volume 1, 2001. doi:10.1109/CEC.2001.934366. In proceedings [1003]. Online available at http://rob.ccmlab.ca:8080/~terry/papers/2001-Extrema_Selection.pdf and http://citeseer.ist.psu.edu/654440.html [accessed 2008-06-01].

[1963] Theo Stewart, editor. *Proceedings of the 13th International Conference on Multiple Criteria Decision Making: Trends in Multicriteria Decision Making (MCDM'1997)*, volume 465 of *Lecture Notes in Economics and Mathematical Systems*, January 6–10, 1997, University of Cape Town, Cape Town, South Africa. Springer-Verlag Telos. ISBN: 978-3-54064-741-6. See http://www.uct.ac.za/depts/maths/mcdm97 [accessed 2007-09-10]. Published January 15, 1999.

[1964] T. R. Stickland, C. M. N. Tofts, and N. R. Franks. A path choice algorithm for ants. *Naturwissenschaften*, 79(12):567–572, December 1992. ISSN: 0028-1042 (Print) 1432-1904 (Online). doi:10.1007/BF01131415. Online available at http://www.springerlink.com/content/v2t8328844843156/fulltext.pdf [accessed 2008-06-12].

[1965] Kilian Stoffel and Lee Spector. High-performance, parallel, stack-based genetic programming. In *Genetic Programming 1996: Proceedings of the First Annual Confer-*

ence, pages 224–229, 1996. In proceedings [1208]. Online available at http://cognet.mit.edu/library/books/view?isbn=0262611279 and http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/stoffel_1996_hpsbGP.html [accessed 2008-06-20].

[1966] Ivan Stojmenović, editor. *Handbook of Sensor Networks: Algorithms and Architectures*. Wiley-Interscience, October 5, 2005. ISBN: 978-0-47168-472-5.

[1967] Robert R. Stoll. *Set Theory and Logic*. Dover Publications, reprint edition, October 1, 1979. ISBN: 0-4866-3829-4, 978-0-48663-829-4.

[1968] Tobias Storch and Ingo Wegener. Real royal road functions for constant population size. In *Genetic and Evolutionary Computation – GECCO 2003*, pages 1406–1417, 2003. doi:10.1007/3-540-45110-2_14. In proceedings [335]. See also [1969, 1970].

[1969] Tobias Storch and Ingo Wegener. Real royal road functions for constant population size. Computational Intelligence CI-167/04, Design and Management of Complex Technical Processes and Systems by means of Computational Intelligence Methods, Collaborative Research Center 531, University of Dortmund, February 2004. Secretary of the SFB 531. Online available at http://hdl.handle.net/2003/5456 [accessed 2008-07-22]. See also [1968, 1970].

[1970] Tobias Storch and Ingo Wegener. Real royal road functions for constant population size. *Theoretical Computer Science*, 320(1):123–134, June 12, 2004. ISSN: 0304-3975. doi:10.1016/j.tcs.2004.03.047. Online available at http://dx.doi.org/10.1016/j.tcs.2004.03.047 [accessed 2008-07-22]. See also [1968, 1969].

[1971] Rainer Storn. Differential evolution design of an IIR-filter with requirements for magnitude and group delay. Technical Report TR-95-026, International Computer Science Institute, 1947 Center Street, Berkeley, CA 94704-1198, Berkeley, CA, 1995. Online available at http://citeseer.ist.psu.edu/storn95differential.html and http://www.icsi.berkeley.edu/ftp/pub/techreports/1995/tr-95-026.pdf [accessed 2007-08-13].

[1972] Rainer Storn. On the usage of differential evolution for function optimization. In M. Smith, M. Lee, J. Keller, and J. Yen, editors, *1996 Biennial Conference of the North American Fuzzy Information Processing Society*, pages 519–523, 1996. IEEE Press, Piscataway, NJ.

[1973] Rainer Storn. Designing digital filters with differential evolution. In *New Ideas in Optimization*, pages 109–125. McGraw-Hill Education, 1999. In collection [448].

[1974] Rainer Storn and Kenneth Price. Differential evolution – a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, International Computer Science Institute, 1947 Center Street, Berkeley, CA 94704, Berkeley, CA, 1995. Online available at http://citeseer.ist.psu.edu/182432.html and http://http.icsi.berkeley.edu/~storn/TR-95-012.pdf [accessed 2007-08-13].

[1975] Felix Streichert. Evolutionäre algorithmen: Implementation und anwendungen im asset-management-bereich (evolutionary algorithms and their application to asset management). Master's thesis, Institut A für Mechanik, Universitä Stuttgart, August 2001. Supervisors: Arnold Kirstner and Werner Koch. Online available at http://www-ra.informatik.uni-tuebingen.de/mitarb/streiche/ [accessed 2007-08-17].

[1976] Bernd Streitberg and Joachim Röhmel. Exact distributions for permutations and rank tests: An introduction to some recently published algorithms. *Statistical Software Newsletter*, 12(1):10–17, 1986. ISSN: 0173-5896.

[1977] Bernd Streitberg and Joachim Röhmel. Exakte verteilungen für rang- und randomisierungstests im allgemeinen c-stichprobenfall. *EDV in Medizin und Biologie*, 18(1):12–19, 1987. ISSN: 0300-8282. See also [1976].

[1978] Vinod Subramanian, Rajkumar Arumugam, and Ali A. Minai. Self-organization of connectivity and geographical routing in large-scale sensor networks. In Ali Minai, Dan Braha, Helen Harte, Larry Rudolph, Temple Smith, Gunter Wagner, and Yaneer Bar-Yam, editors, *Proceedings of the Fourth International Conference on Complex*

Systems, June 9–14, 2002, Nashua, NH. Online available at http://necsi.org/events/iccs/2002/NAp07_subramanian_iccs4-1.pdf [accessed 2007-08-01].

[1979] P. N. Suganthan, N. Hansen, J. J. Liang, Kalyanmoy Deb, Y. P. Chen, Anne Auger, and S. Tiwari. Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization. KanGAL Report 2005005, Kanpur Genetic Algorithms Laboratory, KanGAL, Indian Institute of Technology Kanpur, India, May 2005. Online available at http://www.iitk.ac.in/kangal/papers/k2005005.pdf [accessed 2007-10-07]. See also [1980, 449].

[1980] P. N. Suganthan, N. Hansen, J. J. Liang, Kalyanmoy Deb, Y. P. Chen, Anne Auger, and S. Tiwari. Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization. Technical Report, Nanyang Technological University, Singapore, May 2005. Online available at http://www.ntu.edu.sg/home/epnsugan/index_files/CEC-05/Tech-Report-May-30-05.pdf [accessed 2007-10-07]. See also [1979, 449].

[1981] Berk Sunar, William J. Martin, and Douglas R. Stinson. A provably secure true random number generator with built-in tolerance to active attacks. *IEEE Transactions on Computers*, 56(1):109–119, 2007. ISSN: 0018-9340. doi:10.1109/TC.2007.4. Online available at http://www.cacr.math.uwaterloo.ca/~dstinson/papers/rng-IEEE.pdf [accessed 2007-09-15].

[1982] Erik Sundermann and Ignace L. Lemahieu. Pet image reconstruction using simulated annealing. In Murray H. Loew, editor, *Proceedings of the Society of Photo-Optical Instrumentation Engineers (SPIE) Conference, Medical Imaging 1995: Image Processing*, volume 2434, pages 378–386, May 1995. doi:10.1117/12.208709.

[1983] Patrick David Surry. *A Prescriptive Formalism for Constructing Domain-specific Evolutionary Algorithms*. PhD thesis, University of Edinburgh, 1998. Online available at http://citeseer.ist.psu.edu/258999.html [accessed 2008-07-22].

[1984] Herb Sutter and James Larus. Software and the concurrency revolution. *Queue*, 3(7): 54–62, 2005. ISSN: 1542-7730. Online available at http://research.microsoft.com/~larus/Papers/queue01.pdf and http://portal.acm.org/citation.cfm?id=1095421 [accessed 2007-08-13].

[1985] Reiji Suzuki and Takaya Arita. Repeated occurrences of the baldwin effect can guide evolution on rugged fitness landscapes. In *Proceedings of the IEEE Symposium on Artificial Life (CI-ALife'07)*, pages 8–14. Omnipress, April 1–5, 2007. doi:10.1109/ALIFE.2007.367650. Online available at http://www.alife.cs.is.nagoya-u.ac.jp/~reiji/publications/2007_ieeealife_suzuki.pdf [accessed 2008-09-08].

[1986] William R. Swartout, Paul Rosenbloom, and Peter Szolovits, editors. *Proceedings of the 10th National Conference on Artificial Intelligence, AAAI*, July 12–16, 1992, San Jose, California, USA. The AAAI Press/The MIT Press. ISBN: 0-2625-1063-4. See http://www.aaai.org/Conferences/AAAI/aaai92.php [accessed 2007-09-06].

[1987] Gilbert Syswerda. A study of reproduction in generational and steady state genetic algorithms. In *Proceedings of the First Workshop on Foundations of Genetic Algorithms FOGA*, pages 94–101, July 1990. In proceedings [1924].

[1988] Francis R. Szabo and Paul E. Kladitis. Design, modeling and testing of polysilicon optothermal actuators for power scavenging wireless microrobots. In *Proceedings of the 2004 International Conference on MEMS, NANO and Smart Systems, 2004. ICMENS 2004*, pages 446–452, August 25–27, 2004. IEEE Computer Society, Los Alamitos, CA, USA. ISBN: 0-7695-2189-4.

[1989] M. Szachniuk, Ł. Popenda, Z. Gdaniec, R.W. Adamiak, and J. Błażewicz. Nmr analysis of rna bulged structures: Tabu search application in noe signal assignment. In *Proceedings of the 2005 IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology CIBCB'05*, pages 172–178. IEEE, November

2005, San Diego, USA. Online available at http://www.man.poznan.pl/~lpopenda/HomePage/pub/CIBCB2005172.pdf [accessed 2007-08-25].

[1990] Robert Szewczyk, Joe Polastre, Alan Mainwaring, John Anderson, and David Culler. An analysis of a large scale habitat monitoring application. In *Proceedings of The Second ACM Conference on Embedded Networked Sensor Systems SenSys 2004*, pages 214–226, November 3–5, 2004, Baltimore, MD. Online available at http://www.eecs.harvard.edu/~mdw/course/cs263/papers/ and http://portal.acm.org/citation.cfm?doid=1031495.1031521 [accessed 2007-08-01].

[1991] George G. Szpiro. A search for hidden relationships: Data mining with genetic algorithms. *Computational Economics*, 10(3):267–277, August 1997. Online available at http://citeseer.ist.psu.edu/459445.html and http://ideas.repec.org/a/kap/compec/v10y1997i3p267-77.html [accessed 2007-07-03].

T

[1992] Heidi A. Taboada and David W. Coit. Data clustering of solutions for multiple objective system reliability optimization problems. *Quality Technology and Quantitative Management Journal*, 4(2):35–54, June 2007. Online available at http://www.soe.rutgers.edu/ie/research/working_paper/paper05-019.pdf [accessed 2007-09-10].

[1993] Heidi A. Taboada, Fatema Baheranwala, David W. Coit, and Naruemon Wattanapongsakorn. Practical solutions for multi-objective optimization: An application to system reliability design problems. *Reliability Engineering & System Safety*, 92 (3):314–322, March 2007. doi:10.1016/j.ress.2006.04.014. Online available at http://www.rci.rutgers.edu/~coit/RESS_2007.pdf [accessed 2007-09-10].

[1994] Walter Alden Tackett. *Recombination, Selection, and the Genetic Construction of Computer Programs*. PhD thesis, Faculty of the Graduate School of the University of Southern California, Los Angeles, CA 90089-2562, USA, April 17, 1994. Online available at http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/ftp.io.com/papers/watphd.tar.Z [accessed 2007-09-07]. CENG Technical Report 94-13.

[1995] Genichi Taguchi. *Introduction to Quality Engineering: Designing Quality into Products and Processes*. Asian Productivity Organization / American Supplier Institute Inc. / Quality Resources / Productivity Press Inc., January 1986. ISBN: 9-2833-1083-7, 978-9-28331-083-9, 9-2833-1084-5. Translation of Sekkeisha no tame no hinshitsu kanri.

[1996] Éric D. Taillard, Luca Maria Gambardella, Michael Gendrau, and Jean-Yves Potvin. Adaptive memory programming: A unified view of metaheuristics. *European Journal of Operational Research*, 135(1):1–16, November 6, 2001. ISSN: 0377-2217. doi:10.1016/S0377-2217(00)00268-X. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.54.8460 and http://ina2.eivd.ch/Collaborateurs/etd/articles.dir/ejor135_1_1_16.pdf [accessed 2008-11-11].

[1997] Hideyuki Takagi. Active user intervention in an ec search. In *Proceedings of International Conference on Information Sciences (JCIS2000)*, pages 995–998, 2000. In proceedings [2154]. Online available at http://www.kyushu-id.ac.jp/~takagi/TAKAGI/IECpaper/JCIS2K_2.pdf [accessed 2007-08-29].

[1998] Hideyuki Takagi. Interactive evolutionary computation: Fusion of the capacities of ec optimization and human evaluation. *Proceedings of the IEEE*, 89(9):1275–1296, 2001. Online available at http://www.design.kyushu-u.ac.jp/~takagi/TAKAGI/IECsurvey.html [accessed 2007-08-29].

[1999] Tetsuyuki Takahama and Setsuko Sakai. Constrained optimization by applying the α constrained method to the nonlinear simplex method with mutations. *IEEE Transactions on Evolutionary Computation*, 9(5):437–451, October 3, 2005. ISSN: 1089-778X. INSPEC Accession Number: 8608336. doi:10.1109/TEVC.2005.850256. Online

available at http://www.chi.its.hiroshima-cu.ac.jp/~takahama/eng/papers/aSimplex-TEC2005.pdf [accessed 2008-07-23].

[2000] El-Ghazali Talbi, Pierre Liardet, Pierre Collet, Evelyne Lutton, and Marc Schoenauer, editors. *Revised Selected Papers of the 7th International Conference on Artificial Evolution, Evolution Artificielle, EA 2005*, volume 3871 of *Lecture Notes in Computer Science (LNCS)*, October 26–28, 2005, Lille, France. Springer Berlin/Heidelberg. ISBN: 3-5403-3589-7. Published in 2006.

[2001] Cher Ming Tan, editor. *Simulated Annealing*. IN-TECH Education and Publishing, Kirchengasse 43/3, 1070 Vienna, Austria, September 2008. ISBN: 978-9-53761-907-7. Online available at http://intechweb.org/downloadfinal.php?is=978-953-7619-27-5&type=B [accessed 2009-01-13].

[2002] Kay Chen Tan, Eik Fun Khor, Tong Heng Lee, and Ramasubramanian Sathikannan. An evolutionary algorithm with advanced goal and priority specification for multi-objective optimization. *Journal of Artificial Intelligence Research*, 18:183–215, February 2003. Online available at http://www.jair.org/media/842/live-842-1969-jair.pdf and http://citeseer.ist.psu.edu/tan03evolutionary.html [accessed 2007-08-25].

[2003] Kay Chen Tan, Arthur Tay, and Ji Cai. Design and implementation of a distributed evolutionary computing software. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 33(3):325–338, August 2003. doi:10.1109/TSMCC.2003.817359.

[2004] L. G. Tan and Mark C. Sinclair. Wavelength assignment between the central nodes of the cost 239 european optical network. In *Proceedings of the 11th UK Performance Engineering Workshop, UKPEW'95*, pages 235–247. Springer, September 5–9, 1995, Liverpool John Moores University, Liverpool, UK. Online available at http://citeseer.ist.psu.edu/99015.html and http://uk.geocities.com/markcsinclair/ps/ukpew11_tan.ps.gz [accessed 2008-07-29].

[2005] Ming Tan, Hong-Bin Fang, Guo-Liang Tian, and Gang Wei. Testing multivariate normality in incomplete data of small sample size. *Journal of Multivariate Analysis*, 93(1):164–179, 2005. ISSN: 0047-259X. Online available at http://dx.doi.org/10.1016/j.jmva.2004.02.014 [accessed 2007-09-15].

[2006] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems. Principles and Paradigms*. Pearson/Prentice Hall International, March 2003. ISBN: 0-1323-9227-5, 978-0-13239-227-3, 978-0-13121-786-7. Some information available at http://www.cs.vu.nl/~ast/books/ds1/ [accessed 2007-08-13].

[2007] Reiko Tanese. Distributed genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 434–439, 1989. In proceedings [1820].

[2008] Hui-Chin Tang. Combined random number generator via the generalized chinese remainder theorem. *Journal of Computational and Applied Mathematics*, 142(2):377–388, May 15, 2002. ISSN: 0377-0427. doi:10.1016/S0377-0427(01)00424-1. Online available at http://dx.doi.org/10.1016/S0377-0427(01)00424-1 [accessed 2007-09-16].

