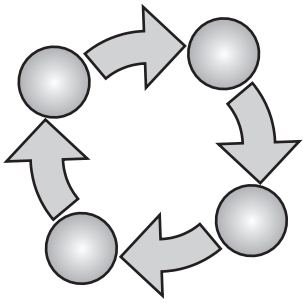
Global Optimization Algorithms – Theory and Application –



Thomas Weise
Version: January 4, 2008

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 meta-heuristics like simulated annealing, hill climbing, tabu search, and random optimization.

With this book, we want to address two major audience groups:

1. It can help students since we try to describe the algorithms in an understandable, consistent way and, maybe even more important, include all background knowledge needed to understand them. Thus, you can find summaries on stochastic theory and theoretical computer science in Part IV on page 501. 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 maybe will 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 653.

In order to maximize the utility of this electronic book, it contains automatic, clickable links. They are not highlighted with color so the book is still b/w printable, but you can click on

- entries in the table of contents,
- citation references like [1],
- page references like “47”,

- references such as "see Figure 2.1 on page 48" to sections, figures, tables, and listings, and
- 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 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 to ease understanding will be given. In the second part starting at page 275, we elaborate on different application examples in detail. With the Sigoa framework, a possible Java implementation of the optimization algorithms is discussed and we show how some of solutions of the previous problem instances can be realized in Part III on page 367. Finally, in the last part following at page 501, 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. Additionally, the sequence of the content is not very good. It will probably be changed. Therefore, I try to update, correct, and improve this book continuously.

This 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`.

[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.

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

If you want to provide feedback or find errors, missing things, want to criticize something, or have any additional ideas or suggestions, I would be very happy. Do not hesitate to contact me via my email address `tweise@gmx.de`.

At many places in this book we refer to Wikipedia [2] which is a great source of knowledge. Wikipedia 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 may add to the book's utility.

References in form of links to websites, URLs, and URIs are always annotated with a version, which denotes the date when I visited them. The reference then regards the content at the location pointed to at this specific date, since a website's content may change over time. Hence, we cannot guarantee for its current or future validity, correctness, or even legality.

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 [3]. 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* [4] which, together with the biological evolution [5], 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 science which even extends into daily life [6].

Global optimization[2] is the branch of applied mathematics and numerical analysis that deals with the optimization of single or multiple, possible even conflicting, criteria. These criteria are expressed as a set of mathematical functions[3] $F = \{f_1, f_2, \ldots, f_n\}$, the so-called objective functions. The result of an optimization process is the set of inputs for which these objective functions return optimal values.

Definition 1 (Objective Function). An objective function $f : X \mapsto Y \subseteq \mathbb{R}$ is a function which is subject to optimization. Its codomain Y and its range must be a subset of the real numbers. ($Y \subseteq \mathbb{R}$). The domain X of f can

[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 introduced in set theory in Definition 114 on page 510.

contain any given type of elements like numbers, lists, construction plans, and so on, depending on the optimization problem. Objective functions are not necessarily mere mathematical expressions but can be complex algorithms that, for example, involve numerous simulations.

Global optimization comprises all techniques that can be used to find the best elements x_i in the domain X in respect to the criteria $f \in F$. The difference between optimization algorithms and search algorithms[4] is subtle. When performing a search algorithm, we know a definite criterion that tells us whether an element x_i is a solution or not. With this criterion, we can for instance quickly find the position of an element in a list. In global optimization algorithms on the other hand we possibly are not sure about the characteristics of the solutions beforehand and have only the objective functions which describe if a given configuration is good or not. Since these functions provide a more general problem definition, we could consider search algorithms as special case of global optimization methods.

1.1 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 [7]. Before digging any deeper into the matter, we will attempt to build a taxonomy of these algorithms as overview and discuss some basic use cases.

1.1.1 Taxonomy According to Method of Operation

Generally, global optimization algorithms can be divided in two basic classes: deterministic and probabilistic algorithms.

Deterministic algorithms (see also Definition 201 on page 588) 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[5]. If the relation between a solution candidate and its "fitness" is however cannot be understood or observed, neighboring solution candidates may differ largely in their utility, 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[6] come into play. Here,

[4] State space search algorithms are discussed in Chapter 15 on page 251.

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

[6] The common properties of probabilistic algorithms are specified in Definition 208 on page 591.

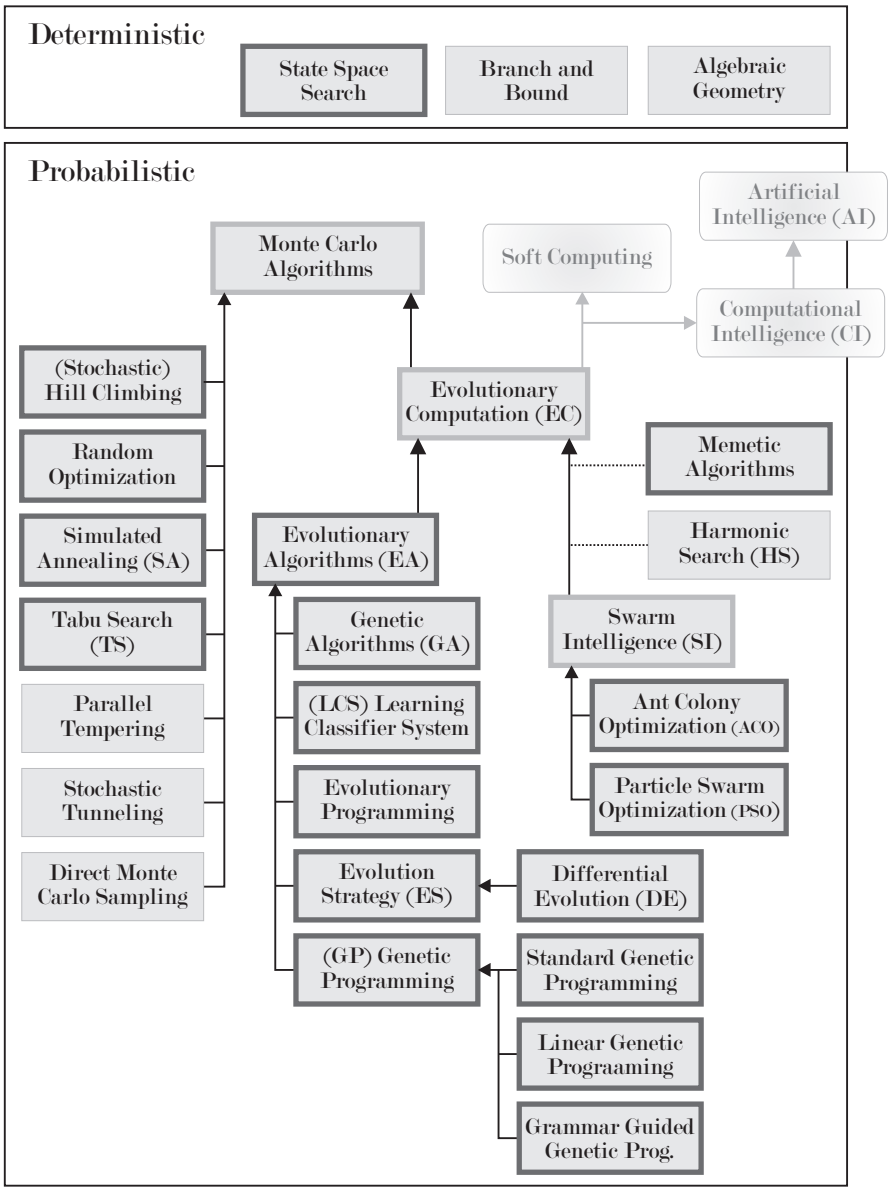


Fig. 1.1: The taxonomy of global optimization algorithms.

most often the Monte Carlo algorithms[7] are applied which trade guaranteed correctness of the solution in for a shorter runtime. This does not mean that the results obtained using them are totally 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 as used in global optimization are functions that help decide which one of a set of possible solutions is to be examined next. Heuristics can be used by both, deterministic as well as probabilistic algorithms. Deterministic algorithms usually will employ heuristics in order to define the processing order of the solution candidates. One example for such strategies are informed searches, as discussed in Section 15.3 on page 258. 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 2 (Heuristic). A heuristic[8] [8, 9, 10] is a part of an optimization algorithm. It 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.

Definition 3 (Metaheuristic). A metaheuristic[9] is a heuristic method for solving a very general class of problems. It combines objective functions or heuristics in a hopefully efficient way.

Metaheuristics often work population-based or use a model of some natural phenomenon or physical process as heuristic function. Simulated annealing for example decides which solution candidate to be evaluated 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.

In principle, all the probabilistic optimization algorithms that we consider in this book as well as some of the deterministic ones (the greedy state space search for example) are metaheuristics.

An important class of probabilistic, Monte Carlo metaheuristics is evolutionary computation[10]. It encompasses all such 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[11] as well as a part of the artificial intelligence[12] area. Its most important members

[7] See Definition 210 on page 592 for a in-depth discussion of the Monte Carlo-type probabilistic algorithms

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

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

[10] http://en.wikipedia.org/wiki/Evolutionary_computation [accessed 2007-09-17]

[11] http://en.wikipedia.org/wiki/Soft_computing [accessed 2007-09-17]

[12] http://en.wikipedia.org/wiki/Artificial_intelligence [accessed 2007-09-17]

are evolutionary algorithms and swarm intelligence, which will be discussed in-depth in this book.

Besides the evolutionary approaches, which are mostly nature-inspired, there exist also methods that copy physical processes like simulated annealing and parallel tempering, as well as purely artificial techniques 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 in this book 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 is wants to solve a problem with such an approach is however more interested in its "interfacing features" such as speed and precision.

An interesting thing is that these are again (obviously) conflicting objectives. 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[13] 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 algorithm, we can distinguish two main types of optimization use cases.

Definition 4 (Online Optimization). Online optimization problems represent tasks that need to be solved quickly, such as robot localization, load balancing, services composition for business processes in the running IT system of an enterprise (see for example Section 18.2.1 on page 313), or updating a factory's machine job schedule after new orders came in. Here we generally have only a time span between some ten milliseconds to some minutes to find a good solution and will generally trade in optimality for speed. From the examples, it furthermore 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 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 example Section 18.1 on page 299), or creating long-term schedules for transportation crews. Such optimization processes will usually be carried out only once in a long time.

[13] Pareto frontiers will be discussed in .

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

1.2 Optima, Gradient Descend, and Search Space

1.2.1 Local and Global Optima

Global optimization is about finding optimal configurations. So it cannot be a bad idea to start out by defining what an optimum[14] is. In the case of a single function, i.e. $F = \{f\}$, an optimum is either a maximum or a minimum. Figure 1.2 illustrates such a function f defined over a two-dimensional search space $X = (X_1, X_2)$. As outlined there, we distinguish between local and global optima. A global optimum is an optimum of the whole domain X while a local optimum is only an optimum of one of its subsets.

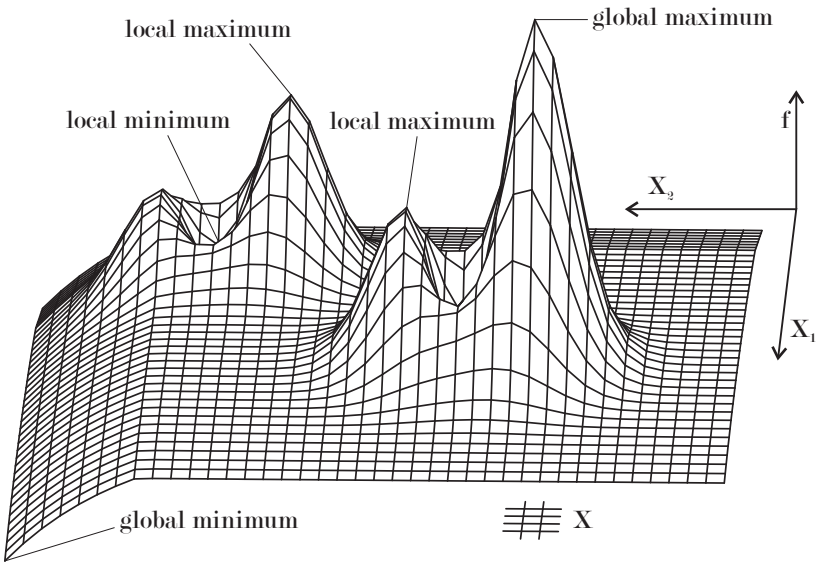


Fig. 1.2: Global and local optima of a two-dimensional function.

Definition 6 (Local Maximum). A (local) maximum $\hat{x}_l \in X$ of an objective function $f : X \mapsto \mathbb{R}$ is an input element with $f(\hat{x}_l) \geq f(x)$ for all x neighboring $\hat{x}_l$.

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

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

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

Definition 7 (Local Minimum). A (local) minimum $\check{x}_l \in X$ of an objective function $f : X \mapsto \mathbb{R}$ is an input element with $f(\check{x}_l) \leq f(x)$ for all x neighboring $\check{x}_l$.

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

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

Definition 8 (Local Optimum). An (local) optimum $x_l^\star \in X$ of an objective function $f : X \mapsto \mathbb{R}$ is either a local maximum or a local minimum (or both).

Definition 9 (Global Maximum). A global maximum $\hat{x} \in X$ of an objective function $f : X \mapsto \mathbb{R}$ is an input element with $f(\hat{x}) \geq f(x) \forall x \in X$.

Definition 10 (Global Minimum). A global minimum $\check{x} \in X$ of an objective function $f : X \mapsto \mathbb{R}$ is an input element with $f(\check{x}) \leq f(x) \forall x \in X$.

Definition 11 (Global Optimum). A global optimum $x^\star \in X$ of an objective function $f : X \mapsto \mathbb{R}$ is either a global maximum or a global minimum (or both).

Even a one-dimensional function may have more than one global maximum, multiple minima, or both in its domain X, take the sinus function for example. An optimization algorithm may thus yield a list of optimal inputs rather than a single maximum or minimum. Therefore, we define the optimal set as[15]:

Definition 12 (Optimal Set). The optimal set $X^\star$ is the set containing all optimal elements so that $x_i^\star$ is optimal $\Leftrightarrow x_i^\star \in X^\star \subseteq X$.

Furthermore, when more than one objective function is to be optimized, the definition of what is an optimum can change. Especially if multiple criteria conflict, there exist different methods to specify which solutions are optimal. In Section 1.3 on page 12 we discuss the most prominent approaches for this case.

1.2.2 Restrictions of the Search Space

Figure 1.3 shows how the application of a global optimization algorithm for the maximization of a real-valued function $y = f(x) \in \mathbb{R}$ may look like. The domain X of the function, for instance the real numbers $\mathbb{R}$ too, is restricted

[15] see also Definition 11

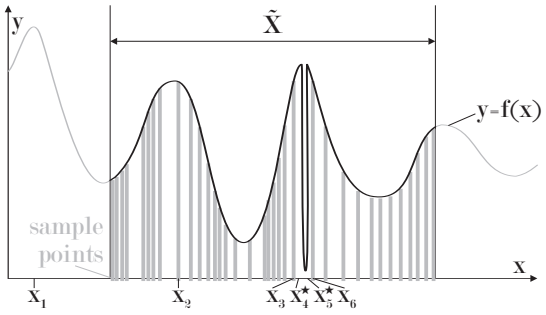


Fig. 1.3: Possible results of global optimization.

to a subset $\tilde{X} \subseteq X$ that can be processed by the algorithm. When using computers, this is always done at least implicitly by the data types selected – 8 bytes of floating point data for example can represent 2^{64} different real numbers at most. In the example given by Figure 1.3, the true maximum x_1 of the function f cannot be found since it is situated outside the accessible domain $\tilde{X}$.

Definition 13 (Individual). An individual is an element x of the examinable part $\tilde{X}$ of the solution space X. We will subsequently use the terms *solution candidate* or *individual* synonymously when talking about the sample inputs of an optimization algorithm (although the term *individual* originally solely stems from evolutionary computation).

Figure 1.4 illustrates how the accessible domain of the example Figure 1.2 that we have introduced in Section 1.2 for instance could look like from the viewpoint of an optimization algorithm and which individuals are now considered as optimal.