[2009] Jane Tateson, Christopher Roadknight, Antonio Gonzalez, Taimur Khan, Steve Fitz, Ian Henning, Nathan Boyd, Chris Vincent, and Ian Marshall. Real world issues in deploying a wireless sensor network for oceanography. In *Workshop on Real-World Wireless Sensor Networks REALWSN'05*, June 20–21, 2005. Stockholm, Sweden. Online available at http://www.cs.kent.ac.uk/pubs/2005/2209/content.pdf and http://www.sics.se/realwsn05/papers/tateson05realworld.pdf [accessed 2007-09-15].

[2010] Gerard Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, second edition, January 2004. ISBN: 0-5217-9483-8, 978-0-52179-483-1. Partly online available at http://books.google.de/books?id=mhcS-eBnbpYC [accessed 2008-11-25].

[2011] Astro Teller. Genetic programming, indexed memory, the halting problem, and other curiosities. In *Proceedings of the 7th annual Florida Artificial Intelligence Re-*

search Symposium, pages 270–274, May 1994. IEEE Press, Pensacola, Florida, USA. Online available at http://www.cs.cmu.edu/afs/cs/usr/astro/public/papers/Curiosities.ps and http://citeseer.ist.psu.edu/256509.html [accessed 2008-06-16].

[2012] Astro Teller. Turing completeness in the language of genetic programming with indexed memory. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, pages 136–141, 1994. doi:10.1109/ICEC.1994.350027. In proceedings [1411]. Online available at http://www.cs.cmu.edu/afs/cs/usr/astro/public/papers/Turing.ps and http://www.astroteller.net/work/pdfs/Turing.pdf [accessed 2007-09-17].

[2013] Astro Teller. Evolving programmers: The co-evolution of intelligent recombination operators. In *Advances in Genetic Programming 2*, chapter 3, pages 45–68. MIT Press, 1996. In collection [61]. Online available at http://www.cs.cmu.edu/afs/cs/usr/astro/public/papers/AiGPII.ps and http://www.cs.cmu.edu/afs/cs/usr/astro/mosaic/chapterII/chapterII.html [accessed 2008-06-16].

[2014] Astro Teller and Manuela Veloso. Algorithm evolution for face recognition: What makes a picture difficult. In *International Conference on Evolutionary Computation*, pages 608–613, 1995. In proceedings [1000]. Online available at http://www.cs.cmu.edu/afs/cs/usr/astro/public/papers/icecFinal.ps and http://citeseer.ist.psu.edu/558916.html [accessed 2008-06-16].

[2015] Astro Teller and Manuela Veloso. PADO: Learning tree structured algorithms for orchestration into an object recognition system. Technical Report CMU-CS-95-101, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA, February 10, 1995. Online available at http://www.cs.cmu.edu/afs/cs/usr/astro/public/papers/PADO-Tech-Report.ps and http://citeseer.ist.psu.edu/teller95pado.html [accessed 2008-06-16].

[2016] Astro Teller and Manuela Veloso. PADO: A new learning architecture for object recognition. In Katsushi Ikeuchi and Manuela Veloso, editors, *Symbolic Visual Learning*, pages 81–116. Oxford University Press, Inc., New York, NY, USA, May 1, 1996. ISBN: 0-1950-9870-6, 978-0-19509-870-9. Online available at http://www.cs.cmu.edu/afs/cs/usr/astro/public/papers/PADO.ps and http://citeseer.ist.psu.edu/564427.html [accessed 2008-06-16].

[2017] Alexandre Temporel and Tim Kovacs. A heuristic hill climbing algorithm for mastermind. In *Proceedings of the 2003 UK Workshop on Computational Intelligence (UKCI-03)*, pages 189–196. University of Bristol, September 1–3, 2003, Department of Engineering Mathematics and Department of Computer Science The University of Bristol, UK. ISBN: 0-8629-2537-1. Online available at http://citeseer.ist.psu.edu/701434.html and http://www.cs.bris.ac.uk/Publications/Papers/2000067.pdf [accessed 2007-09-11].

[2018] Liliana Teodorescu. High energy physics data analysis with gene expression programming. In *Nuclear Science Symposium Conference Record*, volume 1, pages 143–147. IEEE, October 23–29, 2005. INSPEC Accession Number: 8976991. doi:10.1109/NSSMIC.2005.1596225.

[2019] Igor V. Tetko, David J. Livingstone, and Alexander I. Luik. Neural network studies, 1. comparison of overfitting and overtraining. *Journal of Chemical Information and Computer Sciences*, 35(5):826–833, 1995.

[2020] Sam R. Thangiah. Vehicle routing with time windows using genetic algorithms. In *Practical Handbook of Genetic Algorithms: New Frontiers*, pages 253–277. CRC Press, Inc., 1995. In collection [369]. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.23.7903 [accessed 2008-10-27].

[2021] *Proceedings of the Fourth Joint Conference on Information Science (JCIS 1998), Section: The Second International Workshop on Frontiers in Evolutionary Algorithms (FEA 1998)*, volume 2, October 24–28, 1998, Research Triangle Park, North Carolina,

USA. The Association for Intelligent Machinery. Workshop held in conjunction with Fourth Joint Conference on Information Sciences.

[2022] *Sixth International Workshop on Learning Classifier Systems (IWLCS-2003)*, July 12–16, 2003, The Holiday Inn Chicago, Chicago, IL 60654, USA. Held during GECCO-2003 (see also [334, 335]). See also [1181].

[2023] *AISB'00 Convention: Artificial Intelligence and Society*, April 17–21, 2000, University of Birmingham, England, UK. The Society for the Study of Artificial Intelligence and the Simulation of Behaviour (SSAISB). Online available at http://www.aisb.org.uk/publications/proceedings.shtml [accessed 2008-09-10].

[2024] *AISB'01 Convention: Agents and Cognition*, March 21–24, 2001, University of York, UK. The Society for the Study of Artificial Intelligence and the Simulation of Behaviour (SSAISB). ISBN: 1-9029-5619-7, 1-9029-5622-1, 1-9029-5617-0, 1-9029-5618-9, 1-9029-5621-3, 1-9029-5620-5.

[2025] *AISB'02 Convention: Logic, Language and Learning*, April 3–5, 2002, Depts. of Computing and Electrical & Electronic Engineering, Imperial College, UK. The Society for the Study of Artificial Intelligence and the Simulation of Behaviour (SSAISB). Online available at http://www.aisb.org.uk/publications/proceedings.shtml [accessed 2008-09-10].

[2026] *AISB 2003 Convention: Cognition in Machines and Animals*, April 7–11, 2003, University of Wales, Aberystwyth, UK. The Society for the Study of Artificial Intelligence and the Simulation of Behaviour (SSAISB). Online available at http://www.aisb.org.uk/publications/proceedings.shtml [accessed 2008-09-10].

[2027] *AISB 2004 Convention: Motion, Emotion and Cognition*, March 29–April 1, 2004, University of Leeds, UK. The Society for the Study of Artificial Intelligence and the Simulation of Behaviour (SSAISB). Online available at http://www.aisb.org.uk/publications/proceedings.shtml [accessed 2008-09-10].

[2028] *AISB 2005: Social Intelligence and Interaction in Animals, Robots and Agents*, April 12–15, 2005, University of Hertfordshire, Hatfield, England, UK. The Society for the Study of Artificial Intelligence and the Simulation of Behaviour (SSAISB). Online available at http://www.aisb.org.uk/publications/proceedings.shtml [accessed 2008-09-10].

[2029] *AISB'06: Adaptation in Artificial and Biological Systems*, April 3–6, 2006, University of Bristol, Bristol, England. The Society for the Study of Artificial Intelligence and the Simulation of Behaviour (SSAISB). Online available at http://www.aisb.org.uk/publications/proceedings.shtml [accessed 2008-09-10].

[2030] *AISB'07 Artificial and Ambient Intelligence*, April 1–4, 2007, Newcastle University, Newcastle upon Tyne, UK. The Society for the Study of Artificial Intelligence and the Simulation of Behaviour (SSAISB). Online available at http://www.aisb.org.uk/publications/proceedings.shtml [accessed 2008-09-10].

[2031] Beem database combined results – the beem database – tool performance, March 31, 2007. The SPIN performance in the BEEM example problems. Online available at http://spinroot.com/spin/beem.html [accessed 2008-10-02]. Database online available at http://anna.fi.muni.cz/models/ [accessed 2008-10-02], see also [1631].

[2032] Guy Théraulaz and Eric Bonabeau. Coordination in distributed building. *Science*, 269(5224):686–688, August 4, 1995. doi:10.1126/science.269.5224.686.

[2033] Guy Théraulaz and Eric Bonabeau. Swarm smarts. *Scientific American*, pages 72–79, March 2000. Online available at http://www.santafe.edu/~vince/press/SciAm-SwarmSmarts.pdf [accessed 2008-06-12].

[2034] Dirk Thierens. On the scalability of simple genetic algorithms. Technical Report UU-CS-1999-48, Department of Information and Computing Sciences, Utrecht University, 1999. Persistent Identifier: URN:NBN:NL:UI:10-1874/18986. Online available at http://igitur-archive.library.uu.nl/math/2007-0118-201740/thierens_99_on_the_scalability.pdf and http://www.cs.uu.nl/research/

techreps/repo/CS-1999/1999-48.pdf [accessed 2008-08-12], , online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.40.5295 [accessed 2008-08-12].

[2035] Dirk Thierens and David E. Goldberg. Convergence models of genetic algorithm selection schemes. In *PPSN III: Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature*, pages 119–129, 1994. In proceedings [492]. Online available at http://www.cs.uu.nl/groups/DSS/publications/convergence/convMdl.ps [accessed 2008-07-21].

[2036] Dirk Thierens, David E. Goldberg, and Ângela Guimarães Pereira. Domino convergence, drift, and the temporal-salience structure of problems. In *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, pages 535–540, 1998. doi:10.1109/ICEC.1998.700085. In proceedings [1001]. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.46.6116 [accessed 2008-08-12].

[2037] Dirk Thierens, Hans-Georg Beyer, Josh C. Bongard, Jurgen Branke, John Andrew Clark, Dave Cliff, Clare Bates Congdon, Kalyanmoy Deb, Benjamin Doerr, Tim Kovacs, Sanjeev Kumar, Julian Francis Miller, Jason Moore, Frank Neumann, Martin Pelikan, Riccardo Poli, Kumara Sastry, Kenneth Owen Stanley, Thomas Stützle, Richard A. Watson, and Ingo Wegener, editors. *Proceedings of Genetic and Evolutionary Computation Conference, GECCO 2007*, July 7–12, 2007, University College London, Gower Street, London WC 1E 6BT, London, UK. ACM Press, New York, NY, USA. ISBN: 978-1-59593-697-4. See also [2038].

[2038] Dirk Thierens, Hans-Georg Beyer, Josh C. Bongard, Jurgen Branke, John Andrew Clark, Dave Cliff, Clare Bates Congdon, Kalyanmoy Deb, Benjamin Doerr, Tim Kovacs, Sanjeev Kumar, Julian Francis Miller, Jason Moore, Frank Neumann, Martin Pelikan, Riccardo Poli, Kumara Sastry, Kenneth Owen Stanley, Thomas Stützle, Richard A. Watson, and Ingo Wegener, editors. *Genetic and Evolutionary Computation Conference – Companion Material, GECCO 2007*, July 7–12, 2007, University College London, Gower Street, London WC 1E 6BT, London, UK. ACM Press, New York, NY, USA. ISBN: 978-1-59593-698-1. See also [2037], ACM Order Number 910071.

[2039] Herve Thiriez and Stanley Zionts, editors. *Proceedings of the 1st International Conference on Multiple Criteria Decision Making (MCDM'1975)*, Lecture notes in economics and mathematical systems, May 21–23, 1975, Jouy-en-Josas, France. Springer. ISBN: 978-0-38707-794-9. Published in 1976.

[2040] Richard K. Thompson and Alden H. Wright. Additively decomposable fitness functions. Technical Report, Computer Science Department, The University of Montana, Missoula, MT 59812-1008, USA, 1996. Online available at http://citeseer.ist.psu.edu/thompson96additively.html and http://www.cs.umt.edu/u/wright/papers/add_decomp.ps.gz [accessed 2007-11-27].

[2041] Henk Tijms. *Understanding Probability: Chance Rules in Everyday Life*. Cambridge University Press, August 16, 2004. ISBN: 0-5215-4036-4, 978-0-52154-036-0.

[2042] Chuan-Kang Ting and Hans Kleine Buning. A mating strategy for multi-parent genetic algorithms by integrating tabu search. In *Proceedings of the 2003 Congress on Evolutionary Computation CEC2003*, volume 2, pages 1259–1266, 2003. INSPEC Accession Number: 8088166. doi:10.1109/CEC.2003.1299813. In proceedings [1803].

[2043] Chuan-Kang Ting, Chungnan Lee, and Sheng-Tun Li. A novel hybrid optimization algorithm based on genetic algorithm and tabu search. ICS 2000, Feng Chia University, October 26, 2006. Online available at http://hdl.handle.net/2377/2598 [accessed 2008-10-25]. Also: Proceedings of International Computer Symposium, Taiwan, pages 157–162, 2000.

[2044] Ashutosh Tiwari, Joshua Knowles, Erel Avineri, Keshav Dahal, and Rajkumar Roy, editors. *Applications of Soft Computing – Recent Trends*, volume 36 of *Advances in*

Soft Computing. Springer-Verlag Berlin and Heidelberg GmbH & Co. KG, June 7, 2006. ISBN: 978-3-54029-123-7. Editor-in-chief: Janusz Kacprzyk.

[2045] Tommaso Toffoli. Reversible computing. In *Proceedings of the 7th Colloquium on Automata, Languages and Programming*, Lecture Notes In Computer Science (LNCS), pages 632–644, 1980. Springer-Verlag, London, UK. ISBN: 3-5401-0003-2.

[2046] Virginia Joanne Torczon. *Multi-Directional Search: A Direct Search Algorithm for Parallel Machines*. PhD thesis, Rice University, Houston, TX, USA, May 1989. Online available at http://citeseer.ist.psu.edu/176729.html [accessed 2008-06-14]. Supervisors: John E. Dennis, Jr., Andrew E. Boyd, Kenneth W. Kennedy, Jr., and Richard A. Tapia.

[2047] Aimo Törn and Antanas Žilinskas. *Global Optimization*, volume 350/1989 of *Lecture Notes in Computer Science (LNCS)*. Springer-Verlag, Berlin Heidelberg New York, 1989. ISBN: 3-5405-0871-6, 0-3875-0871-6, 978-3-54050-871-7. doi:10.1007/3-540-50871-6. Series editors: G. Goos and J. Hartmanis.

[2048] Naoyoshi Toshine, Nobuyuki Iwauchi, Shin'ichi Wakabayashi, T. Koide, and I. Nishimura. A parallel genetic algorithm with adaptive adjustment of genetic parameters. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, pages 679–686, 2001. In proceedings [1937].

[2049] Paolo Toth and Daniele Vigo. The granular tabu search and its application to the vehicle-routing problem. *INFORMS Journal on Computing*, 15(4):333–346, 2003. ISSN: 1526-5528. doi:10.1287/ijoc.15.4.333.24890.

[2050] Marc Toussaint and Christian Igel. Neutrality: A necessity for self-adaptation. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2002)*, pages 1354–1359, 2002. In proceedings [703]. Online available at http://citeseer.ist.psu.edu/toussaint02neutrality.html [accessed 2007-07-28].

[2051] Ioan Cristian Trelea. The particle swarm optimization algorithm: convergence analysis and parameter selection. *Information Processing Letters*, 85(6):317–325, March 31, 2003. doi:10.1016/S0020-0190(02)00447-7. Online available at http://dx.doi.org/10.1016/S0020-0190(02)00447-7 [accessed 2008-08-13].

[2052] Maryjane Tremayne, Colin C. Seaton, and Christopher Glidewell. Structures of three substituted arenesulfonamides from x-ray powder diffraction data using the differential evolution technique. *Acta Crystallographica Section B: Structural Science*, 58: 823–834, October 2002. ISSN: 0108-7681. Online available at http://scripts.iucr.org/cgi-bin/paper?S0108768102011928 [accessed 2007-08-13].

[2053] Krzysztof Trojanowski. *Evolutionary Algorithms with Redundant Genetic Material for Non-Stationary Environments*. PhD thesis, Instytut Podstaw Informatyki PAN, Institute of Computer Science, Warsaw, University of Technology, Poland, 1994. Advisor: Zbigniew Michalewicz.

[2054] Michael W. Trosset. I know it when i see it: Toward a definition of direct search methods. *SIAG/OPT Views and News: A Forum for the SIAM Activity Group on Optimization*, 9:7–10, Fall 1997. Online available at http://citeseer.ist.psu.edu/135600.html and http://www.math.wm.edu/~trosset/Research/Opt/dsm3.ps.Z [accessed 2008-06-14], online available at http://www.mat.uc.pt/siagopt/vn9.pdf [accessed 2008-07-02].