Even in this restricted space, optimization algorithms can only rely on sample information (represented by gray lines) since it is not possible to evaluate the objective function(s) for all possible inputs $x \in \tilde{X}$ (all possible 2^{64} floating point numbers in the above example, for instance). If no optimal boundaries are known beforehand, it thus cannot be determined if a good solution candidate found is a global optimum or not. In Figure 1.3, the true optimal set $X^\star = \{x_4^\star, x_5^\star\}$ was not discovered by the optimization algorithm. Instead, the set $\{x_2, x_3, x_6\}$ has been returned.

Another example for the constraints of the space of possible solution is solving the artificial ant problem[16] [11] with genetic programming. Artificial ants are simulated little insects driven by a small program describing their behavior. They are placed on a map with food and obstacles. Optimization is

[16] See Section 17.4 on page 284 for the artificial ant problem and Chapter 4 on page 139 for genetic programming.

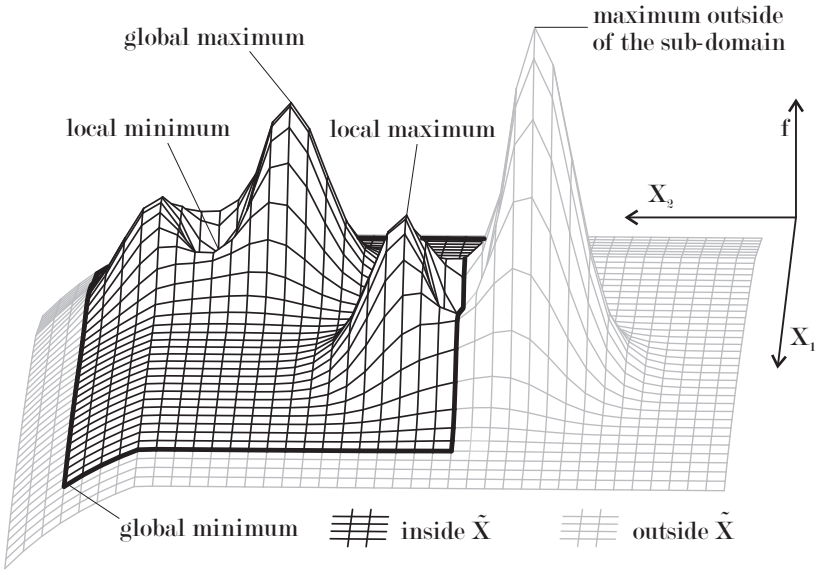


Fig. 1.4: Global and local optima of a two-dimensional function.

used to find a program $x^\star \in X$ allowing the ant to pile a maximum of food. An objective function $y = f(x) \in Y \subseteq \mathbb{R}^+$ is constructed which computes positive real numbers denoting the amount of food piled by the ant driven by the program $x \in X$. Since we (and also the computers we use) are limited in memory and time, we cannot evaluate programs of huge or even infinite length so we have to restrict X to a subset $\tilde{X}$ of programs which contain, let's say, 100 instructions at most. Then, $\tilde{X}$ will contain programs that are one, two, three, ... – up to one hundred instructions long. If we further have four different instructions (with only one parameter each) and three constants to our disposal, there roughly exist $\sum_{i=1}^{100}(4*3)^i \approx 9*10^{107}$ different programs in $\tilde{X}$. Even if evaluating a program takes only one millisecond, we would need $\approx 3*10^{97}$ years to check all of them. Genetic programming therefore applies an evolutionary heuristic which iteratively helps us to find prospecting sample points x. The optimization process is then narrowed to these interesting individuals.

In the subsequent text of this book we will take the understanding of this matter for granted and not explicitly distinguish between the true optimum of a function, its global optima inside $X^\star$, and the optima found by an optimization algorithm.

It may however be an interesting fact to know that there exist proofs that some optimization algorithms (like simulated annealing and random optimiza-

tion) will always find the global optimum (when granted a very long, if not infinite, processing time).

1.2.3 Fitness Landscapes and Gradient Descend

Figure 1.2 on page 8 represents an example for the objective values of a function $f : \mathbb{R}^2 \mapsto \mathbb{R}$. Such a function can be considered as a *field*[17] an assignment of a (quantity (the objective values) to every point of the (two-dimensional) space. From its illustration, it also looks very much like a landscape with hills and valleys.

Definition 14 (Fitness Landscape). In biology, a fitness landscape[18] is a visualization of the relationship of the genotypes to their corresponding reproduction probability [12, 13, 14, 15, 16]. In global optimization algorithms, it displays the relation of the solution candidates to their fitness[19] or objective values [17, 18, 19].

Definition 15 (Gradient). A gradient[20] 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 $grad(f)$.

Optimization algorithms depend on some form of gradient in objective or fitness space order to find good individuals. Of course, we normally do not directly differentiate the objective functions – in most cases, the search space $\tilde{X}$ is not even $\mathbb{R}^n$. Generally, we use samples of the search space to approximate the gradient. By comparing two individuals $x_1, x_2 \in \tilde{X}$ and finding, for instance, $f(x_1) > f(x_2)$, we estimate that the gradient at x_2 would somehow point into the direction of x_1.

By descending this gradient (into the opposite direction), we can hope to find an $x_3 < x_2$ and finally the global minimum. Hence, Figure 1.10a on page 24 is the best case since regardless where an optimization algorithm starts exploring the solution space, it will always find a *gradient* into the correct direction. Because of its smooth nature, Figure 1.10b and probably also Figure 1.2 can be handled by most optimization algorithms correctly.

1.3 Multi-objective Optimization

Global optimization techniques are not just applied to find the maxima or minima of single objective functions f. In many real-world design or decision

[17] http://en.wikipedia.org/wiki/Field_%28physics%29 [accessed ,]

[18] http://en.wikipedia.org/wiki/Fitness_landscape [accessed 2007-07-03]

[19] You can find a detailed discussion on fitness in evolutionary algorithms in Section 2.3 on page 65.

[20] http://en.wikipedia.org/wiki/Gradient [accessed 2007-11-06]

making problems they are applied to sets F of n functions f_i which represent multiple criteria [20, 21, 22].

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

Algorithms designed to optimize such a set F of objective functions are usually named with the prefix *multi-objective*, like multi-objective evolutionary algorithms[21]. Multi-objective optimization often means to compromise conflicting goals, for example when trying to build a car which is fast, safe, and environment-friendly. In this case, there will *always*[22] be more than one optimal solution. The tasks of global optimization are therefore

1. to find solutions that are as good as possible and
2. that are also widely different from each other [23].

Let visualize this situation by looking again at the artificial ant example[23]. The efficiency of an ant may 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 on the map. The more food it piles, the longer the distance it needs to walk. If its behavior is controlled by a clever program, it may walk along a shorter route which would not be discovered by an ant with a clumsy program. Thus, the distance it has to cover to find the food or the time it needs to do say may also be considered. If two programs produce the same results, and one is shorter (i. e. contains fewer instructions) than the other, the shorter one should be preferred. Looking closer at this example yields another realization: To find the global optimum could mean to maximize one function $f_i \in F$ and to minimize another one $f_j \in F, (i \neq j)$, so it makes no sense to speak of a *global maximum* or a *global minimum* in terms of multi-objective optimization. We will thus retreat to the optimal set $x^\star \in X^\star \subseteq \tilde{X}$. Since compromises can be defined in many ways, there exist different approaches to define what exactly optimal elements $x^\star$ are leading to different results for $X^\star$.

1.3.1 Weighted Sum

The simplest method is computing a weighted sum $g(x)$ of the functions $f_i(x) \in F$ (see Equation 1.3). The weights w_i represent the importance of the single functions and also determine if the function should be maximized ($w_i > 0$) or minimized ($w_i < 0$). Using this method, the multi-objective problem is reduced to single-objective optimization.

---

[21] see Definition 35 on page 48

[22] Notice that multiple optima *may* also occur in single-objective optimization or when optimizing consistent goals.

[23] see Section 17.4 on page 284 for more details

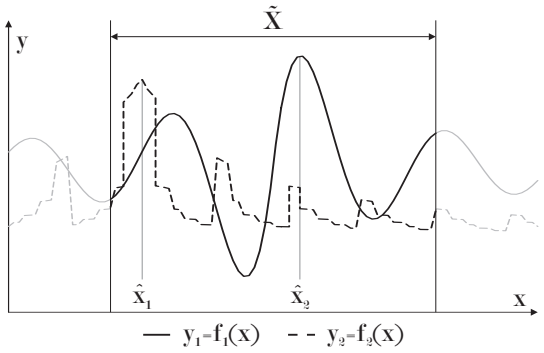


Fig. 1.5: Two functions f_1 and f_2 with different maxima $\hat{x}_1$ and $\hat{x}_2$.

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

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

The drawback of this approach is that it cannot handle functions that rise or fall with different speed[24] properly. The sum of $f_1(x) = -x^2$ and $f_2(x) = e^{x-2}$ will always disregard one of two functions, depending on the interval chosen. For small x, f_2 is negligible compared to f_1. For $x > 6$ 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 for example setting 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 37.1.3 on page 589). Figure 1.6 demonstrates the optimization using weighted sums for the example given in Figure 1.5. The weights are set to 1, which maximizes both functions f_1 and f_2 and leads to a single optimum $x^\star = \hat{x}$.

1.3.2 Pareto Optimization

Pareto efficiency[25], or Pareto optimality, is an important notion in neoclassical economics with broad applications in game theory, engineering and the social sciences [24, 25].

It defines the front of solutions that can be reached by trading-off conflicting objectives in an optimal manner. From this front, a decision maker (be it

[24] see Section 37.1.3 on page 589
or `http://en.wikipedia.org/wiki/Asymptotic_notation` [accessed 2007-07-03]

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

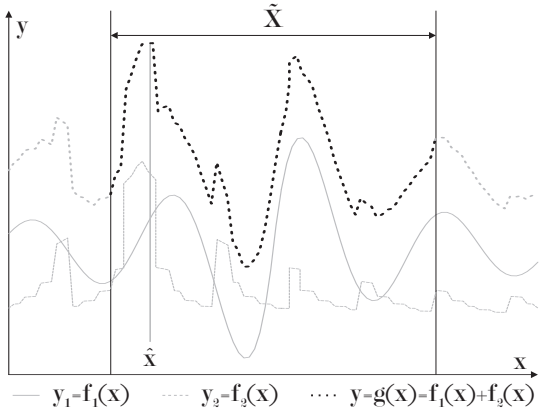


Fig. 1.6: Optimization using the weighted sum approach.

a human or another algorithm) can finally choose the configuration that, in his opinion, suits best [26, 27, 28, 29, 30].

The notation of Pareto optimal is strongly based on the definition of domination:

Definition 16 (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 objective functions. Referring to the definition of a function set in Equation 1.3 on page 13 we write:

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

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

The Pareto domination relation defines a (strict) partial order (see Definition 110 on page 509) on the set of possible objective values. In contrast, the weighted sum approach imposes a total order by projecting them into the real numbers $\mathbb{R}$.

Definition 17 (Pareto Optimal). An element $x^\star \in \tilde{X}$ is Pareto optimal (and hence, part of the optimal set $X^\star$) if it is not dominated by any other element in $\tilde{X}$. $X^\star$ is called the Pareto set or the Pareto Frontier.

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

In Figure 1.7 we illustrate the impact of Equation 1.8 on our example outlined in Figure 1.5, assuming again that f_1 and f_2 should both be maximized.

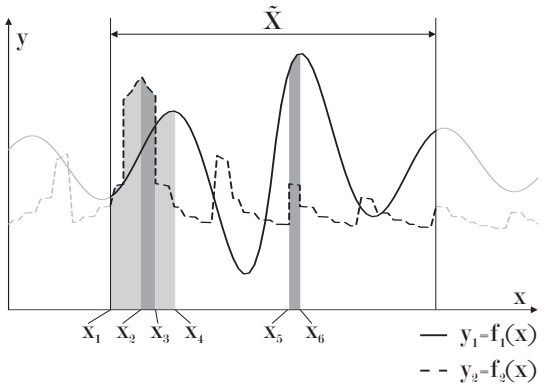


Fig. 1.7: Optimization using the Pareto Frontier approach.

The areas shaded with dark gray are Pareto optimal and thus, the optimal set $X^\star = [x_2, x_3] \cup [x_5, x_6]$ contains infinite many elements (theoretically, practically $\tilde{X}$ is always finite).

The points in light gray area between x_1 and x_2 are all 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 $\tilde{X}$ (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 it 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 $\tilde{X}$ 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 suffers – in order to increase f_1, f_2 would be decreased and vice versa. So the new point is not dominated by x_2. Although 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.

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.

Problems of Pure Pareto Optimization

The complete Pareto optimal set is often not the wanted result of an optimization algorithm and we are interested in some areas of the Pareto front only. We can again take the Artificial Ant example to visualize this problem. In Section 1.3 on page 13 we have introduced multiple additional conflicting criteria.

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

Pareto optimization may now yield for example:

- A program consisting of 100 instructions, allowing the ant to gather 50 food items when walking a distance of 500 length units.
- A program consisting of 100 instructions, allowing the ant to gather 60 food items when walking a distance of 5000 length units.
- A program consisting of 10 instructions, allowing the ant to gather 1 food item when walking a distance of 5 length units (straight ahead).
- A program consisting of 0 instructions, allowing the ant to gather 0 food item when walking a distance of 0 length units (straight ahead).

The Pareto optimal set 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, when the size-limit of the non-dominated list is reached, some algorithms use a clustering technique to prune it while maintaining diversity. This is normally wanted, since it will preserve a broad scan of the Pareto frontier. In this case however, 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 will be discarded. And, last but not least, non-dominated elements have a higher probability to reproduce. This then leads inevitably to the creation of a great proportion of useless offspring. In the next generation, the useless offspring will need a good share of the processing time to be evaluated. So there are some reasons to force the optimization process into a wanted direction. In Section 18.2.2 on page 321 you can find an illustratitive discussion on the drawbacks of strict Pareto optimization in a practical example (evolving web service compositions).

1.3.3 The Method of Inequalities

One method of dealing with these problems is the *Method of Inequalities* (MOI) [31, 32, 33] which has its roots in operations research.

We can apply this method by specifying a goal range $[\check{g}_i, \hat{g}_i]$ for each objective function f_i. Based on inequalities we can define three categories. Each individual $x \in \tilde{x}$ belongs to one of them:

1. It fulfills none of the goals,
$$(f_i(x) < \check{g}_i) \vee (f_i(x) > \hat{g}_i) \; \forall \; i \in 1 \ldots |F| \tag{1.9}$$
2. it fulfills all of the goals, or
$$\check{g}_i \leq f_i(x) \leq \hat{g}_i \; \forall \; i \in 1 \ldots |F| \tag{1.10}$$
3. it fulfills some (but not all) of the goals.
$$(\exists \; \check{g}_i \leq f_i(x) \leq \hat{g}_i) \wedge (\exists \; (f_j(x) < \check{g}_j) \vee (f_j(x) > \hat{g}_j) \tag{1.11}$$

Using these groups we can create a new comparison mechanism:

1. The individuals 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 individuals that are in the same group are compared on basis on the Pareto domination relation.

Now the optimization process will be driven into the direction of the interesting part of the Pareto front. Less effort will be spent in creating and evaluating individuals in parts of the search space that cannot contain any valid solution.

Other Related Methods (TODO)

Goal Attainment [34] and *Goal Programming*[26] [35] are techniques very near to the method of inequalities. Often, characteristics of these methods are added to evolutionary algorithms [36, 37, 38, 39, 40].

1.3.4 External Decision Maker

To circumvent all the limitations of the previous approaches, Fonseca and Fleming introduced their general concept of an external decision maker which (or who) decides which solution candidates prevail [41, 22]. The basic idea behind this is that Pareto optimization provides only a partial order[27] between the individuals. There can be two individuals $x_1, x_2 \in \tilde{X}$ that do not dominate each other. A special case of this situation is the non-dominated set, the so-called *Pareto-front* which we try to estimate with the optimization process.