[2055] Athanasios Tsakonas, Georgios Dounias, Jan Jantzen, Hubertus Axer, Beth Bjerregaard, and Diedrich Graf von Keyserlingk. Evolving rule-based systems in two medical domains using genetic programming. *Artificial Intelligence in Medicine*, 32 (3):195–216, 2004. ISSN: 0933-3657. doi:10.1016/j.artmed.2004.02.007. Online available at http://dx.doi.org/10.1016/j.artmed.2004.02.007 [accessed 2007-09-09].

[2056] Constantino Tsallis and Ricardo Ferreira. On the origin of self-replicating information-containing polymers from oligomeric mixtures. *Physics Letters A*, 99(9): 461–463, December 26, 1983. doi:10.1016/0375-9601(83)90958-1. Online available at http://dx.doi.org/10.1016/0375-9601(83)90958-1 [accessed 2008-07-01].

[2057] Chao-Hsi Tsao and Jyh-Long Chern. Application of a global optimization process to the design of pickup heads for compact and digital versatile disks. *Optical Engineering*, 45(10):103001, October 2006. doi:10.1117/1.2361281.

[2058] Christian F. Tschudin. Fraglets – a metabolistic execution model for communication protocols. In *Proceedings of 2nd Annual Symposium on Autonomous Intelligent Networks and Systems (AINS)*, June 30–July 3, 2003, SRI International, Menlo Park, CA, USA. Online available at http://cn.cs.unibas.ch/pub/doc/2003-ains.pdf and http://path.berkeley.edu/ains/final/007%20-%2022-tschudin.pdf [accessed 2008-05-02]. See also http://www.fraglets.net/ [accessed 2007-09-17].

[2059] Christian F. Tschudin and Lidia A. R. Yamamoto. A metabolic approach to protocol resilience. In *Autonomic Communication – Revised Selected Papers from the First International IFIP Workshop, WAC 2004*, volume 3457/2005 of *Lecture Notes in Computer Science (LNCS)*, pages 191–206. Springer Berlin / Heidelberg, October 18–19, 2004, Berlin, Germany. ISBN: 978-3-54027-417-9. doi:10.1007/11520184_15. Online available at http://cn.cs.unibas.ch/pub/doc/2004-wac-lncs.pdf [accessed 2008-06-20], presentation online available at http://www.autonomic-communication.org/wac/wac2004/program/presentations/wac2004-tschudin-yamamoto-slides.pdf [accessed 2008-06-20].

[2060] Shigeyoshi Tsutsui and Ashish Ghosh. Genetic algorithms with a robust solution searching scheme. *IEEE Transactions on Evolutionary Computation*, 1(3):201–208, September 1997. ISSN: 1089-778X. CODEN: ITEVF5. INSPEC Accession Number: 5858813. doi:10.1109/4235.661550.

[2061] Shigeyoshi Tsutsui and David E. Goldberg. Simplex crossover and linkage identification: Single-stage evolution vs. multi-stage evolution. In *Proceedings IEEE International Conference on Evolutionary Computation*, pages 974–979, 2002. doi:10.1109/CEC.2002.1007057. In proceedings [703]. IlliGAL Report No. 2002001 January 2002, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign, 117 Transportation Building, 104 S. Mathews Avenue, Urbana, Illinois 61801, USA. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.16.377 and http://www.illigal.uiuc.edu/web/technical-reports/2002/01/08/simplex-crossover-and-linkage-identification-single-stage-evolution-vs-multi-stage-evolu [accessed 2008-09-10].

[2062] Shigeyoshi Tsutsui, Ashish Ghosh, and Yoshiji Fujimoto. A robust solution searching scheme in genetic search. In *PPSN IV: Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*, pages 543–552, 1996. In proceedings [2118]. Online available at http://www.hannan-u.ac.jp/~tsutsui/ps/ppsn-iv.pdf [accessed 2008-07-19].

[2063] Shigeyoshi Tsutsui, Masayuki Yamamura, and Takahide Higuchi. Multi-parent recombination with simplex crossover in real coded genetic algorithms. In *Proceedings of the 5th International Conference on Artificial Evolution (EA'01)*, pages 73–84, 2001. In proceedings [428]. Online available at http://www2.hannan-u.ac.jp/~tsutsui/ps/icga99.pdf [accessed 2008-09-10].

[2064] Dean M. Tullsen, Susan Eggers, and Henry M. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *Proceedings of the 22th Annual International Symposium on Computer Architecture*, pages 392–403. ACM Press, June 1995, Santa Margherita Ligure, Italy. Online available at http://www.cs.washington.edu/research/smt/papers/ISCA95.ps and http://citeseer.ist.psu.edu/tullsen95simultaneous.html [accessed 2007-09-11].

[2065] Alan Mathison Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42:230–265, 1936. Online available at http://www.thocp.net/biographies/turing_alan.html and http://www.abelard.org/turpap2/tp2-ie.asp [accessed 2007-08-11].

[2066] Alan Mathison Turing. On computable numbers, with an application to the Entscheidungsproblem, errata. *Proceedings of the London Mathematical Society*, 43:544–546, 1937. Online available at the same location as [2065].

[2067] Peter Turney, L. Darrell Whitley, and Russell W. Anderson. Evolution, learning, and instinct: 100 years of the baldwin effect. *Evolutionary Computation*, 4(3):iv–viii, Fall 1996. ISSN: 1063-6560, 1530-9304. doi:10.1162/evco.1996.4.3.iv. Online available at http://www.geocities.com/Athens/4155/edit.html [accessed 2008-09-10].

[2068] *Proceedings of the 4th Workshop on Parallel Processing: Logic, Organisation, and Technology (WOPPLOT 92)*, September 1992, Tutzing, Germany. it is not clear to the author whether this has actually taken place or not.

[2069] G. H. Tzeng, Herbert F. Wang, U. P. Wen, and Po-Lung Yu, editors. *Proceedings of the 10th International Conference on Multiple Criteria Decision Making: Expand and Enrich the Domains of Thinking and Application (MCDM'1992)*, 1992, Taipei, Taiwan. Springer. ISBN: 978-0-38794-297-1. Published in June 1994.

U

[2070] I. Ullah and S. Kota. Globally-optimal synthesis of mechanisms for path generation using simulated annealing and powell's method. In *Proceedings of the 1996 ASME Design Engineering Technical Conferences and Computers in Engineering*, page 8, August 1996, Irvine/CA, USA.

[2071] Christian Ullenboom. *Java ist auch eine Insel – Programmieren mit der Java Standard Edition Version 6*. Galileo Computing, Galileo Press, 6. aktualisierte und erweiterte edition, 2007. ISBN: 3-8984-2838-9. Online available at http://www.galileocomputing.de/openbook/javainsel6/ [accessed 2007-07-03].

[2072] *Proceedings of the Seventh Conference on Evolutionary and Deterministic Mehtods for Design, Optimization and Control with Applications to Indutsrial and Societal Problems, EUROGEN2007*, June 11–13, 2007, University of Jyväskylä, Jyväskylä, Finland. See http://www.mit.jyu.fi/scoma/Eurogen2007/ [accessed 2007-09-16].

[2073] Olgierd Unold and Grzegorz Dabrowski. Use of learning classifier system for inferring natural language grammar. In *Design and Application of Hybrid Intelligent Systems, HIS03, the Third International Conference on Hybrid Intelligent Systems*, pages 272–278, 2003. In proceedings [8] and also [2074].

[2074] Olgierd Unold and Grzegorz Dabrowski. Use of learning classifier system for inferring natural language grammar. In *Revised Selected Papers of the International Workshops on Learning Classifier Systems*, chapter 2, part I, pages 17–24. Springer Berlin/Heidelberg, 2003. doi:10.1007/978-3-540-71231-2_2. In collection [1181] and also [2073].

[2075] Rasmus K. Ursem. *Models for Evolutionary Algorithms and Their Applications in System Identification and Control Optimization*. PhD thesis, Department of Computer Science, University of Aarhus, Denmark, April 1, 2003. Advisors: Thiemo Krink and Brian H. Mayoh. Online available at http://www.daimi.au.dk/~ursem/publications/RKU_thesis_2003.pdf and http://citeseer.ist.psu.edu/572321.html [accessed 2007-08-14].

[2076] *Proceedings of Mendel'95, the 1st International Mendel Conference on Genetic Algorithms on the occasion of 130th anniversary of Mendel's laws*, September 1995, Brno, Czech Republic. Ústav Automatizace a Informatiky. ISBN: 8-0214-0672-0.

[2077] *Proceedings of the 2nd International Mendel Conference on Genetic Algorithms, MENDEL 1996*, June 26–28, 1996, Brno University of Technology, Brno, Czech Republic. Ústav Automatizace a Informatiky. ISBN: 8-0214-0769-7.

[2078] *Proceedings of the 2nd International Mendel Conference on Genetic Algorithms, MENDEL 1997*, June 26–28, 1997, Brno University of Technology, Brno, Czech Republic. Ústav Automatizace a Informatiky. ISBN: 8-0214-0769-7.

[2079] *Proceedings of the 4th International Mendel Conference on Genetic Algorithms, Optimisation Problems, Fuzzy Logic, Neural Networks, Rough Sets*, June 24–26, 1998, Technical University of Brno, Faculty of Mechanical Engineering, Brno, Czech Republic. Ústav Automatizace a Informatiky. ISBN: 8-0214-1131-7.

[2080] *Proceedings of the 3rd International Conference on Genetic Algorithms*, June 1997, Brno University of Technology, Brno, Czech Republic. Ústav Automatizace a Informatiky. ISBN: 8-0214-1131-7.

[2081] *Proceedings of the 8th International Conference on Soft Computing, MENDEL'02*, June 5–7, 2002, Brno University of Technology, Brno, Czech Republic. Ústav Automatizace a Informatiky. ISBN: 8-0214-2135-5.

[2082] *Proceedings of the 9th International Conference on Soft Computing, MENDEL'03*, 2003, Brno University of Technology, Brno, Czech Republic. Ústav Automatizace a Informatiky. ISBN: 8-0214-2411-7.

[2083] *Proceedings of the 10th International Conference on Soft Computing, MENDEL'04*, June 16–18, 2004, Brno University of Technology Faculty of Mechanical Engeneering, Brno, Brno, Czech Republic. Ústav Automatizace a Informatiky. ISBN: 8-0214-2676-4.

[2084] *Proceedings of the 11th International Conference on Soft Computing, MENDEL'05*, June 15–17, 2005, Brno University of Technology, Brno, Brno, Czech Republic. Ústav Automatizace a Informatiky. ISBN: 8-0214-2961-5.

[2085] Jamie Uys. *Animals are Beautiful people*. Warner Brothers, 1974, South Africa.

V

[2086] *Proceedings of the 7th International Conference on Soft Computing, Evolutionary Computation, Genetic Programing, Fuzzy Logic, Rough Sets, Neural Networks, Fractals, Bayesian Methods, MENDEL 2001*, June 6–8, 2001, Brno University of Technology, Brno, Czech Republic. V Opavě: Slezská univerzita, Filozoficko-přírodovědecká fakulta, Ústav informatiky. ISBN: 8-0724-8107-X.

[2087] Rob J. M. Vaessens, Emile H. L. Aarts, and Jan Karel Lenstra. A local search template. In *Parallel Problem Solving from Nature, PPSN II*, pages 67–76, 1992. In proceedings [1357]. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.54.4457 [accessed 2008-11-11]. See also [2088].

[2088] Rob J. M. Vaessens, Emile H. L. Aarts, and Jan Karel Lenstra. A local search template. *Computers and Operations Research*, 25(11):969–979, November 1998. ISSN: 0305-0548. doi:10.1016/S0305-0548(97)00093-2. Online available at http://dx.doi.org/10.1016/S0305-0548(97)00093-2 [accessed 2008-11-11]. See also [2087].

[2089] Werner Van Belle. *Automatic Generation of Concurrency Adaptors by Means of Learning Algorithms*. PhD thesis, Vrije Universiteit Brussel, Faculteit van de Wetenschappen, Departement Informatica, August 2001. Promotor: Theo D'Hondt, co-Promotor: Tom Tourwé. Online available at http://borg.rave.org/adaptors/Thesis6.pdf [accessed 2008-06-23].

[2090] Werner Van Belle, Tom Mens, and Theo D'Hondt. Using genetic programming to generate protocol adaptors for interprocess communication. In Andy M. Tyrrell, Pauline C. Haddow, and Jim Torresen, editors, *Proceedings of the 5th International Conference on Evolvable Systems: From Biology to Hardware. ICES 2003*, volume 2606/2003 of *Lecture Notes in Computer Science (LNCS)*, pages 67–73. Springer Berlin / Heidelberg, March 17–20, 2003, Trondheim, Norway. ISBN: 978-3-54000-730-2. Online available at http://werner.yellowcouch.org/Papers/genadap03/ICES-online.pdf and http://prog.vub.ac.be/Publications/2003/vub-prog-tr-03-33.pdf [accessed 2008-06-23].

[2091] J. Koen van der Hauw. Evaluating and improving steady state evolutionary algorithms on constraint satisfaction problems. Master's thesis, Computer Science Department of Leiden University, August 1996, Computer Science Department of Leiden University. supervisors: Guszti Eiben (≡ Ágoston E. Eiben) and Han La Poutré. Online available at http://citeseer.ist.psu.edu/128231.html [accessed 2007-08-24].

[2092] Christiaan M. van der Walt and Etienne Barnard. Data characteristics that determine classifier performance. In *Proceedings of the Sixteenth Annual Symposium of the Pattern Recognition Association of South Africa*, pages 160–165, November 23–25, 2005. Online available at http://www.meraka.org.za/pubs/CvdWalt.pdf [accessed 2007-08-08].

[2093] Bas C. van Fraassen. *Laws and Symmetry*. Clarendon Paperbacks. Oxford University Press, USA, January 1990. ISBN: 978-0-19824-860-6.

[2094] Jano I. van Hemert and Carlos Cotta, editors. *Evolutionary Computation in Combinatorial Optimization, Proceedings of the 8th European Conference, EvoCOP 2008*, volume 4972/2008 of *Lecture Notes in Computer Science (LNCS), Subseries: SL 1 – Theoretical Computer Science and General Issues*, March 26–28, 2008, Naples, Italy. Springer, Berlin / Heidelberg. ISBN: 3-5407-8603-1, 978-3-54078-603-0. Library of Congress Control Number: 2008922955.

[2095] P. J. M. van Laarhoven and E. H. L. Aarts, editors. *Simulated Annealing: Theory and Applications*. Mathematics and Its Applications. Springer, Kluwer Academic Publishers, Norwell, MA, USA, June 30, 1987. ISBN: 9-0277-2513-6, 978-9-02772-513-4. Reviewed in [982]. Partly online available at http://books.google.de/books?id=-IgUab6Dp_IC [accessed 2008-03-26].

[2096] Erik van Nimwegen and James P. Crutchfield. Optimizing epochal evolutionary search: Population-size dependent theory. *Machine Learning*, 45(1):77–114, October 2001. ISSN: 0885-6125 (Print) 1573-0565 (Online). doi:10.1023/A:1012497308906. Editor: Leslie Pack Kaelbling. Online available at http://www.springerlink.com/content/nv23534338656352/fulltext.pdf [accessed 2008-07-02].

[2097] Erik van Nimwegen, James P. Crutchfield, and Martijn Huynen. Neutral evolution of mutational robustness. *Proceedings of the National Academy of Science of the United States of Americs (PNAS) – Evolution*, 96(17):9716–9720, August 17, 1999. Online available at http://www.pnas.org/cgi/reprint/96/17/9716.pdf [accessed 2008-07-02].

[2098] Erik van Nimwegen, James P. Crutchfield, and Melanie Mitchell. Statistical dynamics of the royal road genetic algorithm. *Theoretical Computer Science*, 229(1–2):41–102, November 6, 1999. ISSN: 0304-3975. doi:10.1016/S0304-3975(99)00119-X. Online available at http://dx.doi.org/10.1016/S0304-3975(99)00119-X and http://citeseer.ist.psu.edu/nimwegen98statistical.html [accessed 2008-06-01].

[2099] Robbert van Renesse. The importance of aggregation. In A. Schiper, A. A. Shvartsman, H. Weatherspoon, and B. Y. Zhao, editors, *Future Directions in Distributed Computing*, volume 2584/2003 of *Lecture Notes in Computer Science (LNCS)*, pages 87–92. Springer Berlin/Heidelberg, 2003.

[2100] Harry L. Van Trees. *Detection, Estimation, and Modulation Theory, Part I*. Wiley-Interscience, reprint edition, February 2007. ISBN: 978-0-47109-517-0.

[2101] David A. van Veldhuizen. *Multiobjective Evolutionary Algorithms: Classifications, Analyses, and New Innovations*. PhD thesis, Air Force Institute of Technology, Air University, Wright-Patterson Air Force Base, Ohio, May 1999. AFIT/DS/ENG/99-01. Online available at http://handle.dtic.mil/100.2/ADA364478 and http://citeseer.ist.psu.edu/vanveldhuizen99multiobjective.html [accessed 2007-08-17].

[2102] David A. Van Veldhuizen, Jesse B. Zydallis, and Gary B. Lamont. Issues in parallelizing multiobjective evolutionary algorithms for real world applications. In *SAC'02: Proceedings of the 2002 ACM symposium on Applied computing*, pages 595–602, 2002, Madrid, Spain. ACM Press, New York, NY, USA. ISBN: 1-5811-3445-2.

doi:http://doi.acm.org/10.1145/508791.508906. Online available at http://doi.acm.org/10.1145/508791.508906 [accessed 2007-08-14].

[2103] Leonardo Vanneschi and Marco Tomassini. A study on fitness distance correlation and problem difficulty for genetic programming. In *Graduate Student Workshop, GECCO Conference*, pages 307–310, 2002. In proceedings [1326]. Online available at http://personal.disco.unimib.it/Vanneschi/GECCO_2002_PHD_WORKSHOP.pdf and http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/vanneschi_2002_gecco_workshop.html [accessed 2008-07-20]. See also [416].

[2104] Leonardo Vanneschi, Manuel Clergue, Philippe Collard, Marco Tomassini, and Sébastien Vérel. Fitness clouds and problem hardness in genetic programming. In *Genetic and Evolutionary Computation – GECCO-2004, Part II*, pages 690–701, 2004. doi:10.1007/b98645. In proceedings [545]. Online available at http://www.i3s.unice.fr/~verel/publi/gecco04-FCandPbHardnessGP.pdf and http://hal.archives-ouvertes.fr/hal-00160055/en/ [accessed 2007-10-14], online available at http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/vanneschi_fca_gecco2004.html and http://hal.inria.fr/docs/00/16/00/55/PDF/fc_gp.pdf [accessed 2008-07-20].

[2105] Leonardo Vanneschi, Marco Tomassini, Philippe Collard, and Sébastien Vérel. Negative slope coefficient. A measure to characterize genetic programming. In *Proceedings of the 9th European Conference on Genetic Programming*, pages 178–189, 2006. In proceedings [429]. Online available at http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/eurogp06_VanneschiTomassiniCollardVerel.html [accessed 2008-07-20].

[2106] Leonardo Vanneschi, Steven Gustafson, Alberto Moraglio, Ivanoe De Falco, and Marc Ebner, editors. *Genetic Programming – Proceedings of the 12th European Conference, EuroGP 2009*, volume 5481/2009 of *Lecture Notes in Computer Science (LNCS), Subseries: SL 1 – Theoretical Computer Science and General Issues*, April 15–17, 2009, Tübingen, Germany. Springer, Berlin / Heidelberg. ISBN: 3-6420-1180-2, 978-3-64201-180-1. doi:10.1007/978-3-642-01181-8.

[2107] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Information Science and Statistics. Springer-Verlag, second edition, November 19, 1999 (first edition: 1995). ISBN: 978-0-38798-780-4.

[2108] Francisco J. Varela and Paul Bourgine, editors. *Toward a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life*, December 11–13, 1991, Paris, France. A Bradford book, The MIT Press. ISBN: 0-2627-2019-1. Published in 1992.

[2109] Athanasios V. Vasilakos, Kostas G. Anagnostakis, and Witold Pedrycz. Application of computational intelligence techniques in active networks. *Soft Computing – A Fusion of Foundations, Methodologies and Applications*, 5(4):264–271, August 2001. ISSN: 1432-7643 (Print) 1433-7479 (Online). doi:10.1007/s005000100100. Online available at http://www.springerlink.com/content/xxuyux0wgmnt73gy/fulltext.pdf [accessed 2008-09-03].

[2110] Vesselin K. Vassilev and Julian Francis Miller. The advantages of landscape neutrality in digital circuit evolution. In *ICES'00: Proceedings of the Third International Conference on Evolvable Systems*, volume 1801/2000, pages 252–263. Springer-Verlag, London, UK, April 17-19, 2000, Edinburgh, Scotland, UK. ISBN: 3-5406-7338-5. Online available at http://citeseer.ist.psu.edu/vassilev00advantages.html [accessed 2007-11-02].

[2111] Sergei Vassilvitskii. How slow is the k-means method? *Discrete and Computational Geometry*, (Special Issue: Proceedings of 22nd Annual ACM Symposium on Computational Geometry), June 2006, Sedona, Arizona. Online available at http://www.stanford.edu/~sergeiv/papers/kMeans-socg.pdf and http://portal.acm.org/citation.cfm?id=1137880 [accessed 2007-08-11].

[2112] Ilpo Vattulainen, T. Ala-Nissila, and K. Kankaala. Physical tests for random numbers in simulations. *Physical Review Letters*, 73(7):2513–2516, 1994. Online available at http://citeseer.ist.psu.edu/vattulainen94physical.html [accessed 2007-07-28].

[2113] Gerhard Venter and Jaroslaw Sobieszczanski-Sobieski. Particle swarm optimization. *AIAA Journal*, 41(8):1583–1589, 2003. AIAA 2002–1235. 43rd AIAA/ASME/ASCE/AHS/ASC, Structures, Structural Dynamics, and Materials Conference, April 22-25, 2002, Denver, Colorado. Online available at http://citeseer.ist.psu.edu/venter02particle.html [accessed 2007-08-20].

[2114] Sébastian Verel, Philippe Collard, and Manuel Clergue. Measuring the evolvability landscape to study neutrality. In *GECCO'06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 613–614, 2006. In proceedings [352]. Online available at http://portal.acm.org/citation.cfm?id=1144107 and http://arxiv.org/abs/0709.4011 [accessed 2008-02-27].

[2115] *6th Metaheuristics International Conference (MIC2005)*, August 22–26, 2005, Vienna, Austria. See http://www.mic2005.org/ [accessed 2007-09-12].

[2116] Frank J. Villegas, Tom Cwik, Yahya Rahmat-Samii, and Majid Manteghi. A parallel electromagnetic genetic-algorithm optimization (ego) application for patch antenna design. *IEEE Transactions on Antennas and Propagation*, 52(9):2424–2435, September 3, 2004. ISSN: 0018-926X. INSPEC Accession Number: 8091160. doi:10.1109/TAP.2004.834071.

[2117] Staal A. Vinterbo and Lucila Ohno-Machado. A genetic algorithm approach to multidisorder diagnosis. *Artificial Intelligence in Medicine*, 18(2):117–132, 2000. Online available at http://dx.doi.org/10.1016/S0933-3657(99)00036-6 [accessed 2007-09-05].

[2118] Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberger, and Hans-Paul Schwefel, editors. *Parallel Problem Solving from Nature – PPSN IV, International Conference on Evolutionary Computation. Proceedings of the 4th International Conference on Parallel Problem Solving from Nature*, volume 1141/1996 of *Lecture Notes in Computer Science (LNCS)*, September 22–24, 1996, Berlin, Germany. Springer. ISBN: 3-5406-1723-X. doi:10.1007/3-540-61723-X. See http://ls11-www.informatik.uni-dortmund.de/PPSN/ppsn4/ppsn4.html [accessed 2007-09-05].

[2119] Paul T. von Hippel. Mean, median, and skew: Correcting a textbook rule. *Journal of Statistics Education*, 13(2), 2005. Online available at http://www.amstat.org/publications/jse/v13n2/vonhippel.html [accessed 2007-09-15].

[2120] Richard von Mises. *Probability, Statistics, and Truth*. Dover Publications, second english, reproduced 1981 dover edition, 1957. ISBN: 0-4862-4214-5, 978-0-48624-214-9. Partly online available at http://books.google.de/books?id=wFFK_P8Cpk0C [accessed 2008-08-19].

[2121] Richard von Mises and Hilda Geiringer. *Mathematical Theory of Probability and Statistics*. Academic Press Inc., New York, February 1965. ISBN: 0-1272-7356-5, 978-0-12727-356-3.

[2122] Michael D. Vose. Generalizing the notion of schema in genetic algorithms. *Artificial Intelligence*, 50(3):385–396, 1991. ISSN: 0004-3702. doi:10.1016/0004-3702(91)90019-G.

[2123] Michael D. Vose and Alden H. Wright. Form invariance and implicit parallelism. *Evolutionary Computation*, 9(3):355–370, 2001. ISSN: 1063-6560. doi:10.1162/106365601750406037. Online available at http://citeseer.ist.psu.edu/610097.html [accessed 2007-07-29].

[2124] Stefan Voss, Silvano Martello, Ibrahim H. Osman, and Cathérine Roucairol, editors. *2nd Metaheuristics International Conference (MIC'97) – Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, July 21–24, 1997, Sophia Antipolis, France. Kluwer Academic Publishers, Boston, USA. ISBN: 0-7923-8369-9. Published in 1998.

[2125] Martin Šimka. Testing true random number generators used in cryptography. In *Proceedings of the IV. PhD students conference*, pages 95–96. Technical University of Košice, Slovakia, May 2004. Online available at http://www.best.tuke.sk/simka/pub.html [accessed 2007-09-15].

[2126] Martin Šimka, Miloš Drutarovský, and Viktor Fischer. Embedded true random number generator in actel fpgas. In *Workshop on Cryptographic Advances in Secure Hardware – CRASH 2005*, pages 6–7, September 2005. Leuven, Belgium. Online available at http://www.best.tuke.sk/simka/pub.html [accessed 2007-09-15].

W

[2127] Maria Wächtler. Entwicklung eines programmes zur auswertung von ^{51}v-festkörper-nmr-mas-spektren mit hilfe genetischer algorithmen am beispiel von spektren von modellkomplexen für vanadiumhaltige haloperoxidasen. Master's thesis, Friedrich-Schiller-Universität Jena, Chemisch-geowissenschaftliche Fakultät, Institut für Physikalische Chemie, 2007. Supervisors: Prof. Dr. Gerd Buntkowsky and Prof. Dr. Jürgen Popp.

[2128] Conrad Hal Waddington. Canalization of development and the inheritance of acquired characters. *Nature*, 150(3811):563–565, November 14, 1942. doi:10.1038/150563a0. Online available at http://www.nature.com/nature/journal/v150/n3811/pdf/150563a0.pdf [accessed 2008-09-10]. See also [2131].

[2129] Conrad Hal Waddington. Genetic assimilation of an acquired character. *Evolution*, 7(2):118–128, June 1953. ISSN: 0014-3820.

[2130] Conrad Hal Waddington. The "baldwin effect", "genetic assimilation" and "homeostasis". *Evolution*, 7(4):386–387, 1953. ISSN: 0014-3820.

[2131] Conrad Hal Waddington. Canalization of development and the inheritance of acquired characters. In *Adaptive individuals in evolving populations: models and algorithms*, pages 91–97. Addison-Wesley Longman Publishing Co., Inc., 1996. In collection [171]. See also [2128].

[2132] Andreas Wagner. *Robustness and Evolvability in Living Systems*. Princeton Studies in Complexity. Princeton University Press, August 2005. ISBN: 0-6911-2240-7.

[2133] Andreas Wagner. Robustness, evolvability, and neutrality. *FEBS Letters*, 579(8): 1772–1778, March 21, 2005. ISSN: 0014-5793. doi:10.1016/j.febslet.2005.01.063. Online available at http://www.citeulike.org/user/timflutre/article/297788 and http://dx.doi.org/10.1016/j.febslet.2005.01.063 [accessed 2007-08-05].

[2134] Günter P. Wagner and Lee Altenberg. Complex adaptations and the evolution of evolvability. *Evolution*, 50(3):967–976, June 1996. ISSN: 00143820. Online available at http://citeseer.ist.psu.edu/437225.html and http://dynamics.org/Altenberg/PAPERS/CAEE/ [accessed 2007-08-05].

[2135] James Alfred Walker and Julian Francis Miller. Evolution and acquisition of modules in cartesian genetic programming. In *7th European Conference on Genetic Programming, EuroGP 2004*, pages 187–197, 2004. In proceedings [1115]. Online available at http://www.cartesiangp.co.uk/papers/2004/wmeurogp2004.pdf and http://www.elec.york.ac.uk/intsys/users/jfm7/eurogp2004.pdf [accessed 2007-11-03].

[2136] James Alfred Walker and Julian Francis Miller. Improving the evolvability of digital multipliers using embedded cartesian genetic programming and product reduction. In Juan Manuel Moreno, Jordi Madrenas, and Jordi Cosp, editors, *Evolvable Systems: From Biology to Hardware, Proceedings of 6th International Conference in Evolvable Systems (ICES 2005)*, volume 3637 of *Lecture Notes in Computer Science (LNCS)*, pages 131–142. Springer, September 12-14, 2005, Sitges, Spain. ISBN: 3-5402-8736-1. Online available at http://www.cartesiangp.co.uk/papers/2005/wmices2005.pdf [accessed 2007-11-03].

[2137] James Alfred Walker and Julian Francis Miller. Investigating the performance of module acquisition in cartesian genetic programming. In *GECCO'05: Proceedings of the 2005 Conference on Genetic and Evolutionary Computation*, pages 1649–1656, 2005. doi:10.1145/1068009.1068287. In proceedings [202]. Online available at http://doi.acm.org/10.1145/1068009.1068287 and http://www.cartesiangp.co.uk/papers/2005/wmgecco2005.pdf [accessed 2007-11-03].

[2138] James Alfred Walker and Julian Francis Miller. Embeddeded cartesian genetic programming and the lawnmower and hierarchical-if-and-only-if problems. In *GECCO'06: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, pages 911–918, 2006. In proceedings [352]. Online available at http://www.cartesiangp.co.uk/papers/2006/wmgecco2006.pdf [accessed 2007-11-03].

[2139] James Alfred Walker and Julian Francis Miller. Predicting prime numbers using cartesian genetic programming. In *Proceedings of the 10th European Conference on Genetic Programming*, pages 205–216, 2007. doi:10.1007/978-3-540-71605-1_19. In proceedings [617].

[2140] Matthew Wall. Galib: A c++ library of genetic algorithm components. version 2.4, documentation revision b. Technical Report, Mechanical Engineering Department, Massachusetts Institute of Technology, August 1996. Online available at http://lancet.mit.edu/ga/dist/galibdoc.pdf [accessed 2007-08-22].

[2141] David A. R. Wallace. *Groups, Rings and Fields*. Springer Undergraduate Mathematics Series. Springer, February 2004. ISBN: 978-3-54076-177-8. Online available at http://books.google.de/books?id=EmO9ejuMHNUC [accessed 2007-07-28].

[2142] Frederick H. Walters, Stephen L. Morgan, Lloyd R. Parker, Jr., and Stanley N. Deming. *Sequential Simplex Optimization: A Technique for Improving Quality and Productivity in Research, Development, and Manufacturing*. Chemometrics Series. CRC Press LLC, 2000 Corporate Blvd., N.W., Boca Raton, Florida 33431, USA, August 26, 1991. ISBN: 0-8493-5894-9, 978-0-84935-894-4. Licensed electronical reprint by MultiSimplex AB, Sternbergsgr. 9, S-371 32 Karlskrona, Sweden. Online available at http://www.chem.sc.edu/faculty/morgan/pubs/WaltersParkerMorganDeming_SequentialSimplexOptimization.pdf [accessed 2008-06-14].

[2143] David L. Waltz, editor. *Proceedings of the National Conference on Artificial Intelligence, AAAI*, August 18–20, 1982, Pittsburgh, PA, USA. AAAI Press. ISBN: 0-2625-1051-2. See http://www.aaai.org/Conferences/AAAI/aaai82.php [accessed 2007-09-06].

[2144] Fang Wang and Yuhui Qiu. Empirical study of hybrid particle swarm optimizers with the simplex method operator. In *ISDA'05: Proceedings of the 5th International Conference on Intelligent Systems Design and Applications*, pages 308–313, September 8–10, 2005, Wroclaw, Poland. IEEE Computer Society, Washington, DC, USA. ISBN: 0-7695-2286-6. doi:10.1109/ISDA.2005.44.

[2145] Fang Wang and Yuhui Qiu. Multimodal function optimizing by a new hybrid nonlinear simplex search and particle swarm algorithm. In João Gama, Rui Camacho, Pavel Brazdil, Alípio Jorge, and Luís Torgo, editors, *Proceedings of ECML 2005, the 16th European Conference on Machine Learning*, volume 3720 of *Lecture Nodes in Artificial Intelligence (LNAI), subseries of Lecture Notes in Computer Science (LNCS)*, pages 759–766. Springer, October 3–7 2005, Porto, Portugal. ISBN: 3-5402-9243-8. doi:10.1007/11564096_78.

[2146] Fang Wang, Yuhui Qiu, and Yun Bai. A new hybrid nm method and particle swarm algorithm for multimodal function optimization. In A. Fazel Famili, Joost N. Kok, José María Peña, Arno Siebes, and A. J. Feelders, editors, *Advances in Intelligent Data Analysis VI, 6th International Symposium on Intelligent Data Analysis, IDA 2005*, volume 3646 of *Lecture Notes in Computer Science (LNCS)*, pages

497–508. Springer, September 8–10, 2005, Madrid, Spain. ISBN: 3-5402-8795-7. doi:10.1007/11552253_45.