[26] `http://en.wikipedia.org/wiki/Goal_programming` [accessed 2007-07-03]
[27] A definition of *partial order* relations is specified in Definition 110 on page 509.

Fitness assignment however requires some sort of total order[28], 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 themselves by performing for example Pareto ranking as introduced in Section 2.3.3 on page 67, where the number of individuals prevailing 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 [42]. This region will then exclude for example the infeasible (but Pareto-optimal) programs for the artificial ant discussed in . What the user wants is a detailed scan of this region, which often cannot be delivered by pure Pareto optimization since there, the optimal individuals will be distributed over the complete, much broader Pareto front.

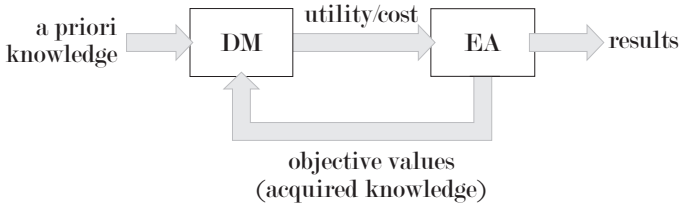


Fig. 1.8: An external decision maker providing an EA with utility values.

Here comes the external decision maker as an expression of the user's preferences [43] into play, as illustrated in Figure 1.8. The task of this decision maker provides a cost function $c : \mathbb{R}^n \mapsto \mathbb{R}$ (or utility function, if the underlying optimizer is maximizing) which maps the space of objective values $\mathbb{R}^n$ to the space of real numbers $\mathbb{R}$. 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 c is free and may incorporate any of the previously mentioned methods. c 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, or 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.

[28] The concept of *total orders* is introduced in Definition 111 on page 509.

Fonseca and Fleming's make a clear distinction between fitness and cost values. Cost values have some meaning outside the EA and are based on user preferences. Fitness values on the other hand are an internal construct of the evolutionary process with no meaning outside the EA. Fitness is to be computed on the basis of cost values [44, 43, 45].

1.3.5 Prevalence Optimization

In this section, we define the prevalence relation, which we will use in the further course of this book as a general way of the comparing individuals in the optimization process. In principle, it has the same features as the method of Fonseca and Fleming just discussed. The major difference is that we do not use any form of cost function but simply replace the Pareto comparator by a freely defined scheme. This way, all other optimization algorithms (especially many of the evolutionary approaches) which rely on Pareto comparisons can be used in their original form while retaining the ability of scanning special regions of interests of the optimal front provided by the decision making concept.

Like Fonseca and Fleming, we relaxing the domination[29] relation by providing a free definable comparator function c_F. Combined with the fitness assignment strategies discussed later[30], it covers the same multi-objective techniques as proposed in [41] and [46, 36].

Generally, the prevalence concept is just a form of notation, but giving it an own defined name will help to distinguish it from pure Pareto optimization.

Definition 18 (Prevalence). An element x_1 prevails over an element x_2 ($x_1 \succ x_2$) if the application-dependent, transitive comparator function $c_F(x_1, x_2) \in \mathbb{R}$ returns a value less than 0.

$$(x_1 \succ x_2) \Leftrightarrow c_F(x_1, x_2) < 0 \tag{1.12}$$

$$(x_1 \succ x_2) \wedge (x_2 \succ x_3) \Rightarrow x_1 \succ x_3 \forall x_1, x_2, x_3 \in \tilde{X} \tag{1.13}$$

Like in Pareto optimization, the prevalence comparator introduces a partial order on the set of possible objective values. The optimal set can be constructed in a way very similar to Equation 1.8:

$$x^\star \in X^\star \Leftrightarrow \nexists x \in \tilde{X} : x \neq x^\star \wedge x \succ x^\star \tag{1.14}$$

With this definition, we can cover the all the aforementioned approaches for multi-objective optimization. For illustration purposes, we will exercise it on the examples of the weighted sum ($c_{F,weightedS}$) method[31] with the weights

[29] The domination relation is discussed in Definition 16 on page 15.
[30] see Section 2.3 on page 65
[31] see Equation 1.4 on page 14

w_i as well as by the domination-based Pareto optimization[32] ($c_{F,pareto}$) with the objective directions ω_i:

$$c_{F,weightedS}(x_1, x_2) = \sum_{i=1}^{n}(w_i f_i(x_2) - w_i f_i(x_1)) = g(x_2) - g(x_1) \quad (1.15)$$

$$c_{F,pareto}(x_1, x_2) = \begin{cases} -1 \text{ if } x_1 \vdash x_2 \\ 1 \text{ if } x_2 \vdash x_1 \\ 0 \text{ else} \end{cases} \quad (1.16)$$

With the prevalence comparator as instance of Fonseca and Flemings decision making concept, we can easily solve the problem stated in Section 1.3.4 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 1.17 demonstrates one possible comparator function for the Artificial Ant problem.

$$c_{F,ant}(x_1, x_2) = \begin{cases} -1 \text{ if } (f_1(x_1) > 0 \wedge f_1(x_2) = 0) \vee \\ \quad (f_2(x_1) > 0 \wedge f_2(x_2) = 0) \vee \\ \quad (f_3(x_1) > 0 \wedge f_1(x_2) = 0) \\ 1 \text{ if } (f_1(x_2) > 0 \wedge f_1(x_1) = 0) \vee \\ \quad (f_2(x_2) > 0 \wedge f_2(x_1) = 0) \vee \\ \quad (f_3(x_2) > 0 \wedge f_1(x_1) = 0) \\ c_{F,pareto}(x_1, x_2) \quad \text{otherwise} \end{cases} \quad (1.17)$$

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.

1.4 Complicated Fitness Landscapes

1.4.1 Premature Convergence and Multi-Modality

One of the greatest problems in global optimization is that we most often cannot determine if the best solution currently known is a local or a global optimum. In other words, we are not able to say if we should concentrate on refining our current optimum or if we should examine other parts of the search space instead. This problem, of course, goes hand in hand with the multimodality.

[32] see Equation 1.6 on page 15

Definition 19 (Multimodality). Multimodal functions have multiple (indistinguishable good) local optima and may also have multiple global optima [47, 48].

Definition 20 (Premature Convergence). A global 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 currently examined area *and* there exists such another region in the search space that contains a solution superior to the currently exploited one which could be found with reasonable effort [49, 50].

In Figure 1.9 we illustrate how an optimization algorithm prematurely converges. An optimization algorithm will pass several local optima in objective space before reaching a good result. If it gets stuck on such a intermediate solution and cannot proceed to over points in search space anymore, we speak of premature convergence.

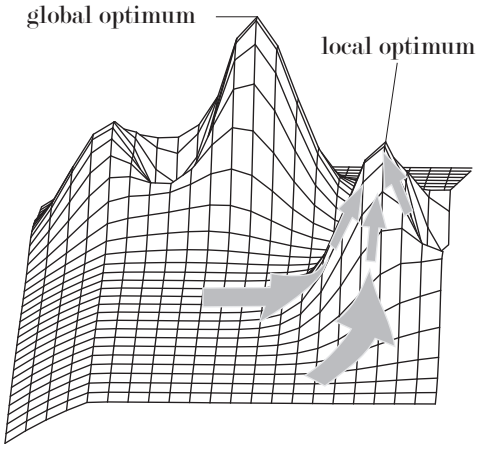


Fig. 1.9: Premature convergence in objective space.

There are many features and parameter settings of optimization algorithms that influence the convergence behavior. Self-adaptation is an important one of these factors that may have positive as well as negative effects on the convergence [51]. The operations that create new solutions from existing ones have also a very large impact [52, 53].

Exploration vs. Exploitation

All optimization algorithms have to trade-off between exploration and exploitation [54, 55, 56, 57, 58].

Definition 21 (Exploration). Exploration in terms of optimization means finding new points in the search space. Since we have got limited memory, this means most often to drop already evaluated individuals. Exploration is the only mean to find new and maybe better solutions but, on the other hand, leads to performance degradation at least until a new good solution is found – which is not guaranteed at all.

Definition 22 (Exploitation). Exploitation means trying to improve the currently known solution(s) by small changes which lead to new individuals very close to them. This way, performance improvements can be achieved. If another, maybe better solution exists in a distant area whatsoever, we will not be able to find it.

Almost all parts of optimization strategies can either be used for increasing exploitation or in favor for exploration. Mutation can, for example, improve an existing solution in small steps, being an exploitation operator. It can however also be implemented in a way that introduces much randomization into individuals and effectively being an exploration operation. For crossover basically goes the same.

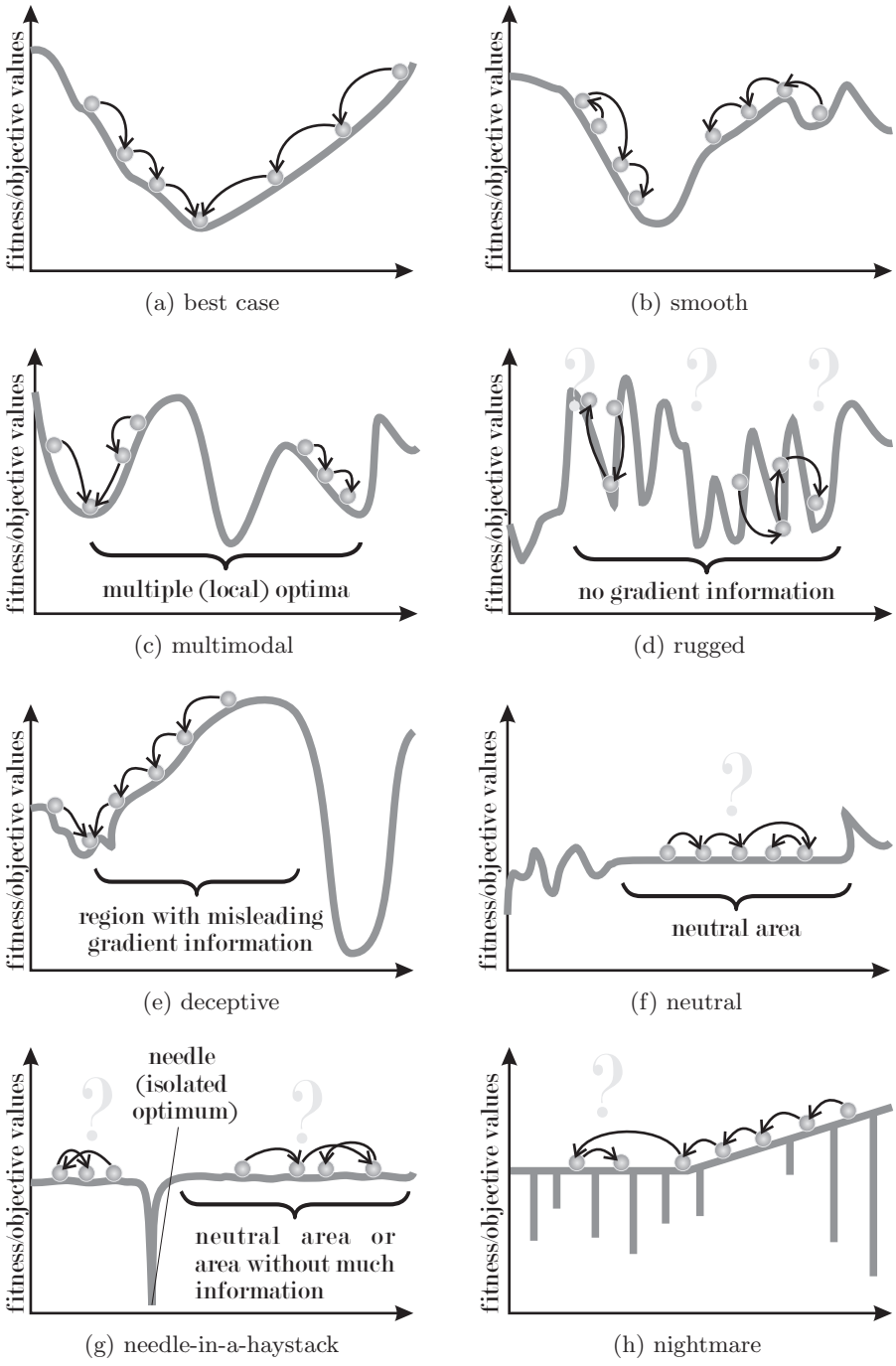
Selection operations[33] choose the set of individuals which will take part in reproduction. They can either return a small group of best individuals or a wide spread of existing solution candidates. The same goes for archive pruning techniques which truncate the set of known good solutions if it becomes too large.

While algorithms that favor exploitation have a fast convergence, they run a great risk of not finding the optimal solution and maybe get stuck at a local optimum. Algorithms that perform excessive exploration may find the global optimum but it will take them very long to do so. A good example for this dilemma is the simulated annealing algorithm discussed in Chapter 10 on page 231. It is often modified to a *simulated quenching* called form which favors exploitation but loses the guaranteed convergence to the optimum.

Exploitation and exploration are of course directly linked with diversity: exploration supports diversity whereas exploitation works against it. Diversity preservation is a major concern in optimization [59, 60, 61, 62, 63] because the loss of it can lead to premature convergence to a local optimum.

In Figure 1.10, we have sketched the different types of fitness landscapes which we are going to discuss in this section. The small bubbles represent solution candidates. An arrow from one bubble to another means that the second individual is offspring of the first one, created by a reproduction operation. The fitness/objective values are subject to minimization.

[33] see for example Section 2.4 on page 78



(a) best case (b) smooth

(c) multimodal (d) rugged

(e) deceptive (f) neutral

(g) needle-in-a-haystack (h) nightmare

Fig. 1.10: Different possible fitness landscapes.

1.4.2 Rugged Fitness Landscapes

We can draw further conclusions from the idea of descending a gradient. If we change a solution candidate x_{old} by a small amount ε and get $x_{new} = x_{old} \oplus \varepsilon$, we also expect the change in the fitness landscape to be small $f(x_{new}) \approx f(x_{old})$. This principle is called the *principle of strong causality* and formally stated in Section 3.7.1 on page 134. If it holds for many points in our solution space $\tilde{X}$, an optimization algorithm can climb along steady gradients and will be able to find good solutions. In fitness landscapes where small changes in the solution candidates often lead to large changes in the objective values as outlined in Figure 1.10d, it becomes harder to decide which region of the solution space to explore see. A small change to a very bad solution candidate may then lead to a new local optimum and the best solution candidate currently known may be surrounded directly by points that are inferior to all other tested individuals.

In general we can say the more rugged a fitness landscape is, the worse will optimizers perform [64, 65]. This does not necessarily mean that we cannot find good solutions, but it may take very long to do so.

As a measure for the ruggedness of a fitness landscape, the autocorrelation function as well as the correlation length of random walks can be used [66]. Here we borrow its definition from [67]: Given a random walk $(s_t, s_{t+1}, \ldots)$, the autocorrelation function ρ of an objective function f is the autocorrelation function of the time series $(f(s_t), f(s_{t+1}), \ldots)$.

$$\rho(k, f) = \frac{E\left[f(s_t)f(s_{t+k})\right] - E\left[f(s_t)\right]\left[f(s_{t+k})\right]}{var(f(s_t))} \tag{1.18}$$

where $E\left[f(s_t)\right]$ and $var(f(s_t))$ are the expected value and the variance of $f(s_t)$. 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 smoother is the landscape.

1.4.3 Deceptive Fitness Landscapes

Besides ruggedness, another annoying possible feature of objective functions is deceptiveness (or deceptivity). Figure 1.10e illustrates such a deceptive objective function that leads the (minimizing) search process away from the true optimum.

The term deceptiveness is mainly used in the Genetic Algorithms[34] community in the context of the Schema Theorem. There, schemas describe certain areas (hyperplanes) in the search space. If an optimization algorithm has discovered such an area with a better average fitness compared to other regions, it will logically focus on exploring this area. Thus, it is important that

[34] We are going to discuss Genetic Algorithms in Chapter 3 on page 117 and the Schema theorem in Section 3.6 on page 129.

these highly fit areas contain the true optimum at some given point of time in the search process. Objective functions where this is not the case are called deceptive [68, 69, 70].