[2147] Fang Wang, Yuhui Qiu, and Naiqin Feng. Multi-model function optimization by a new hybrid nonlinear simplex search and particle swarm algorithm. In *First International Conference on Advances in Natural Computation, Part III*, pages 562–565, 2005. doi:10.1007/11539902_68. In proceedings [2153].

[2148] Feng-Sheng Wang and Houng-Jhy Jang. Parameter estimation of a bioreaction model by hybrid differential evolution. In *Proceedings of the 2000 Congress on Evolutionary Computation*, pages 410–417, 2000. In proceedings [1002].

[2149] Lei Wang and F. G. Yuan. Energy harvesting by magnetostrictive material (msm) for powering wireless sensors in shm. In *SPIE Smart Structures and Materials & NDE and Health Monitoring, 14th International Symposium (SSN07), 2007 SPIE/ASME Best Student Paper Presentation Contest*, March 18–22, 2007. Online available at http://www.mae.ncsu.edu/research/SSML/paper.html and http://adsabs.harvard.edu/abs/2007SPIE.6529E.121W [accessed 2007-08-01].

[2150] Lipo Wang, editor. *Support Vector Machines: Theory and Applications*, volume 177 of *Studies in Fuzziness and Soft Computing*. Springer, August 2005. ISBN: 978-3-54024-388-5. doi:10.1007/b95439.

[2151] Lipo Wang, Ke Chen, and Yew-Soon Ong, editors. *Proceedings of the First International Conference on Advances in Natural Computation, ICNC 2005, Part I*, volume 3610 of *Lecture Notes in Computer Science (LNCS)*, August 27–29, 2005, Changsha, China. Springer Berlin / Heidelberg. ISBN: 3-5402-8323-4, 978-3-54028-323-2. doi:10.1007/11539087. See also [2152, 2153].

[2152] Lipo Wang, Ke Chen, and Yew-Soon Ong, editors. *Proceedings of the First International Conference on Advances in Natural Computation, ICNC 2005, Part II*, volume 3611 of *Lecture Notes in Computer Science (LNCS)*, August 27–29, 2005, Changsha, China. Springer Berlin / Heidelberg. ISBN: 3-5402-8325-0, 978-3-54028-325-6. doi:10.1007/11539117. See also [2151, 2153].

[2153] Lipo Wang, Ke Chen, and Yew-Soon Ong, editors. *Proceedings of the First International Conference on Advances in Natural Computation, ICNC 2005, Part III*, volume 3612 of *Lecture Notes in Computer Science (LNCS)*, August 27–29, 2005, Changsha, China. Springer Berlin / Heidelberg. ISBN: 3-5402-8320-X, 978-3-54028-320-1. doi:10.1007/11539902. See also [2151, 2152].

[2154] Paul P. Wang, editor. *Proceedings of the Fifth Joint Conference on Information Science (JCIS 2000), Section: The Third International Workshop on Frontiers in Evolutionary Algorithms (FEA 2000)*, February 27–March 3, 2000, Atlantic City, NJ, USA. The Association for Intelligent Machinery. Workshop held in conjunction with Fifth Joint Conference on Information Sciences.

[2155] Zhao-Xia Wang, Zeng-Qiang Chen, and Zhu-Zhi Yuan. Qos routing optimization strategy using genetic algorithm in optical fiber communication networks. *Journal of Computer Science and Technology*, 19(2):213–217, March 2004. ISSN: 1000-9000. CODEN: JCTEEM.

[2156] C. S. Warnekar and G. Krishna. A heuristic clustering algorithm using union of overlapping pattern-cells. *Pattern Recognition*, 11(2):85–93, 1979. Online available at http://dx.doi.org/10.1016/0031-3203(79)90054-2 [accessed 2007-08-11]. (Link not viable on 2007-08-27).

[2157] *Eighth International Workshop on Learning Classifier Systems (IWLCS-2005)*, June 25–29, 2005, Washington DC, USA. Held during GECCO-2005 (see also [202, 199]). See also [1181].

[2158] Keigo Watanabe and M. M. A. Hashem. *Evolutionary Computations – New Algorithms and their Applications to Evolutionary Robots*. Studies in Fuzziness and Soft Computing. Springer, Berlin, Germany, March 2004. ISBN: 3-5402-0901-8,

978-3-54020-901-0. Partly online available at http://books.google.de/books?id=G5HPNrJNTKsC [accessed 2008-04-03].

[2159] Satoshi Watanabe. *Knowing and Guessing: A Quantitative Study of Inference and Information*. John Wiley & Sons, New York, USA, 1969. ISBN: 0-4719-2130-0, 978-0-47192-130-1.

[2160] Shinya Watanabe, Tomoyuki Hiroyasu, and Mitsunori Miki. Ncga: Neighborhood cultivation genetic algorithm for multi-objective optimization problems. In *GECCO Late Breaking Papers; Late Breaking papers at the Genetic and Evolutionary Computation Conference (GECCO-2002)*, pages 458–465, 2002. In proceedings [331]. Online available at http://citeseer.ist.psu.edu/595343.html and http://www.lania.mx/~ccoello/EMOO/watanabe02.pdf.gz [accessed 2007-07-29].

[2161] James D. Watson, Tania A. Baker, Stephen P. Bell, Alexander Gann, Michael Levine, and Richard Losick. *Molecular Biology of the Gene*. Benjamin Cummings, fifth (december 3, 2003) edition, 1987. ISBN: 978-0-80534-635-0.

[2162] David A. Watt. An extended attribute grammar for pascal. *ACM SIGPLAN Notices*, 14(2):60–74, 1979. ISSN: 0362-1340. doi:10.1145/954063.954071. Online available at http://doi.acm.org/10.1145/954063.954071 [accessed 2007-09-15].

[2163] Bruce H. Weber and David J. Depew, editors. *Evolution and Learning: The Baldwin Effect Reconsidered*. Bradford Books. MIT Press, July 2003. ISBN: 0-2622-3229-4, 978-0-26223-229-6. Partly online available at http://books.google.de/books?id=yBtRzBilw1MC [accessed 2008-09-10].

[2164] David C. Wedge and Douglas B. Kell. Rapid prediction of optimum population size in genetic programming using a novel genotype-fitness correlation. In *Genetic and Evolutionary Computation Conference*, 2008. In proceedings [1117].

[2165] Bill Wedley, editor. *Proceedings of the 17th International Conference on Multiple Criteria Decision Making (MCDM'2004)*, August 6–9, 2004, Whistler Conference Center, Whistler, British Columbia, Canada. See http://www.bus.sfu.ca/events/mcdm/ [accessed 2007-09-10].

[2166] Ingo Wegener and Randall Pruim (Translator). *Complexity Theory: Exploring the Limits of Efficient Algorithms*. Springer-Verlag, Berlin Heidelberg, 2005. ISBN: 978-3-54021-045-0. Translated from the German "Komplexitätstheorie – Grenzen der Effizienz von Algorithmen", (Springer-Verlag 2003, ISBN: 3-540-00161-1).

[2167] Karsten Weicker. *Evolutionäre Algorithmen*. Leitfäden der Informatik. B. G. Teubner GmbH, Stuttgart/Leipzig/Wiesbaden, March 2002. ISBN: 3-5190-0362-7.

[2168] Karsten Weicker and Nicole Weicker. Burden and benefits of redundancy. In *Sixth Workshop on Foundations of Genetic Algorithms (FOGA)*, pages 313–333. Morgan Kaufmann, 2000. In proceedings [1927]. Online available at http://www.imn.htwk-leipzig.de/~weicker/publications/redundancy.pdf [accessed 2008-11-02].

[2169] Edward D. Weinberger. Correlated and uncorrelatated fitness landscapes and how to tell the difference. *Biological Cybernetics*, 63:325–336, 1990.

[2170] Edward D. Weinberger. Local properties of kauffman's nk model, a tuneably rugged energy landscape. *Physical Review A*, 44(10):6399–6413, 1991.

[2171] Edward D. Weinberger. $\mathcal{NP}$ completeness of kauffman's n-k model, a tuneable rugged fitness landscape. Working Papers 96-02-003, Santa Fe Institute, February 1996. Online available at http://www.santafe.edu/research/publications/workingpapers/96-02-003.ps [accessed 2007-08-24].

[2172] Edward D. Weinberger and Peter F. Stadler. Why some fitness landscapes are fractal. *Journal of Theoretical Biology*, 163(2):255–275., July 21, 1993. doi:10.1006/jtbi.1993.1120. Online available at http://www.tbi.univie.ac.at/papers/Abstracts/93-01-01.ps.gz and http://dx.doi.org/10.1006/jtbi.1993.1120 [accessed 2008-06-01].

[2173] Klaus Weinert and Jörn Mehnen. Discrete nurbs-surface approximation using an evolutionary strategy. Technical Report Sonderforschungsbereich (SFB) 531, De-

partment of Machining Technology, University of Dortmund, Germany, October 2001. Online available at http://citeseer.ist.psu.edu/566038.html [accessed 2007-08-27].

[2174] David Jeremy Weir. *Characterizing Mildly Context-Sensitive Grammar Formalisms*. PhD thesis, Department of Computer and Information Science, University of Pennsylvania, 1988. Supervisor: Aravind K. Joshi, Order Number:AAI8908403.

[2175] Thomas Weise. Genetic Programming for Sensor Networks. Technical Report, University of Kassel, University of Kassel, January 2006. Online available at http://www.it-weise.de/documents/files/W2006DGPFa.pdf [accessed 2009-06-26].

[2176] Thomas Weise and Kurt Geihs. Genetic Programming Techniques for Sensor Networks. In *Proceedings of 5. GI/ITG KuVS Fachgespräch "Drahtlose Sensornetze"*, pages 21–25, July 17–18, 2006, University of Stuttgart, Stuttgart, Germany. Online available at http://www.it-weise.de/documents/files/W2006DGPFb.pdf and http://elib.uni-stuttgart.de/opus/volltexte/2006/2838/ [accessed 2007-11-07].

[2177] Thomas Weise and Kurt Geihs. DGPF – An Adaptable Framework for Distributed Multi-Objective Search Algorithms Applied to the Genetic Programming of Sensor Networks. In *Proceedings of the Second International Conference on Bioinspired Optimization Methods and their Application, BIOMA 2006*, pages 157–166, 2006. In proceedings [669]. Online available at http://www.it-weise.de/documents/files/W2006DGPFc.pdf [accessed 2009-06-26].

[2178] Thomas Weise, Stefan Achler, Martin Göb, Christian Voigtmann, and Michael Zapf. Evolving Classifiers – Evolutionary Algorithms in Data Mining. Kasseler Informatikschriften (KIS) 2007, 4, University of Kassel, University of Kassel, University of Kassel, September 28, 2007, University of Kassel. Persistent Identifier: urn:nbn:de:hebis:34-2007092819260. Online available at http://www.it-weise.de/documents/files/WAGVZ2007DMC.pdf and http://kobra.bibliothek.uni-kassel.de/handle/urn:nbn:de:hebis:34-2007092819260 [accessed 2007-10-04].

[2179] Thomas Weise, Steffen Bleul, and Kurt Geihs. Web Service Composition Systems for the Web Service Challenge – A Detailed Review. Kasseler Informatikschriften (KIS) 2007, 7, University of Kassel, FB16, Distributed Systems Group, Wilhelmshöher Allee 73, 34121 Kassel, Germany, University of Kassel, November 19, 2007. Persistent Identifier: urn:nbn:de:hebis:34-2007111919638. Online available at http://kobra.bibliothek.uni-kassel.de/handle/urn:nbn:de:hebis:34-2007111919638 and http://www.it-weise.de/documents/files/WBG2007WSCb.pdf [accessed 2007-11-20]. See also [225, 226].

[2180] Thomas Weise, Kurt Geihs, and Philipp Andreas Baer. Genetic Programming for Proactive Aggregation Protocols. In *Proceedings of the 8th International Conference on Adaptive and Natural Computing Algorithms ICANNGA'07, Part 1*, pages 167–173, 2007. doi:10.1007/978-3-540-71618-1. In proceedings [173]. Online available at http://www.it-weise.de/documents/files/W2007DGPFb.pdf [accessed 2009-06-26].

[2181] Thomas Weise, Michael Zapf, and Kurt Geihs. Rule-based Genetic Programming. In *Proceedings of BIONETICS 2007, 2nd International Conference on Bio-Inspired Models of Network, Information, and Computing Systems*, 2007. In proceedings [1019]. Online available at http://www.it-weise.de/documents/files/WZG2007RBGP.pdf [accessed 2009-06-26].

[2182] Thomas Weise, Michael Zapf, Mohammad Ullah Khan, and Kurt Geihs. Genetic Programming meets Model-Driven Development. Kasseler Informatikschriften (KIS) 2007, 2, University of Kassel, July 2, 2007. Persistent Identifier: urn:nbn:de:hebis:34-2007070218786. Online available at http://www.it-weise.de/documents/files/WZKG2007DGPFd.pdf and http://kobra.bibliothek.uni-kassel.de/handle/urn:nbn:de:hebis:34-2007070218786 [accessed 2007-09-17].

[2183] Thomas Weise, Michael Zapf, Mohammad Ullah Khan, and Kurt Geihs. Genetic Programming meets Model-Driven Development. In *Proceedings of Seventh International*

Conference on Hybrid Intelligent Systems, 2007. doi:10.1109/HIS.2007.11. In proceedings [1170]. Online available at http://www.it-weise.de/documents/files/WZKG2007DGPFg.pdf [accessed 2009-06-26].

[2184] Thomas Weise, Steffen Bleul, Diana Elena Comes, and Kurt Geihs. Different approaches to semantic web service composition. In Abdelhamid Mellouk, Jun Bi, Guadalupe Ortiz, Dickson K. W. Chiu, and Manuela Popescu, editors, *Proceedings of The Third International Conference on Internet and Web Applications and Services, ICIW 2008*, pages 90–96, June 8–13, 2008, Athens, Greece. IEEE Computer Society Press, Los Alamitos, CA, USA. ISBN: 978-0-76953-163-2. Library of Congress Control Number: 2008922600. Product Number: E3163. BMS Part Number: CFP0816C-CDR. Online available at http://www.it-weise.de/documents/files/WBCG2008ICIW.pdf [accessed 2009-06-26].

[2185] Thomas Weise, Stefan Niemczyk, Hendrik Skubch, Roland Reichle, and Kurt Geihs. A tunable model for multi-objective, epistatic, rugged, and neutral fitness landscapes. In *Proceedings of Genetic and Evolutionary Computation Conference, GECCO 2008*, pages 795–802, 2008. doi:10.1145/1389095.1389252. In proceedings [1117]. Online available at http://www.it-weise.de/documents/files/WNSRG2008GECCO.pdf [accessed 2009-06-26].

[2186] Thomas Weise, Hendrik Skubch, Michael Zapf, and Kurt Geihs. Global Optimization Algorithms and their Application to Distributed Systems. Kasseler Informatikschriften (KIS) 2008, 3, University of Kassel, FB16, Distributed Systems Group, Wilhelmshöher Allee 73, 34121 Kassel, Germany, University of Kassel, October 14, 2008. Persistent Identifier: urn:nbn:de:hebis:34-2008101424484. Online available at https://kobra.bibliothek.uni-kassel.de/handle/urn:nbn:de:hebis:34-2008101424484 and http://www.it-weise.de/documents/files/WSTG2008GOAATATDS.pdf [accessed 2008-10-17].

[2187] Thomas Weise, Michael Zapf, and Kurt Geihs. Evolving proactive aggregation protocols. In *Genetic Programming – Proceedings of the 11th European Conference on Genetic Programming, EuroGP 2008*, pages 254–265, 2008. doi:10.1007/978-3-540-78671-9_22. In proceedings [1579]. Online available at http://www.it-weise.de/documents/files/WZG2008DGPFa.pdf and http://dx.doi.org/10.1007/978-3-540-78671-9_22 [accessed 2009-06-26].

[2188] Thomas Weise, Alexander Podlich, Kai Reinhard, Christian Gorldt, and Kurt Geihs. Evolutionary freight transportation planning. In *EvoTRANSLOG, 3rd European Workshop on Evolutionary Computation in Transportation and Logistics*, pages 768–777, 2009. doi:10.1007/978-3-642-01129-0_87. In proceedings [802]. Nominated for best paper award. Online available at http://www.it-weise.de/documents/files/WPRGG2009EFTP.pdf [accessed 2009-06-26].

[2189] August Weismann. *The Germ-Plasm – A Theory of Heredity*. Charles Scribner's Sons, New York, USA, 1893. Translated by W. Newton Parker, Ph.D. and Harriet Rönnfeld B.Sc. Online available at http://www.esp.org/books/weismann/germ-plasm/facsimile/ [accessed 2008-09-10].

[2190] Gerhard Weiß and Sandip Sen, editors. *Adaption and Learning in Multi-Agent Systems, IJCAI'95 Workshop Proceedings*, volume 1042 of *Lecture Notes in Computer Science (LNCS)*, 1996, Montréal, Québec, Canada. Springer. ISBN: 3-5406-0923-7. See also [1453, 1454, 1364].

[2191] Eric W. Weisstein. K-means clustering algorithm, 1999–2006. From MathWorld–A Wolfram Web Resource. Online available at http://mathworld.wolfram.com/K-MeansClusteringAlgorithm.html [accessed 2007-08-11].

[2192] Justin Werfel and Radhika Nagpal. Extended stigmergy in collective construction. *IEEE Intelligent Systems*, 21(2):20–28, March/April 2006. doi:10.1109/MIS.2006.25. Online available at http://hebb.mit.edu/people/jkwerfel/ieeeis06.pdf [accessed 2008-06-12].

[2193] Justin Werfel, Yaneer Bar-Yam, Daniela Rus, and Radhika Nagpal. Distributed construction by mobile robots with enhanced building blocks. In *IEEE International Conference on Robotics and Automation (ICRA)*, May 15–19, 2006, Hilton in the Walt Disney World Resort hotel, Walt Disney World Resort, Orlando, Florida (Lake Buena Vista), USA. Online available at http://www.eecs.harvard.edu/~rad/ssr/papers/icra06-werfel.pdf [accessed 2008-06-12].

[2194] Gregory M. Werner and Michael G. Dyer. Evolution of communication in artificial organisms. In *Artificial Life II*, pages 659–687, 1992. Redwood City, CA. In proceedings [1248]. Online available at http://www.isrl.uiuc.edu/~amag/langev/paper/werner92evolution0f.html [accessed 2008-07-28].

[2195] Thomas H. Westerdale. The bucket brigade is not genetic. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 45–59, 1985. In proceedings [856].

[2196] Thomas H. Westerdale. A reward scheme for production systems with overlapping conflict sets. *IEEE Transactions on Systems, Man and Cybernetics*, 16(3):369–383, 1986. ISSN: 0018-9472.

[2197] Thomas H. Westerdale. Altruism in the bucket brigade. In *Proceedings of the Second International Conference on Genetic algorithms and their application*, pages 22–26, 1987. In proceedings [857].

[2198] A. Wetzel. *Evaluation of the Effectiveness of Genetic Algorithms in Combinatorial Optimization*. PhD thesis, University of Pittsburgh, Pittsburgh, PA, 1983. Unpublished manuscript, technical report.

[2199] Ingrid Wetzel. Information systems development with anticipation of change: Focussing on professional bureaucracies. In *Proceedings of Hawaii International Conference on Systems Sciences, HICSS 34*. IEEE Computer Society, January 2001, Maui, Hawaii, USA. Online available at http://citeseer.ist.psu.edu/532081.html and http://swt-www.informatik.uni-hamburg.de/publications/download.php?id=177 [accessed 2007-09-02].

[2200] James F. Whidborne, D.-W. Gu, and Ian Postlethwaite. Algorithms for the method of inequalities – a comparative study. In *Procedings of the 1995 American Control Conference*, volume 5, pages 3393–3397, June 21–23, 1995, Seattle, Washington, USA. ISBN: 0-7803-2445-5. INSPEC Accession Number: 5080557. FA19 = 9:15.

[2201] Peter Alexander Whigham. Context-free grammar and genetic programming. Technical Report Technical Report CS20/94, Department of Computer Science, Australian Defence Force Academy, University of New South Wales, Canberra ACT 2600, Australia, 1994.

[2202] Peter Alexander Whigham. Grammatically-based genetic programming. In *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, pages 33–41, 1995. In proceedings [1757]. Online available at http://citeseer.ist.psu.edu/whigham95grammaticallybased.html [accessed 2007-08-15].

[2203] Peter Alexander Whigham. Inductive bias and genetic programming. In *First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, GALESIA*, pages 461–466, 1995. In proceedings [2309]. Online available at http://citeseer.ist.psu.edu/343730.html [accessed 2008-08-15].

[2204] Peter Alexander Whigham. *Grammatical bias for evolutionary learning*. PhD thesis, School of Computer Science, University College, University of New South Wales, Australian Defence Force Academy, Canberra, New South Wales, Australia, October 14, 1996. Order Number: AAI0597571.

[2205] Peter Alexander Whigham. Search bias, language bias, and genetic programming. In *Genetic Programming 1996: Proceedings of the First Annual Conference*, pages 230–237, 1996. In proceedings [1207]. Online available at http://citeseer.ist.psu.edu/whigham96search.html and ftp://www.cs.adfa.edu.au/pub/xin/whigham_gp96.ps.gz [accessed 2007-09-09].

[2206] R. C. White, jr. A survey of random methods for parameter optimization. *Simulation*, 17(5):197–205, 1971. doi:10.1177/003754977101700504. Online available at http://sim.sagepub.com/cgi/reprint/17/5/197?ck=nck [accessed 2008-03-26].

[2207] L. Darell Whitley. A genetic algorithm tutorial. Technical Report CS-93-103, Computer Science Department, Colorado State University, Fort Collins, March 10, 1993. Online available at http://citeseer.ist.psu.edu/177719.html [accessed 2007-11-29]. See also [2208].

[2208] L. Darell Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4(2):65–85, June 1994. ISSN: ISSN 0960-3174 (Print) 1573-1375 (Online). doi:10.1007/BF00175354. Online available at http://samizdat.mines.edu/ga_tutorial/ga_tutorial.ps and http://www.citeulike.org/user/Bc91/article/1449453 [accessed 2007-08-12]. Also published as technical report [2207].

[2209] L. Darrell Whitley, editor. *Proceedings of the Second Workshop on Foundations of Genetic Algorithms (FOGA)*, July 26–29, 1992, Vail, Colorado, USA. Morgan Kaufmann, San Mateo, CA, USA. ISBN: 1-5586-0263-1. Published February 1, 1993.

[2210] L. Darrell Whitley, editor. *Late Breaking Papers at Genetic and Evolutionary Computation Conference (GECCO'00)*, July 8–12, 2000, The Riviera Hotel and Casino, Las Vegas, Nevada, USA. See also [2216].

[2211] L. Darrell Whitley. The GENITOR algorithm and selective pressure: Why rank-based allocation of reproductive trials is best. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 116–121, 1989. In proceedings [1820]. Online available at http://citeseer.ist.psu.edu/531140.html and http://www.cs.colostate.edu/~genitor/1989/ranking89.ps.gz [accessed 2007-08-21].

[2212] L. Darrell Whitley. Cellular genetic algorithms. In *Proceedings of the 5th International Conference on Genetic Algorithms*, page 658, 1993. In proceedings [730].

[2213] L. Darrell Whitley and Timothy Starkweather. Genitor ii.: A distributed genetic algorithm. *Journal of Experimental & Theoretical Artificial Intelligence*, 2(3):189–214, July 1990. ISSN: 0952-813X. doi:10.1080/09528139008953723.

[2214] L. Darrell Whitley and Michael D. Vose, editors. *Proceedings of the Third Workshop on Foundations of Genetic Algorithms (FOGA)*, July 31–August 2, 1994, Estes Park, Colorado, USA. Morgan Kaufmann, San Francisco, CA, USA. ISBN: 1-5586-0356-5. Published June 1, 1995.

[2215] L. Darrell Whitley, V. Scott Gordon, and Keith E. Mathias. Lamarckian evolution, the baldwin effect and function optimization. In *PPSN III: Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature*, pages 6–15, 1994. In proceedings [492]. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.18.2428 [accessed 2008-09-11].

[2216] L. Darrell Whitley, David Goldberg, Erick Cantú-Paz, Lee Spector, Ian C. Parmee, and Hans-Georg Beyer, editors. *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO'00)*, July 8–12, 2000, The Riviera Hotel and Casino, Las Vegas, Nevada, USA. Morgan Kaufmann. ISBN: 978-1-55860-708-8. See also [2210].

[2217] Dirk Wiesmann, Ulrich Hammel, and Thomas Bäck. Robust design of multilayer optical coatings by means of evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 2(4):162–167, November 1998. ISSN: 1089-778X. CODEN: ITEVF5. INSPEC Accession Number: 6149966. doi:10.1109/4235.738986. See also [2218].

[2218] Dirk Wiesmann, Ulrich Hammel, and Thomas Bäck. Robust design of multilayer optical coatings by means of evolutionary strategies. Sonderforschungsbereich (sfb) 531, Universität Dortmund, March 31 1998. Online available at http://hdl.handle.net/2003/5348 and http://citeseer.ist.psu.edu/325801.html [accessed 2008-07-19]. See also [2217].

[2219] Wikipedia. Wikipedia – the free encyclopedia, 2008. Online available at http://en.wikipedia.org/ [accessed 2009-06-26].

[2220] Frank Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1 (6):80–83, December 1945. ISSN: 00994987. Online available at http://sci2s.ugr.es/keel/pdf/algorithm/articulo/wilcoxon1945.pdf [accessed 2008-08-06].

[2221] Frank Wilcoxon and Roberta A. Wilcox. *Some Rapid Approximate Statistical Procedures*. American Cyanamid Company, Stamford Research Laboratories, Stamford, Connecticut, USA, 1964. Original: 1949.

[2222] Frank Wilcoxon, S. V. Katti, and Roberta A. Wilcox. Critical values and probability levels for the wilcoxon rank sum test and the wilcoxon signed rank test. In H. Leon Harter and D. B. Owen, editors, *Selected Tables in Mathematical Statistics*, volume 1, pages 171–259. American Mathematical Society / Markham, Providence, Rode Island, USA, 1975. ISBN: 0-8218-1901-1, 0-8410-2501-0, 978-0-84102-501-1, 978-0-82181-901-2.

[2223] Herbert S. Wilf. *Algorithms and Complexity*. AK Peters, Ltd., second edition, December 2002. ISBN: 978-1-56881-178-9.

[2224] Claus O. Wilke. *Evolutionary Dynamics in Time-Dependent Environments*. PhD thesis, Fakultät für Physik und Astronomie, Ruhr-Universität Bochum, Shaker Verlag, Aachen, July 1999. ISBN: 978-3-82656-199-3. Online available at http://wlab.biosci.utexas.edu/~wilke/ps/PhD.ps.gz [accessed 2007-08-19].

[2225] Claus O. Wilke. Adaptive evolution on neutral networks. *Bulletin of Mathematical Biology*, 63(4):715–730, July 2001. ISSN: 0092-8240 (Print) 1522-9602 (Online). Online available at http://arxiv.org/abs/physics/0101021v1 [accessed 2008-07-02].

[2226] Daniel N. Wilke, Schalk Kok, and Albert A. Groenwold. Comparison of linear and classical velocity update rules in particle swarm optimization: notes on diversity. *International Journal for Numerical Methods in Engineering*, 70(8):962–984, 2007. Online available at http://doi.wiley.com/10.1002/nme.1867 [accessed 2007-08-20].

[2227] George C. Williams. Pleiotropy, natural selection, and the evolution of senescence. *Evolution*, 11(4):398–411, December 1957. doi:10.2307/2406060. See also [2228].

[2228] George C. Williams. Pleiotropy, natural selection, and the evolution of senescence. *SAGE KE, Science of Aging Knowledge Environment*, 2001(1), October 3, 2001. cp13. See also [2227].

[2229] Daniel B. Willingham and Laura Preuss. The death of implicit memory. *Psyche*, 2(15), October 1995. Online available at http://www.journalpsyche.org/ojs-2.2/index.php/psyche/article/viewFile/2419/2348 and http://psyche.cs.monash.edu.au/v2/psyche-2-15-willingham.html [accessed 2008-12-01].

[2230] Dominic Wilson and Devinder Kaur. Using quotient graphs to model neutrality in evolutionary search. In *Genetic and Evolutionary Computation Conference*, pages 2233–2238, 2008. In proceedings [1117].

[2231] Edward Osborne Wilson. *Sociobiology: The New Synthesis*. Belknap Press / Harvard University Press, Cambridge, Massachusets, USA / Cumbreland, Rhode Island, USA, 1975. See also [2232].

[2232] Edward Osborne Wilson. *Sociobiology: The New Synthesis*. Belknap Press, twenty-fifth anniversary edition edition, March 2000. ISBN: 0-6740-0235-0, 978-0-67400-235-7. See also [2231].

[2233] P. B. Wilson and M. D. Macleod. Low implementation cost iir digital filter design using genetic algorithms. In *Proceedings of the IEE/IEEE Workshop on Natural Algorithms in Signal Processing*, pages 4/1–4/8. IEEE, November 14–16, 1993, Chelmsford, Essex, UK.

[2234] Stewart Wilson. Bid competition and specificity reconsidered. *Complex Systems*, 2 (6):705–723, 1988. ISSN: 0891-2513.

[2235] Stewart W. Wilson. ZCS: A zeroth level classifier system. *Evolutionary Computation*, 2(1):1–18, 1994. Online available at http://citeseer.ist.psu.edu/wilson94zcs.html and http://www.eskimo.com/~wilson/ps/zcs.pdf [accessed 2007-09-12].

[2236] Stewart W. Wilson. Classifier fitness based on accuracy. *Evolutionary Computation*, 3(2):149–175, 1995. Online available at http://citeseer.ist.psu.edu/wilson95classifier.html [accessed 2007-09-12].

[2237] Stewart W. Wilson. Generalization in the XCS classifier system. In *Proceedings of the Third Annual Conference on Genetic Programming 1998*, pages 665–674, 1998. In proceedings [1209]. Online available at http://citeseer.ist.psu.edu/wilson98generalization.html [accessed 2007-09-12].

[2238] Stewart W. Wilson. State of XCS classifier system research. In *Learning Classifier Systems, From Foundations to Applications*, pages 63–82. Springer-Verlag, 2000. In collection [1252]. Online available at http://citeseer.ist.psu.edu/72750.html and http://www.eskimo.com/~wilson/ps/state.ps.gz [accessed 2007-08-23].

[2239] Stewart W. Wilson and David E. Goldberg. A critical review of classifier systems. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 244–255, 1989. In proceedings [1820]. Online available at http://www.eskimo.com/~wilson/ps/wg.ps.gz [accessed 2007-09-12].

[2240] Öjvind Winge. Wilhelm johannsen: The creator of the terms gene, genotype, phenotype and pure line. *Journal of Heredity*, 49(2):83–88, March 1958. ISSN: 1465-7333 (Online), 0022-1503 (Print). Online available at http://jhered.oxfordjournals.org/cgi/reprint/49/2/83.pdf [accessed 2008-08-21]. See also [1056].

[2241] Hans Winkler. *Verbreitung und Ursache der Parthenogenesis im Pflanzen- und Tierreiche*. Verlag Gustav Fischer, Jena, 1920.

[2242] Paul C. Winter, G. Ivor Hickey, and Hugh L. Fletcher. *Instant Notes in Genetics*. Springer, New York / BIOS Scientific Publishers / Taylor & Francis Ltd., 1st: 1998, 2nd ed: 2002, 3rd ed: 2006. ISBN: 1-8599-6166-5, 0-3879-1562-1, 978-0-38791-562-3, 1-8599-6262-9, 978-1-85996-262-6, 978-0-41537-619-8, 0-4153-7619-X.

[2243] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann, first edition, October 1999. ISBN: 978-1-55860-552-7.

[2244] David H. Wolpert and William G. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, The Santa Fe Institute, 1399 Hyde Park Rd., Santa Fe, NM, 87501, USA, February 6, 1995. Online available at http://citeseer.ist.psu.edu/wolpert95no.html and http://www.santafe.edu/research/publications/workingpapers/95-02-010.pdf [accessed 2008-03-28].

[2245] David H. Wolpert and William G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, April 1997. doi:10.1109/4235.585893. Online available at http://citeseer.ist.psu.edu/wolpert96no.html [accessed 2008-03-28].

[2246] Koon-Pong Wong, S. R. Meikle, Dagan Feng, and M. J. Fulham. Estimation of input function and kinetic parameters using simulated annealing: application in a flow model. *IEEE Transactions on Nuclear Science*, 49(3):707–713, June 2002. ISSN: 0018-9499. doi:10.1109/TNS.2002.1039552.

[2247] Man Leung Wong and Kwong Sak Leung. Learning first-order relations from noisy databases using genetic algorithms. In *Proceedings of the Second Singapore International Conference on Intelligent Systems*, pages 159–164, 1994. Online available at http://cptra.ln.edu.hk/staffProfile/mlwongPub.htm and http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/ManLeungWong.html [accessed 2007-08-15].

[2248] Man Leung Wong and Kwong Sak Leung. An adaptive inductive logic programming system using genetic programming. In *Evolutionary Programming IV Proceedings of the Fourth Annual Conference on Evolutionary Programming*, pages

737–752, 1995. In proceedings [1380], Online available at http://cptra.ln.edu.hk/staffProfile/mlwongPub.htm and http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/ManLeungWong.html [accessed 2007-08-15].