1.4.4 Neutral Fitness Landscapes

Definition 23 (Neutrality). We consider the outcome of the application of a reproduction operation to a solution candidate as neutral if it yields no change in phenotypic or objective space (while it may lead to an offspring with a different genotype). The degree of neutrality defines the fraction of neutral results among all possible products of reproduction operations in an area of the search space. Areas is in the fitness landscape where this fraction is very high are considered as *neutral*.

The phenomenon of neutrality in fitness landscapes exists in natural as well as in artificial evolution [71, 72]. It has more facets than ruggedness [67, 73, 74] and may have positive as well as negative effects.

In this section, we will mainly focus on the possible negative aspects, without leaving possible benefits of some degrees of neutrality unmentioned. For all optimization algorithms, it is problematic when the best solution candidate currently found is situated one a plane of the fitness landscape and all neighbors have the same objective values. Then, there is no gradient information and thus no direction into which the optimization algorithm can progress. Furthermore, each reproduction cycle will yield identically well "optimal" solutions and the archive used to keep them will soon overflow.

Definition 24 (Evolvability). Evolvability[35] again can be defined in the contexts of biology and global optimization. A biological system is evolvable if its properties show heritable genetic variation and if natural selection can change these properties or if it can acquire new characteristics via genetic change [75, 76, 72, 71, 77]. The degree of evolvability in an optimization process in its current state defines how likely the reproduction operations will yield solution candidates with new fitness values [78, 79, 80].

The evolvability of neutral part of a fitness landscapes depends on the optimization algorithm used. It is especially low for hill climbers and similar algorithms, since the reproduction operations cannot provide fitness improvements (or even changes). The optimization process degenerates to a random walk in such planar areas, as illustrated in Figure 1.10f. One of the worst cases of fitness landscapes is the *needle-in-a-haystack* problemNeedle-In-A-Haystack sketched in Figure 1.10g, where the optimum occurs as *isolated* spike in a plane.

[35] `http://en.wikipedia.org/wiki/Evolvability` [accessed 2007-07-03]

A certain degree of neutrality can however also be beneficial [81, 82, 83]. Particularly for evolutionary algorithms and in form of redundancy of solution candidate representations, it may (or may not) provide ways for the optimizer to explore the problem space. More details on redundancy can be found in Section 3.7.3 on page 135. Generally, neutrality may have positive effects if it concerns only a subset of the properties of the parental solution candidate while allowing meaningful "modification" of the others (see Section 4.7.2 on page 183 for instance). If most or all possible changes will result in individuals with the same features, it will hinder the optimization algorithm's progress as discussed.

Generally we can state that, in spite of ruggedness which is always a bad for optimization algorithms, neutrality has many aspects that may further as well as hinder the process of finding good solutions.

1.4.5 Dynamically Changing Fitness Landscape

At least it is to be mentioned that there also exist fitness landscapes that change dynamically [84, 85, 86, 87]. The task of an optimization algorithm is then to provide solution candidates with momentarily high fitness for each point of time. Here we have the problem that an optimum in generation t will probably not be an optimum in generation $t + 1$ anymore.

Problems with dynamic characteristics can for example be tackled with special forms [88] of

- evolutionary algorithms [89, 90, 91, 92, 93, 94, 95],
- particle swarm optimizers [96, 97, 98, 99, 100],
- Differential Evolution [101, 102], and
- Ant Colony Optimization [103, 104]

The moving peaks benchmarks by Branke [87] and Morrison and De Jong [85] is a good example for dynamically changing fitness landscapes. You can find it discussed in Section 17.1.3 on page 276.

1.4.6 Overfitting

Definition 25 (Overfitting). Overfitting[36] identifies the emergence of an arbitrarily complicated model in a machine learning process in an effort to fit as much of the available sample data of a real system as possible [105]. A model m created with a finite set of sample data is considered to be overfitted if an alternative model m' if m has a smaller error on the training data but the error of m' is smaller if all possible (maybe even infinite many) system inputs are considered.

[36] `http://en.wikipedia.org/wiki/Overfitting` [accessed 2007-07-03]

This phenomenon can often be encountered in the field of artificial neural networks or in curve fitting. If we, for example, have n sample points (x_i, y_i), we can always construct a polynomial[37] of the degree $n-1$ that passes to all of them. Thus, we may have one hundred sample points which are results of the simple function $y = f(x) = x$ and fit polynomial of the degree 99 to them. The result may be a function that is only correct for exactly the 100 sample points and does not match to $f(x)$ for all other values of x. In Figure 1.11 we have illustrated this example.

The major problem that results from overfitted solutions is the loss of generality.

Definition 26 (Generality). A solution s of an optimization process is general if it is not only valid for the sample inputs $x_1, x_2, \ldots, x_n$ which were used during the optimization process in order to find it, but also for different inputs $\xi \neq x_i \; \forall \; 0 < i \leq n$ if such inputs ξ exist. A solution is also general if it is valid for all possible inputs.

There are two major reasons for overfitting:

1. To few data samples are available for learning. The resulting model in this case can be too trivial and may only work correctly on a small fraction of the possible inputs.
2. If we have noisy data samples, the learning algorithm may have learned a pseudo-model for the noise as part of its result. This model will, of course, not be generally valid since if the noise could be modeled exactly, it would not be noise but part of the correct description of the system.

An overfitted solution will not be able to produce valid results for inputs which differ from the training data used to create it. If it was valid for such different inputs, it would not be an overfitted but a perfectly fitting solution.

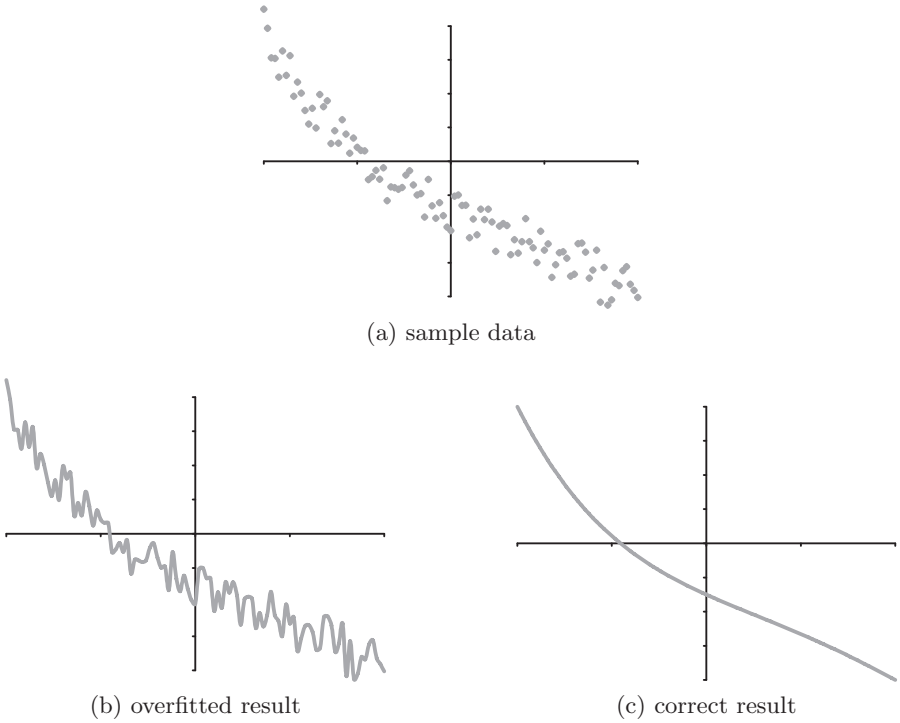
Overfitting is most often encountered in the area of neural networks [106, 107, 108, 109, 110] and curve fitting, but is a problem in many other fields of statistics too. We discuss overfitting in conjunction with genetic programming-based symbolic regression in Section 19.1 on page 329.

1.5 Modeling and Simulating

Whenever we want to solve a real problem, we first need to create some sort of abstraction from it.

Definition 27 (Model). A model[38] is a general abstraction from a real issue that allow us to reason and to deduct properties of the issue. Models often represent simplifications of the real-world issue they are describing by leaving away facts that probably only have minor impact on the conclusions drawn from the models.

[37] http://en.wikipedia.org/wiki/Polynomial [accessed 2007-07-03]
[38] http://en.wikipedia.org/wiki/Model_%28abstract%29 [accessed 2007-07-03]



(a) sample data

(b) overfitted result

(c) correct result

Fig. 1.11: Overfitting in curve fitting.

In the area of global optimization, we often need two types of abstractions:

1. Models of the potential solutions, like
   - a program in genetic programming, for example for the artificial ant problem[39],
   - a construction plan of a skyscraper,
   - a distributed algorithm represented as program for genetic programming,
   - a construction plan of a turbine,
   - a 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
   - a map on which the artificial ant will move which is driven by the evolved program,
   - the an abstraction from the environment in which the skyscraper will be built, with wind blowing from several directions,
   - a model of the network in which the evolved distributed algorithms can run,

[39] see Section 17.4 on page 284

- a physical model of air which blows through the turbine,
- the model of the energy source and other pins which will be attached to the circuit together with the possible voltages on these pins.

Models are just conceptual descriptions of reality. Only by bringing them to life in the form of simulations we can test possible solutions of a problem for their utility and behavior. If the solution candidates function well in the simulation, we can assume that they will behave also well in the real world.

Definition 28 (Simulation). A simulation[40] is the physical realization of a model. Whereas a model describes abstract connections between the properties of a real-world issue, a simulation realizes exactly these connections.

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

Models of the real world are often probabilistic. Consider a model of a computer network as done in Section 37.2.3 on page 603 where the communication is unsafe – messages may get lost with a certain probability. In a model of a skyscraper, the wind may blow with randomly changing strength from a random direction.

If we now test a distributed algorithm or a construction plan of a skyscraper with a simulation that realizes this behavior by utilizing new random numbers for each test, the outcomes of the single simulations will be different, even for exactly the same algorithm or plan. These results may vary totally unpredictable, making it hard to compare them. A highly fit but unlucky individual may be tested in a scenario where even the best solution candidate could only provide good results whereas a worse solution candidate could be lucky and be valuated with good results.

There exist two viable ways in representing probabilistic models in a way that maintains comparability of the results:

1. Repeat the simulations many times and compute the median of the evaluation results. These should be more or less stable and allow us to reason about the fitness of an individual.
2. Before performing any simulation, pre-define the scenarios exactly, i. e. compute all random numbers. In the case of testing a distributed algorithm, we would specify the topology, the message delay, the connection speeds and so on randomly. For the skyscraper tests, we define exactly how strong the wind will blow from which direction also randomly. These specifications are then used in all simulations time and again, so the outcomes will always be the same for same solution candidates. This way, we can still represent probabilistic models correctly but have comparable results. Of course, we need to create multiple scenarios and test all solution

[40] `http://en.wikipedia.org/wiki/Simulation` [accessed 2007-07-03]

candidates with each scenario to prevent overfitting (see Section 1.4.6 on page 27). An overfitted solution in this context would mean an individual that is produced by the learning the properties of all sample scenarios. The result of such a process will be one individual that is specialized only for exactly the one, tested configuration.

1.6 General Features of Global Optimization Algorithms

There are some common semantics and operations that are shared by most optimization algorithms. Many of them for example first create some starting values randomly and then refine them iteratively. During these steps, it is possible to loose a good solution again, so they preserve them in so-called optimal sets. Optimization algorithms will need to terminate at some point of time which is when their termination criterion is reached. In this section we define and discuss such general abstractions.

1.6.1 Iterations

Global optimization algorithms often iteratively evaluate solution candidates in order to approach the optima.

Definition 29 (Iteration). An iteration[41] 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 major loop. In the context of this book, an iterative optimization algorithm starts with the first step $t = 0$ while $t \in \mathbb{N}_0$ is the index of the current iteration and $t + 1$ is the next iteration step. Variables will sometimes be annotated with an index to emphasize their dependency to t: $X_t^\star$ or will be declared as function of time like $\varepsilon(t)$. In some optimization algorithms, iterations are referred to as *generations*. One example of an iterative algorithm is Algorithm 1.1 on the next page.

1.6.2 Termination Criterion

Definition 30 (Termination Criterion). When the termination criterion $terminationCriterion() \in \{\texttt{true}, \texttt{false}\}$ is met (i. e. is evaluated to `true`), the optimization process will halt and return its results.

Some possible termination criteria are [111, 112, 113, 114]:

[41] `http://en.wikipedia.org/wiki/Iteration` [accessed 2007-07-03]

- A maximum computation time specified in advance is exceeded. This is always the total time since the computational time of the single iterations in, for example evolutionary algorithms, may vary and is not known beforehand.
- A total number of generations/iterations is exhausted.
- A total number of solution candidates has been created and evaluated. These last two criteria often used in research because they allow us to compare the performance of different optimization measures for a certain problem.
- An optimization process may be stopped when no improvement could be detected for a specified number of iterations. Then, the process has converged to a (hopefully good) solution and will most probably not be able to make further progress.
- If we optimize something like a decision maker 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 training set is used to verify its results. We can compare the performance of our solution when fed with the training set to if fed with the test set. This comparison may help us detect when no further generalization will be made and when we should terminate the process because further optimization steps will only lead to overfitting.

An optimization process may use one or a combination of some of the criteria above 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
```

1.6.3 Minimization

Some algorithms are defined for single-objective optimization in their original form. Such an algorithm may be defined 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 1.3.5 on page 20, multi-objective optimization processes can be transformed into single-objective minimization processes. Therefore $x_1 \succ x_2 \Leftrightarrow c_F(x_1, x_2) < 0$.

1.7 The Optimal Set

Most multi-objective optimization algorithms return a set of optimal solutions $X^\star$ instead of a single individual $x^\star$. They keep internally track of the set of best solution candidates known. We use the prevalence-based definition for optimal sets as introduced on page 20 since it supersets all other definitions.

1.7.1 Updating the Optimal Set

Whenever a new individual is created, $X^\star$ may change. Possible, the new individual must be included in the optimal set or even prevails some of the solution candidates contained therein.

Definition 31 (*updateOptimalSet*). The *updateOptimalSet* function updates a set of optimal elements $X^\star_{old}$ with the knowledge of a new solution candidate x_{new}. It uses implicit knowledge of the prevalence function c_F.

$$X^\star_{new} = updateOptimalSet(X^\star_{old}, x_{new}) \; : \; \begin{array}{l} x_{new} \in \tilde{X} \wedge \\ X^\star_{new}, X^\star_{old} \subseteq \tilde{X} \wedge \\ X^\star_{new} \subseteq X_{old} \cup \{x_{new}\} \end{array} \quad (1.19)$$

We define two equivalent approaches in Algorithm 1.2 and Algorithm 1.3 which perform the necessary operations. Algorithm 1.2 creates a new, empty optimal set and successively inserts optimal elements whereas Algorithm 1.3 removes all elements that are prevailed by a new individual x_{new} from the old optimal set $X^\star_{old}$.

1.7.2 Obtaining Optimal Elements

If we already have an optimal set, it can simple be updated with new individuals as we have just discussed in Section 1.7.1. An optimization algorithm however may not necessarily maintain such a set. When it is terminates, it the just extracts all optimal elements from its current population, thus also obtaining an optimal set.

Algorithm 1.2: $X^\star_{new} = updateOptimalSet(X^\star_{old}, x_{new})$

Input: $X^\star_{old}$ the optimal set as known before the creation of x_{new}
Input: x_{new} a new solution candidate to be checked
Input: Implicit: c_F the comparator function as declared in the definition of prevalence on page 20, a dominance/pareto based comparator is used as default
Output: $X^\star_{new}$ the optimal set updated with the knowledge of x_{new}

```
1  begin
2      X*_new ⟵ ∅
3      foreach x* ∈ X*_old do
4          if c_F(x_new, x*) > 0 then // x_new ≻ x* (dominated)
5              X*_new ⟵ X*_new ∪ {x*}
6              if c_F(x_new, x*) ≥ 0 then // x* ≻ x_new
7                  return X*_old
8      X*_new ⟵ X*_new ∪ {x_new}
9      return X*_new
10 end
```

Algorithm 1.3: $X^\star_{new} = updateOptimalSet(X^\star_{old}, x_{new})$ (2nd version)

Input: $X^\star_{old}$ the optimal set as known before the creation of x_{new}
Input: x_{new} a new solution candidate to be checked
Input: Implicit: c_F the comparator function as declared in the definition of prevalence on page 20, a dominance/pareto based comparator is used as default
Output: $X^\star_{new}$ the optimal set updated with the knowledge of x_{new}

```
1  begin
2      X*_new ⟵ X*_old
3      foreach x* ∈ X*_new do
4          if c_F(x_new, x*) < 0 then // x_new ≻ x*
5              X*_new ⟵ X*_new \ {x*}
6          else
7              if c_F(x_new, x*) > 0 then // x* ≻ x_new
8                  return X*_old
9      X*_new ⟵ X*_new ∪ {x_new}
10     return X*_new
11 end
```

Definition 32 (*extractOptimalSet*). The *extractOptimalSet* function extracts a set of optimal (non-prevailed) individuals $X^\star_{new}$ from any given list of individuals X_{any}.

$$\begin{aligned} X^\star = extractOptimalSet(X_{any}, c_F) \ : \quad & X^\star \subseteq \tilde{X} \wedge \\ & \forall x^\star \in X^\star \Rightarrow \exists\, i \in 0 \ldots |X_{any}| : x^\star = X_{any}[i] \wedge \\ & \forall x^\star \in X^\star \Rightarrow \nexists x \in X_{any} : c_F(x, x^\star) < 0 \end{aligned} \quad (1.20)$$

Algorithm 1.4 demonstrates how the extraction of an optimal set can be performed. Obviously, this approach could also be used for updating:

$$updateOptimalSet(X^\star_{old}, x_{new}) \equiv extractOptimalSet(setToList(X^\star_{old}) \cup x_{new}) \quad (1.21)$$

Algorithm 1.4: $X^\star = extractOptimalSet(X_{any})$

Input: X_{any} the list to extract the optimal individuals from
Input: x_{any}, x_{chk} solution candidates tested for supremacy
Input: Impricit: c_F the comparator function as declared in the definition of prevalence on page 20, a dominance/pareto based comparator is used as default
Output: $X^\star$ the optimal subset extracted from X_{any}

```
1  begin
2      X* ⟵ X_any
3      i ⟵ |X*| − 1
4      while i > 0 do
5          j ⟵ i − 1
6          while j ≥ 0 do
7              if X*[i] ≻ X*[j] then
8                  X* ⟵ deleteListItem(X*, j)
9                  i ⟵ i − 1
10             else if X*[j] ≻ X*[i] then
11                 X* ⟵ deleteListItem(X*, i)
12                 j ⟵ −1
13             j ⟵ j − 1
14         i ⟵ i − 1
15     return listToSet(X*)
16 end
```

1.7.3 Pruning the Optimal Set

As already mentioned, there may be very many if not infinite many optimal solutions for a problem. On the other hand, the optimal set $X^\star$ computed

by the optimization algorithms 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 [115, 116, 117]. If the number of optimal solutions is large but finite, unrestricted archives for optimal solution candidates can perform better because any reduction of the number of individuals stored leads to a loss of generality [118]. Pruning operations (that try to minimize this loss) will usually base on clustering algorithms [115, 119] introduced in Section 36.3 or on Pareto-ranking, but in general we can define:

Definition 33 (*pruneOptimalSet*). The pruning operation *pruneOptimalSet* reduces the size of an optimal set $X^\star$ to fit an implicitly given upper boundary k_t.

$$X^\star_{new} = pruneOptimalSet(X^\star_{old}) \ : \ X^\star_{old}, X^\star_{new} \subseteq \tilde{X} \tag{1.22}$$

$$|X^\star_{new}| \leq k_t, \ k_t \in \mathbb{N} \tag{1.23}$$

$$X^\star_{new} \subseteq X^\star_{old} \tag{1.24}$$

Pruning via Clustering

Algorithm 1.5 uses clustering [115, 119] to provide the functionality specified in this definition. Basicall, any given clustering algorithm could be used as replacement for *cluster* – see Chapter 36 for more information on clustering.

Algorithm 1.5: $X^\star_{new} = pruneOptimalSet_c(X^\star_{old})$

Input: $X^\star_{old}$ the optimal set to be pruned
Input: Implicit: k_t 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: $X^\star_{new}$ the pruned optimal set

```
1 begin
2 |  B ⟵ cluster(X*_old)
3 |  X*_new ⟵ ∅
4 |  foreach b ∈ B do  X*_new ⟵ X*_new ∪ nucleus(b)
5 |  return X*_new
6 end
```

Adaptive Grid Archiving

An algorithm for adaptive grid archiving has been introduced for the evolutionary algorithm PAES (see Section 2.6.8 on page 107) in [120]. We work

directly on the $n = |F|$ objective values of the individuals and hence, can treat them as n-dimensional vectors. We view the n-dimensional space as a grid, creating d divisions in each dimension. The span of each dimension is defined by the minimum and maximum objective values of the individuals in that dimension. The individuals with the minimum/maximum values are preserved always. Therefore, it is not possible to define maximum optimal set sizes k which are smaller then $2n$. If individuals need to be removed from the set because it became too large, we remove individuals from regions which are the most crowded.

The original sources do not contain an exact description of the algorithm, so we introduce a more or less trivial definition in Algorithm 1.6 on the following page and Algorithm 1.7 on page 39. The preservation of border individuals is achieved in *agaDivide* by putting them into separate, unique-boxes which have an inhabitant count of -1, disabling their later disposal by the pruning algorithm. The unique hyper-boxes are created by adding an additional (unique) row to their box coordinate vector in line 27. The *agaDivide* algorithm is however also used by some selection schemes, namely for the PESA-based selection (see Section 2.4.11 and Section 2.4.12). Therefore, this extra row needs to be removed using the *agaNormalize* method presented in Algorithm 1.8 on page 40. Again, the algorithms presented are not necessarily optimal but simple and correct.

Algorithm 1.6: $(X_l, lst, cnt) = agaDivide(X^\star_{old}, d)$

Input: $X^\star_{old}$ the optimal set to be pruned
Input: Implicit: n the count of objective functions $f \in F$
Input: d the count of divisions to be performed per dimension
Data: i, j, k counter variables
Data: min, max, mul temporary stores
Output: (X_l, lst, cnt) a tuple containing the list representation X_l of $X^\star_{old}$, a list lst assigning grid coordinates to the elements of X_l and a list cnt containing the count of elements in the grid locations defined in lst

```
1  begin
2      min ⟵ createList(n, ∞)
3      max ⟵ createList(n, −∞)
4      i ⟵ n
5      while i > 0 do
6          min[i − 1] ⟵ min{f_i(x*) : x* ∈ X*_old}
7          max[i − 1] ⟵ max{f_i(x*) : x* ∈ X*_old}
8          i ⟵ i − 1
9      mul ⟵ createList(n, 0)
10     i ⟵ n − 1
11     while i ≥ 0 do
12         if max[i] ≠ min[i] then mul[i] ⟵ d / (max[i] − min[i])
13         else max[i] ⟵ max[i] + 1, min[i] ⟵ min[i] − 1
14         i ⟵ i − 1
15     X_l ⟵ setToList(X*_old)
16     lst ⟵ createList(|X_l|, ∅)
17     cnt ⟵ createList(|X_l|, 1)
18     i ⟵ |X_l| − 1
19     k ⟵ −1
20     while i ≥ 0 do
21         j ⟵ n
22         lst[i] ⟵ createList(n, 0)
23         while j > 0 do
24             if (f_j(X_l[i]) ≤ min[j]) ∨ (f_j(X_l[i]) ≥ max[j]) then
25                 lst[i] = listAdd(lst[i], ⟵ k)
26                 cnt[i] ⟵ −1
27                 k ⟵ k − 1
28             lst[i][j − 1] ⟵ (f_j(X_l[i]) − min[j]) ∗ mul[j]
29             j ⟵ j − 1
30         if cnt[i] > 0 then
31             j ⟵ i + 1
32             while j < |X_l| do
33                 if lst[i] = lst[j] then
34                     cnt[i] ⟵ cnt[i] + 1
35                     cnt[j] ⟵ cnt[j] + 1
36                 j ⟵ j + 1
37         i ⟵ i − 1
38     return (X_l, lst, cnt)
39 end
```

Algorithm 1.7: $X^{\star}_{new} = pruneOptimalSet_{aga}(X^{\star}_{old})$

Input: $X^{\star}_{old}$ the optimal set to be pruned
Input: Implicit: n the count of objective functions $f \in F$
Input: Implicit: k the maximum size allowed for the optimal set ($k \geq 2n$)
Input: Implicit: d the count of divisions to be performed per dimension
Data: i a counter variable
Data: X_l the list representation of $X^{\star}_{old}$
Data: lst a list assigning grid coordinates to the elements of X_l
Data: cnt containing the count of elements in the grid locations defined in lst
Output: $X^{\star}_{new}$ the pruned optimal set

```
1  begin
2      if |X*_old| ≤ k then return X*_old
3      (X_l, lst, cnt) ⟵ agaDivide(X*_old, d)  while |X_l| > k do
4          idx ⟵ 0
5          i ⟵ |X_l| − 1
6          while i > 0 do
7              if cnt[i] > cnt[idx] then idx ⟵ i
8          i ⟵ |X_l| − 1
9          while i ≥ 0 do
10             if lst[i] = lst[idx] then cnt[i] ⟵ cnt[i] − 1
11         X_l ⟵ deleteListItem(X_l, idx)
12         cnt ⟵ deleteListItem(cnt, idx)
13         lst ⟵ deleteListItem(lst, idx)
14     return listToSet(X_l)
15 end
```

Algorithm 1.8: $(lst, cnt) = agaNormalize(lst, cnt)$

Input: lst a list assigning grid coordinates to the elements of X_l
Input: cnt containing the count of elements in the grid locations defined in lst
Input: Implicit: n the count of objective functions $f \in F$
Data: i, j, k counter variables
Output: (lst, cnt) the corrected inputs where border elements are now contained in the right boxes