[2249] Man Leung Wong and Kwong Sak Leung. Inducing logic programs with genetic algorithms: the genetic logicprogramming system genetic logic programming and applications. *IEEE Expert*, 10(5):68–76, October 1995. doi:10.1109/64.464935. IEEE Expert Special Track on Evolutionary Programming (Peter John Angeline ed.). Online available at http://cptra.ln.edu.hk/staffProfile/mlwongPub.htm and http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/ManLeungWong.html [accessed 2007-08-15].

[2250] Man Leung Wong and Kwong Sak Leung. Combining genetic programming and inductive logic programming using logic grammars. In *IEEE Conference on Evolutionary Computation*, volume 2, pages 733–736, 1995. In proceedings [1000]. Online available at http://cptra.ln.edu.hk/staffProfile/mlwongPub.htm and http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/ManLeungWong.html [accessed 2007-08-15].

[2251] Man Leung Wong and Kwong Sak Leung. Applying logic grammars to induce subfunctions in geneticprogramming. In *Proceedings of IEEE International Conference on Evolutionary Computation*, volume 2, pages 737–740, 1995. In proceedings [1000]. Online available at http://cptra.ln.edu.hk/staffProfile/mlwongPub.htm and http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/ManLeungWong.html [accessed 2007-08-15].

[2252] Man Leung Wong and Kwong Sak Leung. Evolutionary program induction directed by logic grammars. *Evolutionary Computation*, 5(2):143–180, summer 1997. Special Issue: Trends in Evolutionary Methods for Program Induction. Online available at http://cptra.ln.edu.hk/staffProfile/mlwongPub.htm and http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/ManLeungWong.html [accessed 2007-08-15].

[2253] Man Leung Wong and Kwong Sak Leung. *Data Mining Using Grammar Based Genetic Programming and Applications*, volume 3 of *Genetic Programming*. Springer, January 2000. ISBN: 978-0-79237-746-7.

[2254] John R. Woodward. Evolving turing complete representations. In *Proceedings of the 2003 Congress on Evolutionary Computation*, volume 2, pages 830–837, 2003. doi:10.1109/CEC.2003.1299753. In proceedings [1803]. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.3.6859 and http://www.cs.bham.ac.uk/~wbl/biblio/gp-html/Woodward_2003_Etcr.html [accessed 2008-11-08].

[2255] Nimit Worakul, Wibul Wongpoowarak, and Prapaporn Boonme. Optimization in development of acetaminophen syrup formulation. *Drug Development and Industrial Pharmacy*, 28(3):345–351, 2002. ISSN: 1520-5762 (electronic) 0363-9045 (paper). see erratum [2256].

[2256] Nimit Worakul, Wibul Wongpoowarak, and Prapaporn Boonme. Optimization in development of acetaminophen syrup formulation – erratum. *Drug Development and Industrial Pharmacy*, 28(8):1043–10045, 2002. ISSN: 1520-5762 (electronic) 0363-9045 (paper). see paper [2255].

[2257] Robert P. Worden. A speed limit for evolution. *Journal of Theoretical Biology*, 176 (1):137–152, September 7, 1995. ISSN: 0022-5193. doi:10.1006/jtbi.1995.0183. Online available at http://dx.doi.org/10.1006/jtbi.1995.0183 [accessed 2008-08-10].

[2258] Alden H. Wright, Richard K. Thompson, and Jian Zhang. The computational complexity of n-k fitness functions. *IEEE Transactions on Evolutionary Computation*, 4(4):373–379, November 2000. ISSN: 1089-778X. INSPEC Accession Number: 6791340. doi:10.1109/4235.887236. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.40.3093 and http://www.cs.umt.edu/u/wright/papers/nkcomplexity.ps.gz [accessed 2009-02-26].

[2259] Alden H. Wright, Michael D. Vose, Kenneth Alan De Jong, and Lothar M. Schmitt, editors. *Revised Selected Papers of the 8th International Workshop on Foundations of Genetic Algorithms, FOGA 2005*, volume 3469/2005 of *Lecture Notes in Computer*

Science (LNCS), January 5–9, 2005, Aizu-Wakamatsu City, Japan. Springer Verlag, Berlin Heidelberg. ISBN: 978-3-54027-237-3. doi:10.1007/11513575_1. Published August 22, 2005. see http://www.cs.umt.edu/foga05/ [accessed 2007-09-01].

[2260] Margaret H. Wright. Direct search methods: Once scorned, now respectable. In D. F. Griffiths and G. A. Watson, editors, *Numerical Analysis – Proceedings of the 16th Dundee Biennial Conference in Numerical Analysis*, Pitman Research Notes in Mathematics, pages 191–208, June 27–30, 1995, University of Dundee, Scotland, UK. Addison Wesley Longman Limited / Chapman & Hall/CRC, Harlow, UK. ISBN: 0-5822-7633-0, 978-0-58227-633-8. Online available at http://citeseer.ist.psu.edu/155516.html [accessed 2008-06-14].

[2261] Sewall Wright. The roles of mutation, inbreeding, crossbreeding and selection in evolution. In D. F. Jones, editor, *Proceedings of the Sixth Annual Congress of Genetics*, volume 1, pages 356–366, 1932. Online available at http://www.blackwellpublishing.com/ridley/classictexts/wright.pdf [accessed 2007-08-11].

[2262] Chun-Hsin Wu. Peer-to-peer systems: Macro-computing with micro-computers, July 2003. Presented at 3rd International Conference on Open Source in Taipei, Taiwan. Presentation available at http://www.csie.nuk.edu.tw/~wuch/publications/2003-icos-p2p-wuch.pdf [accessed 2007-08-13].

[2263] Ge Wu, Volkert Hansen, E. Kreysa, and H.-P. Gemünd. Optimierung von fss-bandpassfiltern mit hilfe der schwarmintelligenz (particle swarm optimization). *Advances in Radio Science*, 4:65–71, September 2006. Online available at http://www.adv-radio-sci.net/4/65/2006/ars-4-65-2006.pdf [accessed 2007-08-21].

[2264] Hsien-Chung Wu. *Evolutionary Computation*. Department of Mathematics, National Kaohsiung Normal University, Kaohsiung 802, Taiwan, February 2005. Lecture notes. Online available at http://nknucc.nknu.edu.tw/~hcwu/pdf/evolec.pdf [accessed 2007-07-16].

[2265] Mingfang Wu, Michael Fuller, and Ross Wilkinson. Using clustering and classification approaches in interactive retrieval. *Inf. Process. Manage.*, 37(3):459–484, 2001. Online available at http://citeseer.ist.psu.edu/wu01using.html and http://de.scientificcommons.org/313591 [accessed 2007-08-11].

[2266] Nelson Wu. Differential evolution for optimisation in dynamic environments. Technical Report, School of Computer Science and Information Technology, RMIT University, November 2006. Online available at http://yallara.cs.rmit.edu.au/~newu/portfolio.html [accessed 2007-08-19].

X

[2267] Fatos Xhafa, Francisco Herrera, Ajith Abraham abd Mario Köppen, and Jose Manuel Bénitez, editors. *8th International Conference on Hybrid Intelligent Systems (HIS 2008)*, September 10–12, 2008, Edifici Vèrtex, Plaça Eusebi Güell 6, Technical University of Catalonia, Barcelona, Spain. IEEE Computer Society, IEEE Service, Center, 445 Hoes Lane, P.O. Box 133, Piscataway, NJ 08855-1331, USA. ISBN: 978-0-76953-326-1. Library of Congress Control Number: 2008928438. IEEE Computer Society Order Number P3326. BMS Part Number CFP08360-CDR. see http://his2008.lsi.upc.edu/ [accessed 2009-03-02].

[2268] Bowei Xi, Zhen Liu, Mukund Raghavachari, Cathy H. Xia, and Li Zhang. A smart hill-climbing algorithm for application server configuration. In *WWW'04: Proceedings of the 13th international conference on World Wide Web*, pages 287–296, May 17–20, 2004, New York, NY, USA. ACM Press, New York, NY, USA. ISBN: 1-5811-3844-X. doi:10.1145/988672.988711. Session: Server performance and scalability. Online available at http://citeseer.ist.psu.edu/xi04smart.html [accessed 2007-09-11].

[2269] Yong L. Xiao and Donald E. Williams. Game: Genetic algorithm for minimization of energy, an interactive program for three-dimensional intermolecular interactions. *Computers & Chemistry*, 18(2):199–201, 1994.

[2270] Shengwu Xiong, Weiwu Wang, and Feng Li. A new genetic programming approach in symbolic regression. In *Proceedings of 15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'03)*, pages 161–167, November 3–5, 2003. IEEE Computer Society, Los Alamitos, CA, USA. doi:10.1109/TAI.2003.1250185.

[2271] Kai Xu, Sushil J. Louis, and Roberto C. Mancini. A scalable parallel genetic algorithm for x-ray spectroscopic analysis. In *GECCO'05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 811–816, 2005. doi:http://doi.acm.org/10.1145/1068009.1068145. In proceedings [202]. Online available at `http://doi.acm.org/10.1145/1068009.1068145` [accessed 2007-08-14].

Y

[2272] Hirozumi Yamaguchi, Kozo Okano, Teruo Higashino, and Kenichi Taniguchi. Synthesis of protocol entities' specifications from service specifications in a petri net model with registers. In *ICDCS'95: Proceedings of the 15th International Conference on Distributed Computing Systems*, pages 510–517, May 30–June 2, 1995, Vancouver, British Columbia, Canada. IEEE Computer Society, Washington, DC, USA. ISBN: 0-8186-7025-8. Online available at `http://citeseer.ist.psu.edu/yamaguchi95synthesis.html` [accessed 2008-06-15].

[2273] Lidia A. R. Yamamoto and Christian F. Tschudin. Experiments on the automatic evolution of protocols using genetic programming. In *Autonomic Communication – Revised Selected Papers from the Second International IFIP Workshop, WAC 2005*, volume 3854/2006 of *Lecture Notes in Computer Science (LNCS)*, pages 13–28. Springer Berlin / Heidelberg, October 2–5 2005, Vouliagmeni, Athens, Greece. ISBN: 978-3-54032-992-3. doi:10.1007/11687818_2. Online available at `http://cn.cs.unibas.ch/people/ly/doc/wac2005-lyct.pdf` [accessed 2008-06-20]. See also [2274].

[2274] Lidia A. R. Yamamoto and Christian F. Tschudin. Experiments on the automatic evolution of protocols using genetic programming. Technical Report CS-2005-002, University of Basel, April 21, 2005. Online available at `http://cn.cs.unibas.ch/people/ly/doc/wac2005tr-lyct.pdf` [accessed 2008-06-20]. See also [2273].

[2275] Lidia A. R. Yamamoto and Christian F. Tschudin. Genetic evolution of protocol implementations and configurations. In *IFIP/IEEE International workshop on Self-Managed Systems and Services (SelfMan 2005)*, May 19, 2005, Nice, France. Online available at `http://cn.cs.unibas.ch/pub/doc/2005-selfman.pdf` [accessed 2007-09-17].

[2276] Lidia A. R. Yamamoto, Daniel Schreckling, and Thomas Meyer. Self-replicating and self-modifying programs in fraglets. In *Proceedings of BIONETICS 2007, 2nd International Conference on Bio-Inspired Models of Network, Information, and Computing Systems*, 2007. In proceedings [1019]. Online available at `http://cn.cs.unibas.ch/people/ly/doc/bionetics2007-ysm.pdf` [accessed 2008-05-04].

[2277] Koetsu Yamazaki, Sourav Kundu, and Michitomo Hamano. Genetic programming based learning of control rules for variable geometry structures. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, pages 412–415, 1998. In proceedings [1209]. Online available at `http://citeseer.ist.psu.edu/kundu98genetic.html` [accessed 2007-09-09].

[2278] Hongmei Yan, Yingtao Jiang, Jun Zheng, Chenglin Peng, and Shouzhong Xiao. Discovering critical diagnostic features for heart diseases with a hybrid genetic algorithm. In Faramarz Valafar and Homayoun Valafar, editors, *Proceedings of the International Conference on Mathematics and Engineering Techniques in Medicine and Biological Scienes, METMBS'03*, pages 406–409. CSREA Press, June 2003, Las Vegas, Nevada, USA. ISBN: 1-9324-1504-1.

[2279] Ang Yang, Yin Shan, and Lam Thu Bui, editors. *Success in Evolutionary Computation*, volume 92/2008 of *Studies in Computational Intelligence*. Springer Berlin / Heidelberg, January 16, 2008. ISBN: 3-5407-6285-X, 978-3-54076-285-0, 978-3-54076-286-7. Library of Congress Control Number: 2007939404. doi:10.1007/978-3-540-76286-7.

[2280] Shengxiang Yang, Yew-Soon Ong, and Yaochu Jin, editors. *Evolutionary Computation in Dynamic and Uncertain Environments*, volume 51(XXIII) of *Studies in Computational Intelligence*. Springer, 2007. ISBN: 978-3-54049-772-1. Presentation online available at http://www.soft-computing.de/Jin_CEC04T.pdf.gz [accessed 2007-08-19].

[2281] Ziheng Yang and Joseph P. Bielawski. Statistical methods for detecting molecular adaptation. *Trends in Ecology & Evolution*, 15(12):496–503, December 1, 2000. ISSN: 0169-5347. doi:10.1016/S0169-5347(00)01994-7. Online available at http://dx.doi.org/10.1016/S0169-5347(00)01994-7 and http://citeseer.ist.psu.edu/yang00statistical.html [accessed 2008-07-20].

[2282] Xin Yao, editor. *Progress in Evolutionary Computation, Selected Papers of the AI'93 and AI'94 Workshops on Evolutionary Computation Melbourne, Victoria, Australia, November 16, 1993 and Armidale, NSW, Australia, November 21-22, 1994*, volume 956/1995 of *Lecture Notes in Artificial Intelligence, subseries of Lecture Notes in Computer Science (LNCS)*, May 1995. Springer, Berlin/Heidelberg, Germany. ISBN: 3-5406-0154-6, 978-3-54060-154-8. doi:10.1007/3-540-60154-6.

[2283] Xin Yao. Optimization by genetic annealing. In M. Jabri, editor, *Proceedings of Second Australian Conference on Neural Networks*, pages 94–97, 1991, Sydney, Australia. Online available at http://citeseer.ist.psu.edu/yao91optimization.html and ftp://www.cs.adfa.edu.au/pub/xin/acnn91.ps.Z [accessed 2007-09-10].

[2284] Xin Yao, editor. *Evolutionary Computation: Theory and Applications*. World Scientific Publishing Co Pte Ltd, January 28, 1996. ISBN: 9-8102-2306-4, 978-9-81022-306-9. Partly online available at http://books.google.de/books?id=GP7ChxbJOE4C [accessed 2008-03-25].

[2285] Xin Yao, Edmund K. Burke, José Antonio Lozano, Jim Smith, Juan J. Merelo Guervós, John A. Bullinaria, Jonathan E. Rowe, Peter Tiňo, Ata Kabán, and Hans-Paul Schwefel, editors. *Proceedings of the 8th International Conference on Parallel Problem Solving from Nature – PPSN VIII*, volume 3242/2004 of *Lecture Notes in Computer Science (LNCS)*, September 18–22, 2004, Birmingham, UK. Springer. ISBN: 3-5402-3092-0, 978-3-54023-092-2. doi:10.1007/b100601. See http://events.cs.bham.ac.uk/ppsn04/ [accessed 2007-09-05].

[2286] Xin Yao, F. Wang, K. Padmanabhan, and Sancho Salcedo-Sanz. Hybrid evolutionary approaches to terminal assignment in communications networks. In *Recent Advances in Memetic Algorithms*, pages 129–159. Springer, 2005. doi:10.1007/3-540-32363-5_7. In collection [901].

[2287] Maqsood Yaqub, Ronald Boellaard, Marc A Kropholler, and Adriaan A Lammertsma. Optimization algorithms and weighting factors for analysis of dynamic pet studies. *Physics in Medicine and Biology*, 51:4217–4232, August 2006. doi:10.1088/0031-9155/51/17/007. Online available at stacks.iop.org/PMB/51/4217 [accessed 2007-08-25].

[2288] Frank Yates. *The Design and Analysis of Factorial Experiments*. Imperial Bureau of Soil Science, Commonwealth Agricultural Bureaux, Harpenden, England, UK, 1937. ISBN: 0-8519-8220-4, 978-0-85198-220-5. ASIN: B00086SIZ0. Technical Communication No. 35.

[2289] Tao Ye. *Large-scale network parameter configuration using on-line simulation framework*. PhD thesis, Department of Electrical, Computer and System Engineering, Rensselaer Polytechnic Institute, Troy, New York 12180, USA, March 2003. Adviser: Shivkumar Kalyanaraman. Order-No. AAI3088530. Online

available at http://www.ecse.rpi.edu/Homepages/shivkuma/research/papers/tao-phd-thesis.pdf [accessed 2008-10-16].

[2290] Tao Ye and Shivkumar Kalyanaraman. An adaptive random search alogrithm for optimizing network protocol parameters. Technical Report, Department of Electrical, Computer and System Engineering, Rensselaer Polytechnic Institute, Troy, New York 12180, USA, 2001. Online available at http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.24.8138 and http://www.ecse.rpi.edu/Homepages/shivkuma/research/papers/tao-icnp2001.pdf [accessed 2008-10-16].