```
1  begin
2      i ⟵ |lst|
3      while i ≥ 0 do
4          if |list[i]| > n then
5              lst[i] ⟵ deleteListItem(lst[i], n)
6              j ⟵ |lst|
7              while j ≥ 0 do
8                  if (lst[j] = lst[i]) ∧ (cnt[j] > 0) then
9                      cnt[i] ⟵ cnt[j]
10                     j ⟵ −1
11                 j ⟵ j − 1
12             if cnt[i] < 0 then  cnt[i] ⟵ 1
13         i ⟵ i − 1
14     return (lst, cnt)
15 end
```

1.8 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.

1.8.1 Areas Of Application

Some example areas of application of global optimization algorithms are:

Application	References
chemistry, chemical engineering, and biochemistry	[121, 122, 123, 124]
constraint satisfaction problems	[6]
system design	[31]
multiple criteria decision making	[28, 26]
biology and biomedicine	[123]
structural optimization and design	[125, 123]
economics and finance	[126, 123, 127]
engineering design and process design	[126, 122, 124, 123]
parameter estimation	[124]
mathematical problems	[128]
optical design and engineering	[129, 130]
water resource management	[131]

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.

1.8.2 Conferences, Workshops, etc.

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

HIS: International Conference on Hybrid Intelligent Systems
`http://www.softcomputing.net/hybrid.html` [accessed 2007-09-01]
History: 2007: Kaiserslautern, Germany, see [132]
2006: Auckland, New Zealand, see [133]
2005: Rio de Janeiro, Brazil, see [134]
2004: Kitakyushu, Japan, see [135]

2003: Melbourne, Australia, see [136]
2002: Santiago, Chile, see [137]
2001: Adelaide, Australia, see [138]

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 [139]
2006: Chania, Crete, Greece, see [140]
2004: Whistler, British Columbia, Canada, see [141]
2002: Semmering, Austria, see [142]
2000: Ankara, Turkey, see [143]
1998: Charlottesville, Virginia, USA, see [28]
1997: Cape Town, South Africa, see [144]
1995: Hagen, Germany, see [145]
1994: Coimbra, Portugal, see [146]
1992: Taipei, Taiwan, see [147]
1990: Fairfax, USA, see [148]
1988: Manchester, UK, see [149]
1986: Kyoto, Japan, see [150]
1984: Cleveland, Ohio, USA, see [151]
1982: Mons, Belgium, see [152]
1980: Newark, Delaware, USA, see [153]
1979: Königswinter, Germany, see [154]
1977: Buffalo, New York, USA, see [155]
1975: Jouy-en-Josas, France, see [156]

Mendel: International Conference on Soft Computing
`http://mendel-conference.org/` [accessed 2007-09-09]
History: 2007: Prague, Czech Republic, see [157]
2006: Brno, Czech Republic, see [158]
2005: Brno, Czech Republic, see [159]
2004: Brno, Czech Republic, see [160]
2003: Brno, Czech Republic, see [161]
2002: Brno, Czech Republic, see [162]
2001: Brno, Czech Republic, see [163]
2000: Brno, Czech Republic, see [164]
1999: Brno, Czech Republic, see [165]
1998: Brno, Czech Republic, see [166]

1997: Brno, Czech Republic, see [167]
1996: Brno, Czech Republic, see [168]
1995: Brno, Czech Republic, see [169]

MIC: Metaheuristics International Conference
History: 2007: Montreal, Canada, see [170]
2005: Vienna, Austria, see [171]
2003: Kyoto, Japan, see [172]
2001: Porto, Portugal, see [173]
1999: Angra dos Reis, Brazil, see [174]
1997: Sophia Antipolis, France, see [175]
1995: Breckenridge, Colorado, USA, see [176]

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.

1.8.3 Journals

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

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: bi-monthly, 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 bi-weekly, 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]

1.8.4 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. Includes links, publications, and software.

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

1.8.5 Books

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

Pardalos, Thoai, and Horst: *Introduction to Global Optimization* (see [7])
Floudas and Pardalos: *Frontiers in Global Optimization* (see [123])
Dzemyda, Saltenis, and Zilinskas: *Stochastic and Global Optimization* (see [126])
Gandibleux, Sevaux, Sörensen, and T'kindt: *Metaheuristics for Multiobjective Optimisation* (see [177])
Floudas: *Deterministic Global Optimization: Theory, Methods and Applications* (see [124])
Chankong and Haimes: *Multiobjective Decision Making Theory and Methodology* (see [26])
Steuer: *Multiple Criteria Optimization: Theory, Computation and Application* (see [27])
Haimes, Hall, and Freedman: *Multiobjective Optimization in Water Resource Systems* (see [131])
Charnes and Cooper: *Management Models and Industrial Applications of Linear Programming* (see [35])
Corne, Dorigo and Glover: *New Ideas in Optimisation* (see [178])

2

Evolutionary Algorithms

2.1 Introduction

Definition 34 (Evolutionary Algorithm). Evolutionary algorithms[1] (EA) [179, 180, 181] are generic, population-based meta-heuristic optimization algorithms that use biology-inspired mechanisms like mutation, crossover, natural selection and survival of the fittest.

2.1.1 The Basic Principles from Nature

In 1859, Charles Darwin published his book "On the Origin of Species"[2] [5] in which he first 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 [5, 182, 2]:

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, 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. Many 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.

[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]

9. Individuals that survive and reproduce will likely pass on their traits to their offspring.
10. A species will slowly change and adapt more and more to a given environment during this process which may finally result in even new species.

Evolutionary algorithms abstract from this biological process and also introduce a change in semantics by being *goal-driven* [183]. The solution candidates of a certain problem play the role of individuals. The fitness of them is rated according to objective functions which are subject to optimization and drive the evolution into specific directions.

The advantage of evolutionary algorithms compared to other optimization methods is that they make only few assumptions about the underlying fitness landscape and therefore perform consistently well in many different problem categories.

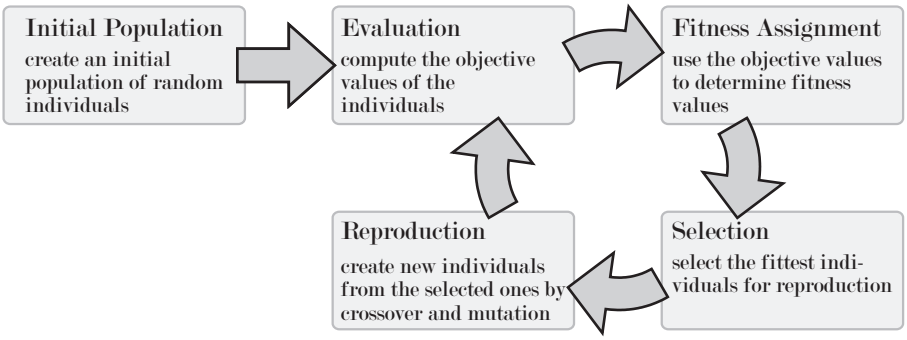


Fig. 2.1: The basic cycle of evolutionary algorithms.

As already mentioned, the basic idea behind genetic and evolutionary algorithms is copying the process of Darwinian evolution in order to find solutions for hard problems. Now let us look a bit deeper into the basic cycle of artificial evolution illustrated in Figure 2.1 and how its single steps correspond with its natural role model.

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

Definition 35 (MOEA). A multi-objective evolutionary algorithm (MOEA) is able to perform an optimization of multiple criteria on the basis of artificial evolution [184, 21, 185, 23, 20, 22, 186].

All evolutionary algorithms proceed in principle according to the following scheme:

1. Initially, a population of individuals with a totally random genome is created.
2. All individuals of the population are tested. This evaluation may incorporate complicated simulation and calculations.
3. With the tests, we have determined the utility of the different features of the solution candidates and can now assign a fitness value to each of them.
4. A subsequent selection process filters out the individuals with low fitness and allows those with good fitness to enter the mating pool with a higher probability.
5. In the reproduction phase, offspring is created by varying or combining these solution candidates and integrated into the population.
6. If a *terminationCriterion* is met, the evolution stops here. Otherwise, it continues at step 2.

At the beginning of the evolution, there exists no idea what is good or what is bad. Basically, only some random genes are coupled together as initial population. I think, back in the Eoarchean[3], the earth age 3.8 billion years ago where most probably the first single-celled life occurred on earth, it was probably the same.

At the beginning of the evolutionary cycle, nature instantiates each genotype in form of a phenotype – a living organism, for example a fish. The survival of the genes of our fish depend on how good it performs in the ocean, in other words, how fit it is. This fitness however is not only determined by one single feature of the phenotype like its size. Although a bigger fish will have better chances to survive, size alone does not help if it is too slow to catch any prey. Also its energy consumption should be low so it does not need to eat all the time. Sharp teeth would be good, and colors that blend into the environment so it cannot be seen to easily by sharks. But, uh, wait a second, if its camouflage is too good, how will it find potential mating partners? And if it is really big, it will also have a higher energy consumption. 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 genetic algorithms[4] this is exactly the same. For each problem that we want to solve, we can specify multiple so-called objective functions. An objective function represents one feature that we are interested in. Let us assume that we want to evolve a car (a pretty weird assumption,

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

[4] Genetic algorithms, a subclass of evolutionary algorithms, are discussed in Chapter 3 on page 117.

but let's stick with it). The genotype would be the construction plan and the phenotype the real car, or at least a simulation of it. One objective function 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. Furthermore, *cheap*, *fast* and a *cool lookout* would be good. So that is five objective functions from which for example the second and the fourth are contradictory.

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. If one fish can beat another one in all categories, i.e. is bigger, stronger, smarter, and so on, we can clearly consider it as fitter 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 maybe have the same probability to reproduce and hence, are not directly comparable[5]. Well, Ok, we cannot decide if a fish with a clever behavioral pattern is worse or better than a really strong one. 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. One of the most poplar methods here is called *Pareto ranking*[7]. It does exactly what we've just discussed: It first selects the individuals that are beaten by no one (we call this non-dominated set) and assigns a good (scalar) fitness value to them. Then it looks at the rest of the population and takes those which are not beaten by the remaining individuals and gives them a slightly worse fitness value - 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 decease. The fitness only is some sort of probability of survival. The process of selection is always stochastic, without guarantees - even a fish that is small and slow, lacking any sophisticated behavior, might survive and could produce even more offspring than a highly fit one.

The selection in evolutionary algorithms works in exactly the same way. The oldest selection scheme is called *Roulette wheel*[8]. The higher the fitness value of an individual the high is its chance to reproduce in the original version

[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 1.3.2 on page 14 and elaborations on Pareto ranking can be found in Algorithm 2.5.

[8] The roulette wheel selection algorithm is introduced in Section 2.4.5 on page 84.

of this selection algorithm. Whenever we need an individual for reproduction, we draw one from the population in way that each solution candidate is selected with a probability proportional to its fitness.

Last but not least, there is the reproduction phase. Fish reproduce sexually. When a female fish and a male fish will mate, their genes will be recombined by crossover. Mutations may further take place which, most often, only slightly affect the characteristics of resulting larva [187]. 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 of their parents and perform even better than those of the current generation.

In genetic algorithms, we do not have such a thing as "gender". Each individual can potentially mate with each other one. In the car example this would mean that we take the engine of one car and place it car body of another one – first in the construction plan, of course. Also, we could alter one feature, 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 like to be bought by the customer is good. If we iteratively perform the presented process again and again, there is a high probability that the solutions finally found will be close to optimal.

At this point it should be mentioned that the direct reference to Darwinian evolution in evolutionary and genetic algorithms is somehow controversial. [188] for example points out that "neither GA [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 copied 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: A species is a class of organisms which are very similar in many aspects such as appearance, physiology, and genetics according to [2]. In principle, there is some elbowroom for us where we well can consider even different solutions to a single problem in GA as members of a different species – for example if a crossover/recombination of their genomes cannot produce another valid solution candidate. So the personal opinion of the author (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.

Basic Evolutionary Algorithms

From 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. All EAs are variations and extensions of Algorithm 2.1 which relies on functions or prototypes that we will introduce step by step.

- *createPop* (see Algorithm 2.34 in Section 2.5 on page 99) produces an initial, randomized population.
- *terminationCriterion* checks whether the evolutionary algorithm should terminate or continue, see Section 1.6 on page 31.
- *updateOptimalSet* checks if a new individual can be included in the optimal, and, if so, if it may lead to other individuals being removed from the optimal set. See Section 1.7.1 on page 33 for details.
- 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 1.7.3 on page 35.
- Most evolutionary algorithms assign a scalar fitness to each individual by comparing its objective values to other individuals in the population or optimal set. Such a fitness assignment process ($assignFitness$, see Section 2.3 on page 65) also provides means to preserve diversity.
- Selection algorithms (*select*, see Section 2.4 on page 78) chose those individuals which can reproduce from the population and/or optimal set. They again perform mainly exploitation but defined in a diversity-preserving manner.
- 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 99.

Algorithm 2.1: $X^\star = simpleEA(c_F)$

Input: c_F the comparator function which allows us to compare the fitness of two solution candidates, used by *updateOptimalSet*
Input: Implicit: p the population size
Data: X_{pop} the population
Data: X_{mp} the mating pool
Output: $X^\star \subseteq \tilde{X}$ the set of the best elements found

```
1 begin
2     X_pop ⟵ createPop(p)
3     while ¬terminationCriterion() do
4         assignFitness(X_pop, ∅)
5         X_mp ⟵ select(X_pop, p)
6         X_pop ⟵ reproducePop(X_mp, p)
7     return extractOptimalSet(X_pop)
8 end
```

2.1.2 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 discuss these family members in the following chapters in more detail and outline here only their general meaning.

- **Genetic Algorithms** (GAs) are introduced in Chapter 3 on page 117. GAs subsume all evolutionary algorithms that use strings, arrays of fixed data types, as genomes. Most commonly, the data type is boolean and the algorithm searches in the space of bit strings.
- The special case, so to say, of Genetic Algorithms, where the genotypes are strings of real numbers, is called **Evolution Strategy** (ES, see Chapter 5 on page 203).
- For **Genetic Programming** (GP), which will be elaborated on in Chapter 4 on page 139), we can provide two definitions: On one hand, GP includes all evolutionary that grow programs, algorithms, and these alike. On the other hand, also all EAs that evolve tree-shaped individuals are instances of Genetic Programming.
- **Learning Classifier Systems** (LCS), discussed in Chapter 7 on page 211, 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.
- **Evolutionary Programming** (EP, see Chapter 6 on page 209) is an evolutionary approach that treats the instances of the genome as different species rather than as individuals. It has mainly merged into GP and the other evolutionary algorithms.

Algorithmic Properties of Evolutionary Algorithms

In the previous section, we have classified different evolutionary algorithms according to their semantics, the type of their search spaces (which includes how their reproduction operations work). These different approaches all use the same scheme which has been introduced in Algorithm 2.1. Improvements of this basic scheme will be valid for all members of the EA family. Because of their generality, we term algorithms that provide such improvements simple as "evolutionary algorithms".

In this chapter, we outline the major building blocks of many of the most efficient evolutionary algorithms [189] which have been developed until today. These EAs can be distinguished in many ways and some of their distinctive features are:

- The population size or the number of populations used.
- The method of selecting the individuals for reproduction.

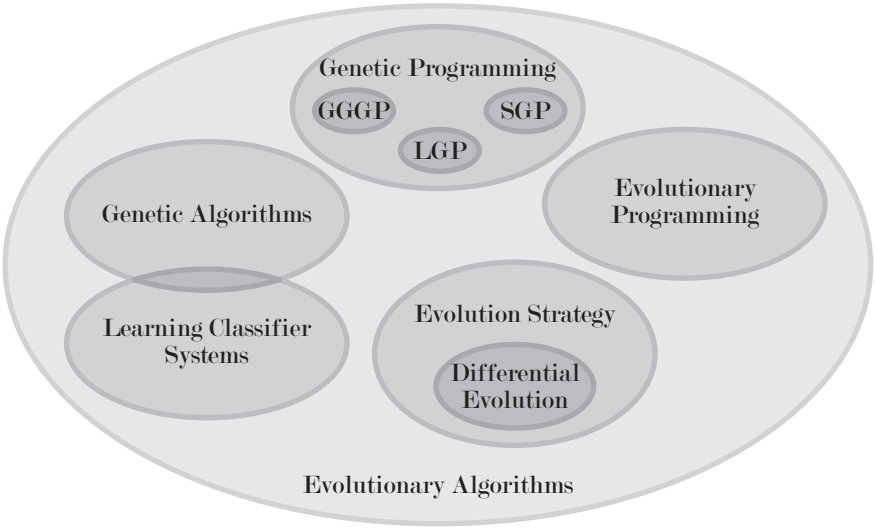


Fig. 2.2: The family of evolutionary algorithms.

- The representations of the individuals – they may be represented as is or in the form of chromosomes of a given genotype.
- The way the offspring is included into the population(s). (replacement-operators)

2.1.3 Populations in Evolutionary Algorithms

One distinctive feature of different evolutionary algorithms in the way populations are treated, especially how the population of the next iteration is selected from the current population and its offspring.

If the next population is entirely formed by the offspring of the current one, we speak of *extinctive selection* [190, 191]. Extinctive selection can be compared with ecosystems of small protozoa which reproduce in a fissiparous manner. In that 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 36 (Generational). In evolutionary algorithms that are *generational* [192], 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 [193]. In left extinctive selections, the best individuals are not allowed to reproduce in order to prevent premature convergence of the optimization process. The worst individuals are not permitted to breed in right extinctive selection schemes in order to reduce the selective pressure because they would 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 [194, 195, 196, 183]. 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 strategy discussed in Chapter 5 on page 203, there exists a notation which also can be used describe the generation transition in evolutionary algorithms [197, 198, 199, 194].

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

Extinctive selection patterns are denoted as (μ, λ)-strategies and will create $\lambda > \mu$ child individuals from the μ patterns and only keep the μ best offspring while discarding the μ parents and the $\lambda - \mu$ worst children.

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

Steady state evolutionary algorithms [200, 201, 202, 203, 204, 205], abbreviated SSEA, are preservative evolutionary algorithm values of λ that are relatively low in comparison with μ. Usually, λ is chosen in a way that a recombination operator (which in this context produces two offspring) is applied exactly once per generation. Hence, we have a $(\mu + 2)$ evolution, since is equal to the number of offspring $\lambda = 2$ and we have a $(\mu + \lambda)$ selection. Although steady state evolutionary algorithms are often observed to produce better results than generational EAs [203, 206, 207], there is also research that indicates that they are not generally superior [208].

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 selected with a certain selection algorithm. Most of the selection algorithms tend to pick the better individuals, but most of them are also randomized, and hence may also choose worse individuals with a certain, lower probability.

Definition 37 (Elitism). An elitist evolutionary algorithm [209, 210, 184] ensures that at least one copy of the best individual(s) of the current generation is passed on to the next generation. The main advantage of elitism is that its convergence is guaranteed, meaning that if the global optimum is 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 sort of preservative strategy which is often reached by using a secondary population only containing the non-prevailed individuals. This population normally does not take part in the genetic operations directly and is updated only at the end of the iterations. Such an archive-based elitism can be combined with generational and preservative evolutionary algorithms as well.

Algorithm 2.2 specifies the basic scheme of elitist evolutionary algorithms. Note that it only differs in line 5 from Algorithm 2.1.

Algorithm 2.2: $X^\star = elitistEA(c_F)$

Input: c_F the comparator function which allows us to compare the fitness of two solution candidates, used by *updateOptimalSet*
Input: Implicit: p the population size
Data: X_{pop} the population
Data: X_{mp} the mating pool
Output: $X^\star \subseteq \tilde{X}$ the set of the best elements found

```
1  begin
2      X* ⟵ ∅
3      X_pop ⟵ createPop(p)
4      while ¬terminationCriterion() do
5          foreach x ∈ X_pop do  X* ⟵ updateOptimalSet(X*, x)
6          X* ⟵ pruneOptimalSet(X*)
7          assignFitness(X_pop, X*)
8          X_mp ⟵ select(X_pop ∪ X*, p)
9          X_pop ⟵ reproducePop(X_mp, p)
10     return X*
11 end
```

2.1.4 Forma Analysis

In his seminal 1975 work, Holland stated the schema theorem[9] which describes how different characteristics of genotypes will be propagated during the evolutionary process in genetic algorithms [211, 209, 54]. Here we are going to discuss this issue from the more general perspective of *forma analysis* from [212] as introduced by Radcliffe[10] and Surry [213, 214, 215, 216, 217, 218]. Forma analysis originally is focused on Genetic Algorithms only. We can (and will) extend many of its basic definitions to other evolutionary algorithms like Genetic Programming easily.

[9] The Schema theorem is discussed in Section 3.6 on page 129.

[10] Radcliffe also has maintained a website about forma analysis. Although now most of its links are dead, maybe it's worth a visit: `http://users.breathe.com/njr/formaPapers.html` [accessed 2007-07-29].

Basically, every solution candidate in the search space of an evolutionary algorithm is characterized by its properties. A property p_1 of a given formula $f : \mathbb{R} \mapsto \mathbb{R}$ in symbolic regression[11] can be whether or not it contains the mathematical expression $x + 1$. This is a rather structural or "genotypic" property. We can also declare a "phenotypic" property p_2 that defines if $|f(0) - 1| \leq 0.1$ holds, i. e. if the result of f is close to a value 1 for a specified input $x = 0$. If we would try to solve a graph-coloring problem for instance, a property $p_3 \in \{black, white, gray\}$ could denote the color of a specific vertex q as illustrated in Figure 2.3.

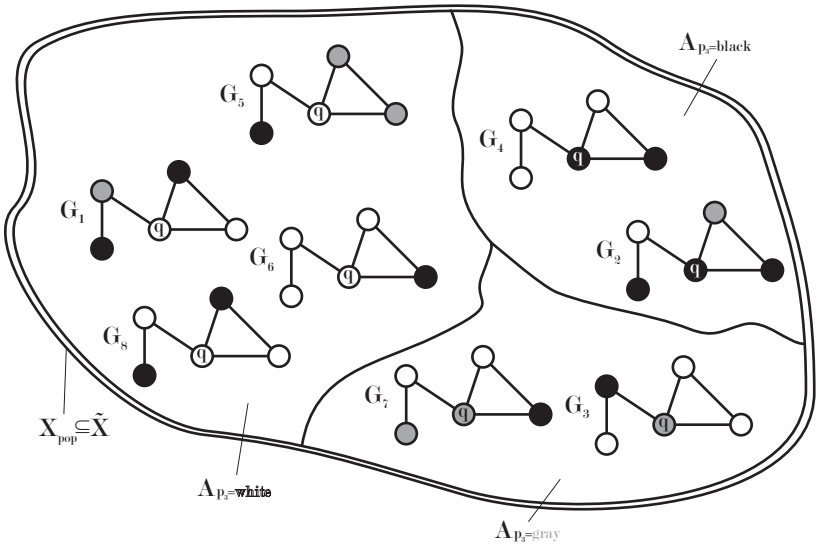


Fig. 2.3: An graph coloring-based example for properties and formae.

In general we can imagine the properties p_i to be some sort of functions that map the individuals to property values. p_1 and p_2 would then both map the space of mathematical functions to the set $\{\texttt{true}, \texttt{false}\}$ whereas p_3 maps the space of all possible colorings for the given graph to the set $\{white, gray, black\}$. On the basis of the properties p_i we can define equivalence relations[12] $\sim_{p_i}$:

$$x \sim_{p_i} y \Rightarrow p_i(x) = p_i(y) \tag{2.1}$$

Obviously, for each two individuals x and y, either $x \sim_{p_i} y$ or $x \not\sim_{p_i} y$ holds. These relations divide the search space into equivalence classes $A_{p_i=v}$.

[11] More information on symbolic regression can be found in Section 19.1 on page 329.
[12] See the definition of equivalence classes in Section 34.6.2 on page 510.

Definition 38 (Forma). An equivalence class $A_{p_i=v}$ that contains all the individuals sharing the same characteristic v in terms of the property p_i is called a *forma* [216] or *predicate* [219].

$$A_{p_i=v} = \{\forall\, p_i(x) = v\} \tag{2.2}$$

$$\forall\, x, y \in A_{p_i=v} \Rightarrow x \sim_{p_i} y \tag{2.3}$$

The number of formae induced by a property, i.e. the number of its different characteristics, is called its *precision* [216]. The precision of p_1 and p_2 is 2, of p_3 it is 3. We can define another property $p_4 \equiv f(0)$ denoting the value a mathematical function has for the input 0. This property would have an infinite large precision.

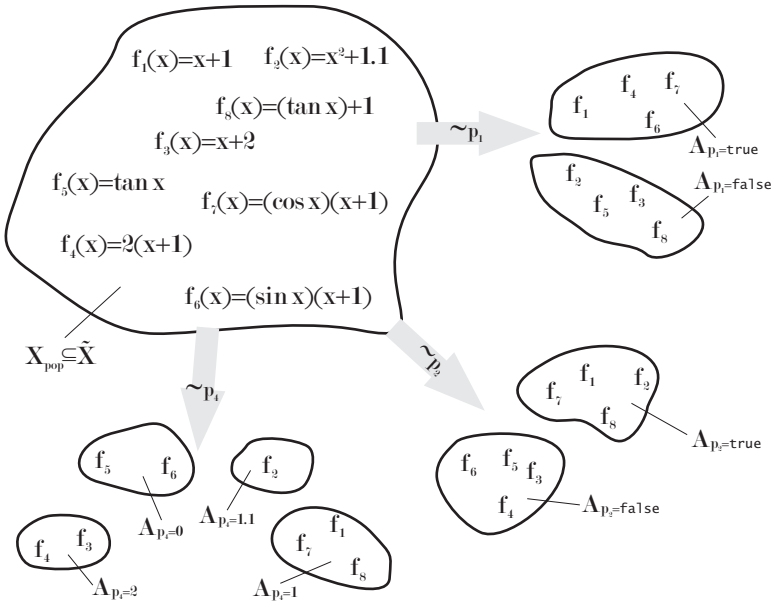


Fig. 2.4: Example for formae in symbolic regression.

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

$$A_{p_i=v} \bowtie A_{p_j=w} \Leftrightarrow A_{p_i=v} \cap A_{p_j=w} \neq \emptyset \tag{2.4}$$

$$A_{p_i=v} \bowtie A_{p_j=w} \Leftrightarrow \exists\, x : x \in A_{p_i=v} \wedge x \in A_{p_j=w} \tag{2.5}$$

$$A_{p_i=v} \bowtie A_{p_i=w} \Rightarrow w = v \tag{2.6}$$

Of course, two different formae of the same property p_i, i. e. two different peculiarities of p_i, are always incompatible. In our initial symbolic regression example hence $A_{p_1=\texttt{true}} \not\bowtie A_{p_1=\texttt{false}}$ 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 p_1 and p_2 on the other hand are compatible: $A_{p_1=\texttt{false}} \bowtie A_{p_2=\texttt{false}}$, $A_{p_1=\texttt{false}} \bowtie A_{p_2=\texttt{true}}$, $A_{p_1=\texttt{true}} \bowtie A_{p_2=\texttt{false}}$, and $A_{p_1=\texttt{true}} \bowtie A_{p_2=\texttt{true}}$. If we take p_3 into consideration, we will find that there exist some formae compatible with some of p_2 and some that are not, like $A_{p_2=\texttt{true}} \bowtie A_{p_3=1}$ and $A_{p_2=\texttt{true}} \bowtie A_{p_3=1.1}$, but $A_{p_2=\texttt{true}} \not\bowtie A_{p_3=0}$ and $A_{p_2=\texttt{true}} \not\bowtie A_{p_3=2}$.

The idea of evolutionary search is that the population X_{pop} represents a sample from the search space. In this sample, different combinations of different formae can be found. An evolutionary algorithm will discovers formae which have a good influence on the overall fitness of the solution candidates. This is done by individual evaluation, subsequent fitness assignment, and selection. We now hope that there are many compatible formae that will be gradually combined in the search process and that finally an optimal configuration is found. One aspect of forma analysis is to find out how the reproduction operations and the selection schemes influence the propagation of the good formae during the evolution. It is for example used to define lower bounds for their replication rate.

If two formae are compatible and there exist individuals in the population that are members of both, the evaluation of these individuals will also be the evaluation of the utility of the both formae. Thus, with one computation of the objective values, we inherently can determine the quality of n compatible forma.

Let us review our introductory example where we have discussed the properties of a fish in terms of forma analysis. Fish can be characterized by the properties "clever" and "strong", for example. 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 [211] for genetic algorithms and is termed *implicit parallelism* (or *intrinsic parallelism*). Since then, it has widely been studied [220, 221, 222, 223].

Holland's *Schema Theorem* elaborates on genotypic forma in genetic algorithms which can be expressed as sequence of 0, 1, and $*$, where $*$ means *don't care*. It is discussed in Section 3.6 on page 129 and provides an estimation on how such schemas will be propagated in a population by time.

Forma analysis provides puts the Schema Theorem into a broader, much more general context. It allows us to draw additional conclusions on how genomes, genetic operations, and genotype-phenotype mappings should be constructed. We discuss some of them in Section 3.7.4 on page 136.

2.2 General Information

2.2.1 Areas Of Application

Some example areas of application of evolutionary algorithms are:

Application	References
multi-objective optimization and function optimization	[41, 22, 189, 224, 225, 226, 39]
combinatorial optimization	[227]
spacecraft design	[228]
constraint satisfaction problems	[183, 224, 22, 39]
solving the satisfiability problem (SAT)	[196]
finance, trade, asset management, and taxing	[111, 229]
system identification and control optimization	[49]
data mining and analysis	[230, 231]
solving hard mathematical problems	[232]
filter design	[233]
chemistry	[234]
scheduling problems	[235, 236, 237, 238]

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 119, genetic programming in Section 4.2.1 on page 142, evolution strategy in Section 5.2.1 on page 204, evolutionary programming in Section 6.2.1 on page 210, and learning classifier systems in Section 7.2.1 on page 212.

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: 2006: Ljubljana, Slovenia, see [239]
2004: Ljubljana, Slovenia, see [240]

CEC: Congress on Evolutionary Computation
`http://ieeexplore.ieee.org/servlet/opac?punumber=7875`
[accessed 2007-09-05]
History: 2007: Singapore, see [241]
2006: Vancouver, BC, Canada, see [242]
2005: Edinburgh, Scotland, UK, see [243]

2004: Portland, Oregon, USA, see [244]
2003: Canberra, Australia, see [245]
2002: Honolulu, HI, USA, see [246]
2001: Seoul, Korea, see [247]
2000: La Jolla, California, USA, see [248]
1999: Washington D.C., USA, see [249]
1998: Anchorage, Alaska, USA, see [250]
1997: Indianapolis, IN, USA, see [251]
1996: Nagoya, Japan, see [252]
1995: Perth, Australia, see [253]
1994: Orlando, Florida, USA, see [254]

Dagstuhl Seminar: Practical Approaches to Multi-Objective Optimization
History: 2006: Dagstuhl, Germany, see [255]
2004: Dagstuhl, Germany, see [256]

EA/AE: Conference on Artificial Evolution (Evolution Artificielle)
History: 2007: Tours, France, see [257]
2005: Lille, France, see [258]
2003: Marseilles, France, see [259]
2001: Le Creusot, France, see [260]
1999: Dunkerque, France, see [261]
1997: Nîmes, France, see [262]
1995: Brest, France, see [263]
1994: Toulouse, France, see [264]

EMO: International Conference on Evolutionary Multi-Criterion Optimization
History: 2007: Matsushima/Sendai, Japan, see [265]
2005: Guanajuato, Mexico, see [266]
2003: Faro, Portugal, see [267]
2001: Zurich, Switzerland, see [268]

EUROGEN: Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems
History: 2007: Jyväskylä, Finland, see [269]
2005: Munich, Germany, see [270]
2003: Barcelona, Spain, see [271]
2001: Athens, Greece, see [272]
1999: Jyväskylä, Finland, see [273]
1997: Triest, Italy, see [274]
1995: Las Palmas de Gran Canaria, Spain, see [275]

EvoCOP: European Conference on Evolutionary Computation in Combinatorial Optimization

`http://www.evostar.org/` [accessed 2007-09-05]

Co-located with EvoWorkshops and EuroGP.

History: 2007: Valencia, Spain, see [276]
2006: Budapest, Hungary, see [277]
2005: Lausanne, Switzerland, see [278]
2004: Coimbra, Portugal, see [279]
2003: Essex, UK, see [280]
2002: Kinsale, Ireland, see [281]
2001: Lake Como, Milan, Italy, see [282]

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: 2007: Valencia, Spain, see [283]
2006: Budapest, Hungary, see [284]
2005: Lausanne, Switzerland, see [285]
2004: Coimbra, Portugal, see [286]
2003: Essex, UK, see [280]
2002: Kinsale, Ireland, see [281]
2001: Lake Como, Milan, Italy, see [282]
2000: Edinburgh, Scotland, UK, see [287]
1999: Göteborg, Sweden, see [288]
1998: Paris, France, see [289]

FEA: International Workshop on Frontiers in Evolutionary Algorithms

Was part of Joint Conference on Information Science

History: 2005: Salt Lake City, Utah, USA, see [290]
2003: Cary, North Carolina, USA, see [291]
2002: Research Triangle Park, North Carolina, USA, see [292]
2000: Atlantic City, NJ, USA, see [293]
1998: Research Triangle Park, North Carolina, USA, see [294]
1997: Research Triangle Park, North Carolina, USA, see [295]

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 143) and the International Conference on Genetic Algorithms (ICGA, see Section 3.2.2 on page 120), also "contains" the International Workshop on Learning Classifier Systems (IWLCS, see Section 7.2.2 on page 212).

History: 2007: London, England, see [296, 297]
2006: Seattle, Washington, USA, see [298]
2005: Washington, D.C., USA, see [299, 300, 301]
2004: Seattle, Washington, USA, see [302, 303]
2003: Chicago, Illinois, USA, see [304, 305]
2002: New York, USA, see [306, 307, 308, 309]
2001: San Francisco, California, USA, see [310, 311]
2000: Las Vegas, Nevada, USA, see [312, 313]
1999: Orlando, Florida, USA, see [314, 315]

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 [316, 317]
2005: Coimbra, Portugal, see [318]
2003: Roanne, France, see [319]
2001: Prague, Czech Republic, see [320]
1999: Portoroz, Slovenia, see [321]
1997: Norwich, England, see [322]
1995: Alès, France
1993: Innsbruck, Austria

Mendel: International Conference on Soft Computing
see Section 1.8.2 on page 42

PPSN: International Conference on Parallel Problem Solving from Nature
`http://ls11-www.informatik.uni-dortmund.de/PPSN/` [accessed 2007-09-05]
History: 2006: Reykjavik, Iceland, see [323]
2004: Birmingham, UK, see [324]
2002: Granada, Spain, see [325]
2000: Paris, France, see [326]
1998: Amsterdam, The Netherlands, see [327]
1996: Berlin, Germany, see [328]
1994: Jerusalem, Israel, see [329]
1992: Brussels, Belgium, see [330]
1990: Dortmund, Germany, see [331]

2.2.3 Journals

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

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 1.8.3 on page 44)

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 1.8.3 on page 43)

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. See [1].

`http://nknucc.nknu.edu.tw/~hcwu/pdf/evolec.pdf` [accessed 2007-09-16]

Last Update:	2005-02-19
Description:	Lecture Nodes on Evolutionary Computation. See [193]

2.2.5 Books

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

Bäck: *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms* (see [179])
Bäck, Fogel, Michalewicz: *Handbook of Evolutionary Computation* (see [180])
Fogel: *Evolutionary Computation: The Fossil Record* (see [332])
Deb: *Multi-Objective Optimization Using Evolutionary Algorithms* (see [20])
Eiben, Smith: *Introduction to Evolutionary Computing* (see [19])
Bentley: *Evolutionary Design by Computers* (see [333])
Morrison: *Designing Evolutionary Algorithms for Dynamic Environments* (see [94])
Weicker: *Evolutionäre Algorithmen* (see [212])
Yang, Ong, Jin: *Evolutionary Computation in Dynamic and Uncertain Environments* (see [88])
Branke: *Evolutionary Optimization in Dynamic Environments* (see [92])
Nedjah, Alba, De Macedo Mourelle: *Parallel Evolutionary Computations* (see [190])
Rothlauf: *Representations for Genetic and Evolutionary Algorithms* (see [334])
Banzhaf and Eeckman: *Evolution and Biocomputation – Computational Models of Evolution* (see [335])
Chen: *Evolutionary Computation in Economics and Finance* (see [336])

2.3 Fitness Assignment

Additional to the concept of comparing elements introduced in Section 1.3 on page 12 it is often useful if not required to assign a single real number determining a solution candidate's fitness to it. By doing so, we may loose the information needed in order to determine if the individual belongs into the optimal set or not. On the other hand, many selection algorithms need a scalar fitness to work (see Section 2.4 on page 78). Furthermore, the fitness assigned to an individual may not just reflect its rank in the population but also incorporate density/niching information which can improve the performance of the optimization algorithm significantly. If many individuals in the population occupy the same rank or do not dominate each other, this information will be very helpful. Therefore, a scalar value $\mathfrak{f}(X_{pop}) \in \mathbb{R}^+$ for an element x will be determined using a list of population individuals X_{pop} and an archive list X_{arc}. The archive is most often used if elitism is applied and usually contains the set of optimal individuals $X^\star$.

Definition 39 (Fitness Assignment). A fitness assignment process creates a function (f) which relates each element of the lists X_{pop} and X_{arc} to a positive real value (or 0).

$$\mathfrak{f} = assignFitness(\ X_{pop}, X_{arc}) \Rightarrow \mathfrak{f}(x) \in \mathbb{R}^+ \forall x \in X_{pop} \wedge \mathfrak{f}(x) \in \mathbb{R}^+ \forall x \in X_{arc} \quad (2.7)$$

Therefore, a fitness assignment process can for example use the (possible multiple) objective value(s) $f \in F$. Such fitness assignment function $\mathfrak{f}$ may be viewed as two-step translation of the form:

$$|F| = n \Rightarrow \mathfrak{f} : \tilde{X} \mapsto \mathbb{R}^n \mapsto \mathbb{R}^+ \tag{2.8}$$

Furthermore, in the context of this book we generally minimize fitness values, i. e. the lower the scalar fitness of an individual the better. Therefore, the following condition must hold for all fitness assignment processes on basis of the prevalence relation:

$$x_1 \succ x_2 \Rightarrow \mathfrak{f}(x_1) \leq \mathfrak{f}f(x_2) \; \forall \; x_1, x_2 \in appendList(X_{pop}, X_{arc}) \tag{2.9}$$

Although it is not very common, it is also possible to chain fitness assignment algorithms. Since we define the fitness assignment processes by using the objective function values either directly or indirectly via the prevalence comparators c_F, a primary scalar fitness assignment function $\mathfrak{f}_1(x)$ could be used in order to compute a secondary fitness assignment function $\mathfrak{f}_2(x)$. By doing so, one could base the tournament fitness assignment process (see Section 2.3.4 on page 69) on SPEA fitness assignment (see Section 2.3.10 on page 76).

2.3.1 Weighted Sum Fitness Assignment

The most primitive fitness assignment strategy would be assigning a weighted sum of the objective values. This method corresponds to the weighted sum prevalence comparator introduced in Section 1.3.5 on page 20. The difference is that the prevalence function $c_{F,weightedS}$ in Equation 1.15 on page 21 compares two different individuals whereas weighted fitness assignment defines one single value defining the worth of a single solution candidate. Like in Equation 1.15, we specify the weights of the objective values w and obtain the formula:

$$\mathfrak{f}(x) = weightedSumFitnessAssign(X_{pop}, X_{arc}) \Leftrightarrow \mathfrak{f}(x) \equiv \sum_{i=1}^{|F|} w_i f_i(x) \tag{2.10}$$

2.3.2 Prevalence-Count Fitness Assignment

Another very simple method of fitness assignment would be to use fitness values that directly reflect the prevalence relation illustrated in Figure 2.5.

There are two methods for computing such fitness values. (Remember that they will later be subject to minimization.)

- Assign to each individual a number inversely proportional to the count of other individuals it prevails ($prevalenceFitnessAssign_1$, Algorithm 2.3).

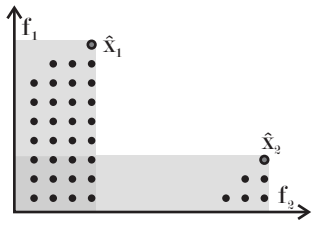


Fig. 2.5: The dominated sets of the individuals $\hat{x}_1$ and $\hat{x}_2$.

It is clear that individuals that dominate many others will receive a higher fitness than those which are prevailed by many. The disadvantage of this approach is that it promotes individuals that reside in larger niches and discriminates individuals in smaller niches. Figure 2.5 illustrates this problem: If the objective functions f_1 and f_2 were to be maximized, the dominated (prevailed) set of $\hat{x}_1$ is much larger than the one of $\hat{x}_2$. $\hat{x}_1$ and $\hat{x}_2$ however are both non-prevailed solutions and should thus be treated equally – the $prevalenceFitnessAssign1$ method would instead lead to constant growth of the niche of $\hat{x}_1$. The niche of $\hat{x}_2$ would diminish and eventual disappear. Also, all the upper the individuals in the niche of $\hat{x}_1$ will be valued higher than $\hat{x}_2$, although their are dominated.

- If assigning to each individual the number of other individuals it is prevailed by [337, 338] ($prevalenceFitnessAssign_2$, Algorithm 2.4), this effect will not occur. Both, $\hat{x}_1$ and $\hat{x}_2$ are non-dominated and will have the same, minimum fitness value. The individuals in their niches will also receive fitness values more useful.

Such fitness assignment methods are very rough but provide pressure in direction of the prevalence front. They do not incorporate any means of diversity preservation.

2.3.3 Rank-Based Fitness Assignment

The rank-based fitness assignment method (also known as *Pareto ranking*) rates individuals based on their rank in the total population. It first sorts the list containing the population and the archive ascending according to the prevalence comparator function c_F. Non-prevailed individuals will thus be at the beginning of this list while individuals prevailed by many others are at the end. Ranks are increasing with the index in this list but are the same for neighboring individuals do not prevailing each other [339, 340, 341]. Algorithm 2.5 presents this assignment procedure. It should be noted that this method is a bit of a crude approach which has some disadvantages if compared with the more sophisticated ideas of $prevalenceFitnessAssign_2$ and the following algorithms.

Algorithm 2.3: $\mathfrak{f}(x) = prevalenceFitnessAssign_1(X_{pop}, X_{arc})$

Input: X_{pop} the population to assign fitness values to
Input: X_{arc} the archive (normally the empty list ())
Data: X_l the list representation of the unison of X_{pop} and X_{arc}
Data: i, j, cnt counter variables
Output: $\mathfrak{f}(x)$ the fitness assignment function which assigns a scalar fitness to all individuals in X_{pop} and X_{arc}