[2291] Gary G. Yen, Simon M. Lucas, Gary Fogel, Graham Kendall, Ralf Salomon, Byoung-Tak Zhang, Carlos A. Coello Coello, and Thomas Philip Runarsson, editors. *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2006*, July 16–21, 2006, Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada. IEEE Press, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA. ISBN: 0-7803-9487-9.

[2292] John Yen and Bogju Lee. A simplex genetic algorithm hybrid. In *IEEE International Conference on Evolutionary Computation*, pages 175–180, 1997. IEEE Computer Society, Washington, DC, USA. INSPEC Accession Number: 5573047. doi:10.1109/ICEC.1997.592291. In proceedings [106].

[2293] John Yen, James C. Liao, David Randolph, and Bogju Lee. A hybrid approach to modeling metabolic systems using genetic algorithms and the simplex method. In *CAIA'95: Proceedings of the 11th Conference on Artificial Intelligence for Applications*, pages 277–285, 1995. IEEE Computer Society, Washington, DC, USA. ISBN: 0-8186-7070-3. doi:10.1109/CAIA.1995.378811. Online available at http://citeseer.ist.psu.edu/33461.html [accessed 2008-07-23].

[2294] John Yen, David Randolph, Bogju Lee, and James C. Liao. A hybrid genetic algorithm for the identification of metabolic models. In *TAI'95: Proceedings of the Seventh International Conference on Tools with Artificial Intelligence*, pages 4–7, November 5–8, 1995, Herndon, VA, USA. IEEE Computer Society, Washington, DC, USA. ISBN: 0-8186-7312-5. INSPEC Accession Number: 5162198. doi:10.1109/TAI.1995.479371.

[2295] Gwoing Tina Yu and Peter Bentley. Methods to evolve legal phenotypes. In *PPSN V: Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*, pages 280–291, 1998. doi:10.1007/BFb0056843. In proceedings [624]. Online available at http://www.cs.ucl.ac.uk/staff/p.bentley/YUBEC2.pdf [accessed 2007-08-17].

[2296] Gwoing Tina Yu and Julian Francis Miller. Neutrality and the evolvability of boolean function landscape. In *EuroGP'01: Proceedings of the 4th European Conference on Genetic Programming*, pages 204–217, 2001. In proceedings [1423]. Online available at http://citeseer.ist.psu.edu/yu01neutrality.html and http://www.cs.mun.ca/~tinayu/index_files/addr/public_html/neutrality.pdf [accessed 2007-11-03], see also http://www.cs.mun.ca/~tinayu/index_files/addr/public_html/neutralityTalk.pdf [accessed 2007-11-03].

[2297] Gwoing Tina Yu and Julian Francis Miller. Finding needles in haystacks is not hard with neutrality. In *EuroGP'02: Proceedings of the 5th European Conference on Genetic Programming*, pages 13–25, 2002. In proceedings [737]. Online available at http://citeseer.ist.psu.edu/yu02finding.html and http://www.cs.mun.ca/~tinayu/index_files/addr/public_html/EuroGP2002.pdf [accessed 2007-11-02].

[2298] Gwoing Tina Yu, Rick Riolo, and Bill Worzel, editors. *Genetic Programming Theory and Practice III, Proceedings of the Genetic Programming Theory Practice 2005 Workshop (GPTP-2005)*, volume 9 of *Genetic Programming Series*, May 12-14, 2005, The Center for the Study of Complex Systems (CSCS), University of Michigan, Ann Arbor, Michigan, USA. Springer. ISBN: 978-0-38728-110-0. See http://www.cscs.umich.edu/gptp-workshops/gptp2005/ [accessed 2007-09-28].

[2299] Gwoing Tina Yu, Lawrence Davis, Cem M. Baydar, and Rajkumar Roy, editors. *Evolutionary Computation in Practice*, volume 88/2008 of *Studies in Computational Intelligence*. Springer, January 2008. ISBN: 978-3-54075-770-2. doi:10.1007/978-3-540-75771-9. Series editor: Janusz Kacprzyk.

[2300] Tina Gwoing Yu. Program evolvability under environmental variations and neutrality. In *Advances in Artificial Life*, pages 835–844, 2007. doi:10.1007/978-3-540-74913-4_84. In proceedings [614], see also [2301].

[2301] Tina Gwoing Yu. Program evolvability under environmental variations and neutrality. In *Genetic and Evolutionary Computation Conference – Companion Material, GECCO 2007*, pages 2973–2978, 2007. doi:10.1145/1274000.1274041. In proceedings [2038]: Workshop Session – Evolution of natural and artificial systems - metaphors and analogies in single and multi-objective problems. Online available at http://doi.acm.org/10.1145/1274000.1274041 [accessed 2009-02-20]. See also [2300].

[2302] D. Yuret and M. Maza. A genetic algorithm system for predicting the oex. *Technical Analysis of Stocks & Commodities*, pages 58–64, June 1994. Online available at http://www.denizyuret.com/pub/tasc94.ps.gz and http://citeseer.ist.psu.edu/yuret94genetic.html [accessed 2007-08-24].

[2303] Deniz Yuret and Michael de la Maza. Dynamic hill climbing: Overcoming the limitations of optimization techniques. In *Proceedings of the Second Turkish Symposium on Artificial Intelligence and Neural Networks*, pages 208–212, June 24–25, 1993, Boğaziçi University, Istanbul, Turky. Online available at http://citeseer.ist.psu.edu/yuret93dynamic.html and http://www.denizyuret.com/pub/tainn93.html [accessed 2007-09-11].

Z

[2304] Vladimir Zakian. New formulation for the method of inequalities. *Proceedings of the Institution of Electrical Engineers*, 126:579–584, 1979. See also [2305].

[2305] Vladimir Zakian. New formulation for the method of inequalities. In Madan G. Singh, editor, *Systems and Control Encyclopedia*, volume 5, pages 3206–3215. Pergamon Press, New York, USA, 1987. ISBN: 978-0-08028-709-6, 0-0802-8709-3. See also [2304].

[2306] Vladimir Zakian. Perspectives on the principle of matching and the method of inequalities. Control systems centre report 769, Control Systems Centre, University of Manchester Institute of Science and Technology (UMIST), P.O. Box 88, Sackville Street, Manchester M60 1QD, UK, 1992. See also [2307].

[2307] Vladimir Zakian. Perspectives on the principle of matching and the method of inequalities. *International Journal of Control*, 65(1):147–175, September 1996. ISSN: 0020-7179, 1366-5820. CODEN: IJCOAZ. doi:10.1080/00207179608921691. See also [2306].

[2308] Vladimir Zakian and U. AI-Naib. Design of dynamical and control systems by the method of inequalities. *IEE Proceedings D: Control Theory & Applications*, 120(11): 1421–1427, 1973. ISSN: 0143-7054.

[2309] A. M. S. Zalzala, editor. *First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA)*, volume 414, September 12–14, 1995, Scheffield, UK. IEE Conference Publication, Institution of Engineering and Technology. ISBN: 978-0-85296-650-1.

[2310] Michael Zapf and Thomas Weise. Offline Emergence Engineering For Agent Societies. In *Proceedings of the Fifth European Workshop on Multi-Agent Systems (EUMAS'07)*, December 14, 2007, Elmouradi Hotel, Hammamet, Tunesia. Also presented at the co-located Fifth Technical Forum Group (TFG5). Online available at http://www.it-weise.de/documents/files/ZW2007EUMASTR.pdf [accessed 2009-06-26]. See also [2311].

[2311] Michael Zapf and Thomas Weise. Offline Emergence Engineering For Agent Societies. Kasseler Informatikschriften (KIS) 2007, 8, University of Kassel, FB16, Distributed Systems Group, Wilhelmshöher Allee 73, 34121 Kassel, Germany, December 7, 2007. Persistent Identifier: urn:nbn:de:hebis:34-2007120719844. Online available at https://kobra.bibliothek.uni-kassel.de/handle/urn:nbn:de:hebis:34-2007120719844 and http://www.it-weise.de/documents/files/ZW2007EUMASTR.pdf [accessed 2007-11-20], see also [2310].

[2312] Marvin Zelen and Norman C. Severo. Probability functions. In Milton Abramowitz and Irene A. Stegun, editors, *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*, chapter 26. Dover Publications / National Bureau of Standards, first. (new ed june 1, 1965) edition, 1964. ISBN: 978-0-48661-272-0.

[2313] Guoli Zhang and Haiyan Lu. Hybrid real-coded genetic algorithm with quasi-simplex technique. *IJCSNS International Journal of Computer Science and Network Security*, 6(10):246–255, October 2006. ISSN: 1738-7906. Online available at http://paper.ijcsns.org/07_book/200610/200610B15.pdf and [accessed 2008-07-29].

[2314] Guoli Zhang, Hai Yan Lu, Gengyin Li, and Hong Xie. A new hybrid real-coded genetic algorithm and application in dynamic economic dispatch. In *WCICA 2006. The Sixth World Congress on Intelligent Control and Automation*, volume 1, pages 3627–3632, June 21–23, 2006, Dalian, China. ISBN: 1-4244-0332-4. INSPEC Accession Number: 9187799. doi:10.1109/WCICA.2006.1713046.

[2315] P. Zhang and A.H. Coonick. Coordinated synthesis of pss parameters in multi-machine power systems using the method of inequalities applied to genetic algorithms. *IEEE Transactions on Power Systems*, 15(2):811–816, May 2000. Online available at http://www.lania.mx/~ccoello/EMOO/zhang00.pdf.gz and http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.8.9768 [accessed 2008-11-14].

[2316] Song Zhang and David H. Laidlaw. DTI fiber clustering and cross-subject cluster analysis. In *Proceedings International Society for Magnetic Resonance in Medicine (ISMRM)*, May 2005. Miami, FL. Online available at http://www.cs.brown.edu/research/vis/docs/pdf/Zhang-2005-DFC.pdf [accessed 2007-08-11].

[2317] Feng Zhao and Leonidas Guibas. *Wireless Sensor Networks: An Information Processing Approach*. Morgan Kaufmann, July 6, 2004. ISBN: 978-1-55860-914-3.

[2318] Jinghui Zhong, Xiaomin Hu, Jun Zhang, and Min Gu. Comparison of performance between different selection strategies on simple genetic algorithms. In *CIMCA'05: Proceedings of the International Conference on Computational Intelligence for Modelling, Control and Automation and International Conference on Intelligent Agents, Web Technologies and Internet Commerce Vol-2 (CIMCA-IAWTIC'06)*, pages 1115–1121, November 28–30, 2005. IEEE Computer Society, Washington, DC, USA. ISBN: 076-9-52504-002-.

[2319] Chi Zhou, Peter C. Nelson, Weimin Xiao, and Thomas M. Tirpak. Discovery of classification rules by using gene expression programming. In *Proceedings of the International Conference on Artificial Intelligence (IC-AI'02)*, pages 1355–1361, June 2002. Las Vegas, U.S.A.

[2320] Chi Zhou, Weimin Xiao, Thomas M. Tirpak, and Peter C. Nelson. Evolving accurate and compact classification rules with gene expression programming. *IEEE Transactions on Evolutionary Computation*, 7(6):519–531, December 2003. ISSN: 1089-778X.

[2321] Wanlei Zhou and Andrzej Goscinski. An analysis of the web-based client-server computing models. In *Proceedings of the Asia-Pacific Web Conference (APWeb'98)*, pages 343–348, September 1998, Beijing. Online available at http://citeseer.ist.psu.edu/296101.html [accessed 2007-08-13].

[2322] Jianping Zhu, Pete Bettinger, and Rongxia Li. Additional insight into the performance of a new heuristic for solving spatially constrained forest planning problems. *Silva Fennica*, 41(4):687–698, 2007. ISSN: 0037-5330. Online available at http://www.metla.fi/silvafennica/full/sf41/sf414687.pdf [accessed 2008-08-23].

[2323] Kenny Qili Zhu. A diversity-controlling adaptive genetic algorithm for the vehicle routing problem with time windows. In *15th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'03)*, pages 176–183, November 3–5, 2003, Sacramento, California, USA. IEEE Computer Society, Washington, DC, USA. ISBN: 0-7695-2038-3. INSPEC Accession Number: 7862120. doi:10.1109/TAI.2003.1250187.

[2324] Liming Zhu, Roger L. Wainwright, and Dale A. Schoenefeld. A genetic algorithm for the point to multipoint routing problem with varying number of requests. In *Proceedings of the IEEE World Congress on Computational Intelligence, The 1998 IEEE International Conference on Evolutionary Computation*, pages 171–176, 1998. doi:10.1109/ICEC.1998.699496. In proceedings [1001]. Online available at http://euler.mcs.utulsa.edu/~rogerw/papers/Zhu-ICEC98.pdf and http://citeseer.ist.psu.edu/382506.html [accessed 2008-07-21].

[2325] Karin Zielinski and Rainer Laur. Stopping criteria for constrained optimization with particle swarms. In *Proceedings of the Second International Conference on Bioinspired Optimization Methods and their Application, BIOMA 2006*, pages 45–54, 2006. In proceedings [669]. Extended version: [2326]. Online available at http://www.item.uni-bremen.de/staff/zilli/zielinski06stopping_PSO.pdf [accessed 2007-09-13].

[2326] Karin Zielinski and Rainer Laur. Stopping criteria for a constrained single-objective particle swarm optimization algorithm. *Informatica*, 31(1): 51–59, 2007. ISSN: Print: 0350-5596, Web: 1854-387. Extended version of [2325]. Online available at http://www.item.uni-bremen.de/staff/zilli/zielinski07informatica.pdf and http://www.informatica.si/vols/vol31.html [accessed 2007-09-13].

[2327] Anatas Žilinskas. Algorithm as 133: Optimization of one-dimensional multimodal functions. *Applied Statistics*, 27(3):367–375, 1978. ISSN: 00359254. doi:10.2307/2347182.

[2328] Stanley Zionts, editor. *Proceedings of the 2nd International Conference on Multiple Criteria Decision Making (MCDM'1977)*, 1977, Buffalo, New York, USA.

[2329] Eckart Zitzler and Lothar Thiele. An evolutionary algorithm for multiobjective optimization: The strength pareto approach. Technical Report 43, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology Zürich (ETH), Gloriastrasse 35, CH-8092 Zurich, Switzerland, May 1998. Online available at http://www.tik.ee.ethz.ch/sop/publicationListFiles/zt1998a.pdf and http://citeseer.ist.psu.edu/225338.html [accessed 2007-07-29].

[2330] Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation*, 8(2): 173–195, 2000. Online available at http://sci2s.ugr.es/docencia/cursoMieres/EC-2000-Comparison.pdf and http://citeseer.comp.nus.edu.sg/362080.html [accessed 2008-04-06].

[2331] Eckart Zitzler, Kalyanmoy Deb, Lothar Thiele, Carlos A. Coello Coello, and David Corne, editors. *Evolutionary Multi-Criterion Optimization, Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization (EMO'01)*, volume 1993/2001 of *Lecture Notes in Computer Science (LNCS)*, March 7–9, 2001, Zurich, Switzerland. Springer-Verlag, Berlin. ISBN: 3-5404-1745-1. See http://www.tik.ee.ethz.ch/emo/ [accessed 2007-09-11].

[2332] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Technical Report 103, Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, CH-8092 Zurich, Switzerland, May 2001. Online available at http://www.tik.ee.ethz.ch/sop/publicationListFiles/zlt2001a.pdf and http://citeseer.ist.psu.edu/514031.html [accessed 2007-07-29]. Errata added 2001-09-27.

[2333] Constantin Zopounidis, editor. *Proceedings of the 18th International Conference on Multiple Criteria Decision Making (MCDM'2006)*, June 9–13, 2006, MAICh (Mediterranean Agronomic Institute of Chania) Conference Centre, Chania, Crete, Greece. See http://www.dpem.tuc.gr/fel/mcdm2006/ [accessed 2007-09-10].

[2334] Jie Zuo, Changjie Tang, and Tianqing Zhang. Mining predicate association rule by gene expression programming. In Xiaofeng Meng, Jianwen Su, and Yujun Wang, editors, *WAIM'02: Proceedings of the Third International Conference on Advances in Web-Age Information Management*, volume 2419/2002 of *Lecture Notes in Computer Science (LNCS)*, pages 281–294. Springer-Verlag, August 11–13, 2002, Beijing, China. ISBN: 3-5404-4045-3. Computer Department, Sichuan University China.

[2335] Katharina Anna Zweig. *On Local Behavior and Global Structures in the Evolution of Complex Networks*. PhD thesis, Fakultät für Informations- und Kognitionswissenschaften der Eberhard-Karls-Universität Tübingen, July 2007. Online available at http://www-pr.informatik.uni-tuebingen.de/mitarbeiter/katharinazweig/downloads/Diss.pdf [accessed 2008-06-12]. Zweig was formerly known as Lehmann.

Index

List of Figures

List of Tables

List of Algorithms

List of Listings