```
1  begin
2      X_l ⟵ appendList(X_pop, X_arc)
3      i ⟵ |X_l| − 1
4      while i ≥ 0 do
5          cnt ⟵ 0
6          j ⟵ |X_l| − 1
7          while j ≤ 0 do
8              if (j ≠ i) ∧ (X_l[i] ≻ X_l[j]) then cnt ⟵ cnt + 1
9              j ⟵ j − 1
10         f(X_l[i]) ⟵ 1/(cnt+1)
11         i ⟵ i − 1
12     return f
13 end
```

Algorithm 2.4: $\mathfrak{f}(x) = prevalenceFitnessAssign_2(X_{pop}, X_{arc})$

Input: X_{pop} the population to assign fitness values to
Input: X_{arc} the archive (normally the empty list ())
Data: X_l the list representation of the unison of X_{pop} and X_{arc}
Data: i, j, cnt counter variables
Output: $\mathfrak{f}(x)$ the fitness assignment function which assigns a scalar fitness to all individuals in X_{pop} and X_{arc}

```
1  begin
2      X_l ⟵ appendList(X_pop, X_arc)
3      i ⟵ |X_l| − 1
4      while i ≥ 0 do
5          cnt ⟵ 0
6          j ⟵ |X_l| − 1
7          while j ≤ 0 do
8              if (j ≠ i) ∧ (X_l[j] ≻ X_l[i]) then cnt ⟵ cnt + 1
9              j ⟵ j − 1
10         f(X_l[i]) ⟵ cnt
11         i ⟵ i − 1
12     return f
13 end
```

Algorithm 2.5: $\mathfrak{f}(x) = rankBasedFitnessAssign(X_{pop}, X_{arc})$

Input: X_{pop} the population to assign fitness values to
Input: X_{arc} the archive (normally the empty list ())
Data: X_l the list representation of the unison of X_{pop} and X_{arc}
Data: i counter variable
Data: r rank counter
Output: $\mathfrak{f}(x)$ the fitness assignment function which assigns a scalar fitness to all individuals in X_{pop} and X_{arc}

```
1  begin
2      X_l ⟵ appendList(X_pop, X_arc)
3      X_l ⟵ sort_a(X_l, c_F)
4      i ⟵ 1
5      r ⟵ 1
6      f(X_l[0]) ⟵ 1
7      while i < |X_l| do
8          if c_F(X_l[i], X_l[i − 1]) > 0 then r ⟵ r + 1
9          f(X_l[i]) ⟵ r
10         i ⟵ i + 1
11     return f
12 end
```

2.3.4 Tournament Fitness Assignment

In tournament fitness assignment, which is a generalization of the q-level binary tournament selection introduced in [212], the fitness of each individual is computed by letting it compete q times against r other individuals (with $r = 1$ as default) and counting its victories. For a better understanding of the tournament metaphor, see Section 2.4.3 on page 81 where the tournament selection scheme is discussed.

2.3.5 Sharing Functions

Definition 40 (Sharing Function). A sharing function Sh is a function that relates two individuals x_1 and x_2 to a value decreasing with their distance $dist(x_1, x_2)$ in a way that it is 1 for $dist(x_1, x_2) = 0$ and 0 if the distance exceeds a specified constant σ.

$$Sh(x_1, x_2) = \begin{cases} 1 \; if \; dist(x_1, x_2) \leq 0 \\ \in [0, 1] \; if \; 0 < dist(x_1, x_2) < \sigma \\ 0 \; otherwise \end{cases} \tag{2.11}$$

$$Sh(x_1, x_2) \in \mathbb{R}^+ \tag{2.12}$$

Sharing functions are employed by many fitness assignment processes [342, 343]. Often, the simple triangular function Sh_t or one of its either convex

Algorithm 2.6: $\mathfrak{f}(x) = tournamentFitnessAssign_{q,r}(X_{pop}, X_{arc})$

Input: X_{pop} the population to assign fitness values to
Input: X_{arc} the archive (normally the empty list ())
Input: Implicit: q the count of tournaments per individuals
Input: Implicit: r the count of other contestants per tournament, normally 1
Data: X_l the list representation of the unison of X_{pop} and X_{arc}
Data: z the counter of tournament wins
Data: i, j, k counter variables
Output: $\mathfrak{f}(x)$ the fitness assignment function which assigns a scalar fitness to all individuals in X_{pop} and X_{arc}

```
1  begin
2      X_l ⟵ appendList(X_pop, X_arc)
3      i ⟵ |X_l| − 1
4      while i > 0 do
5          j ⟵ q
6          z ⟵ 0
7          while j > 0 do
8              b ⟵ true
9              k ⟵ r
10             while (k > 0) ∧ b do
11                 b ⟵ X_l[i] ≻ X_l [⌊random_u (|X_l|)⌋]
12                 k ⟵ k − 1
13             if b then z ⟵ z + 1
14             j ⟵ j − 1
15         f(X_l[i]) ⟵ 1/(z+1)
16         i ⟵ i − 1
17     return f
18 end
```

($Sh_{v,p}$) or concave ($Sh_{n,p}$) pendants with the power $p \in \mathbb{R}^+, p > 0$ are applied. One can also use the exponential version defined as Sh_e.

$$Sh_t(x_1, x_2) = \begin{cases} 1 - \frac{dist(x_1,x_2)}{\sigma} & if\ dist(x_1, x_2) < \sigma \\ 0 & otherwise \end{cases} \tag{2.13}$$

$$Sh_{v,p}(x_1, x_2) = \begin{cases} \left(1 - \frac{dist(x_1,x_2)}{\sigma}\right)^p & if\ dist(x_1, x_2) < \sigma \\ 0 & otherwise \end{cases} \tag{2.14}$$

$$Sh_{n,p}(x_1, x_2) = \begin{cases} 1 - \left(\frac{dist(x_1,x_2)}{\sigma}\right)^p & if\ dist(x_1, x_2) < \sigma \\ 0 & otherwise \end{cases} \tag{2.15}$$

$$Sh_e(x_1, x_2) = \begin{cases} \frac{1}{1-e^{-1}} \left(1 - e^{\frac{dist(x_1,x_2)}{\sigma} - 1}\right) & if\ dist(x_1, x_2) < \sigma \\ 0 & otherwise \end{cases} \tag{2.16}$$

The Euclidian distance measure $dist_{eucl} \equiv dist_{n,2}$ in objective space is normally used in sharing functions, but of course, any other distance measure could be applied.

The niche count $m(x, X_{tst})$ [344] of an individual x is the sum of a sharing function of this individual to all other individuals in a test list X_{tst}. As test list the population which x is part of could be used as well as an external set of elements.

In , we have defined m on basis of an element x and assumed that X_{tst} is a set.

$$m(x, X_{tst}) = \sum_{\forall x_t \in X_{tst}: x_t \neq x} Sh(x, x_t) \tag{2.17}$$

Sets however do not allow the same element to occur more than once. Our general problem is here that this might of course happen in populations of evolutionary algorithms. Then, X_{tst} is necessarily a list. Furthermore, if we stick with the $=$ operator between two elements, we will not be able to take multiple occurrences of the same element as x in X_{sel}. It is however clear that $m(x = 1, X_{tst})$ should be greater than $m(x = 2, X_{tst})$ if $X_{tst} = (1, 2, 3, 1, 1, 1)$. By adding $\max\{0, countItemOccurrences(x, X_{tst}) - 1\}$ we take the fact of the missed sharing values due to $(x = X_{tst}[0] = X_{tst}[3] = \ldots$ into account. The sharing function would be 1 for all equal elements ($Sh(x, x) = 1$). The max is needed just in case that $x \notin X_{tst}$. Based on these modifications, Equation 2.18 provides a better definition for the niche count.

$$m(x, X_{tst}) = \left(\sum_{j=0}^{|X_{tst}|-1} \begin{cases} 0 & if\, x = X_{tst}[j] \\ Sh(X_{tst}[i], X_{tst}[j]) & if\, x \neq X_{tst}[j] \end{cases} \right) + \max\{0, countItemOccurrences(x, X_{tst}) - 1\} \tag{2.18}$$

2.3.6 Niche Size Fitness Assignment

Niche size fitness assignment attributes a fitness value to each individual which reflects how crowded the niche of the individual is. With niche we mean the area in a given distance around the individual (see Section 36.1). The crowdedness then describes the count of other individuals inside that niche. It is usually not wanted that niches are very crowded, since that would mean that the optimization algorithm converges fast. Instead, you want many niches with few individuals inside which means that many different solution paths are followed and a broader scan of the optimal front can be obtained. Therefore, the niche size fitness assignment will attach small fitness values to individuals with crowded niches and large fitness values to those which are lonesome.

Notice that the niche fitness does not contain any information about the prevalence/dominance or other objective features, it solely concentrates on niching and should thus be only used in conjunction with the prevalence function c_F.

The niche size fitness of an individual x is the sum of a sharing function of this individual to all the other individuals in the population and therefore its niche count $m(x, X_{pop} \cup X_{arc})$ (see Equation 2.19 and the equivalent Algorithm 2.7). In its original application in the NPGA [345] (see Section 2.4.9), the triangular sharing function Sh_t is used.

$$\begin{aligned} \mathfrak{f}(x) = \ & nicheSizeFitnessAssign(X_{pop}, X_{arc}) \\ & \Leftrightarrow \mathfrak{f}(x) \equiv m(x, appendList(X_{pop}, X_{arc})) \end{aligned} \tag{2.19}$$

Algorithm 2.7: $\mathfrak{f}(x) = nicheSizeFitnessAssign(X_{pop}, X_{arc})$

Input: X_{pop} the population to assign fitness values to
Input: X_{arc} the archive (which normally equals the set of optimal individuals $X^\star$ or the empty list ())
Input: Implicit: Sh, $dist$ a sharing function and a distance measure
Input: Implicit: c_F the prevalence function
Data: i, j individual indices we iterate over
Data: X_l the joinded lists X_{arc} and X_{pop}
Data: sum the sharing function sum
Output: $\mathfrak{f}(x)$ the fitness assignment function which assigns a scalar fitness to all individuals in X_{pop} and X_{arc}

```
1  begin
2      X_l ⟵ appendList(X_pop, X_arc)
3      i ⟵ |X_l| − 1
4      while i ≥ 0 do
5          sum ⟵ 0
6          j ⟵ |X_l| − 1
7          while j ≥ 0 do
8              if i ≠ j then sum ⟵ sum + Sh(X_l[i], X_l[j])
9          f(X_l[i]) ⟵ sum
10         i ⟵ i + 1
11     return f
12 end
```

2.3.7 NSGA Fitness Assignment

The Nondominated Sorting Genetic Algorithm by Srninivas and Deb [346] uses sharing (see Section 2.3.5 on page 69) in order to assign fitness values. It therefore first picks all the non-prevailed individuals Z from the population X_{pop} and assigns a fitness value to them decreased by a sharing function. After removing these individuals from the population, it repeats the first step

and again selects all the non-prevailed individuals and assigns a fitness value smaller than the smallest fitness of the previous turn to them. This step is iterated until fitness values are related to all individuals. This approach does obviously lead to fitness that must be maximized. In Algorithm 2.8 we modified this process in order to obtain fitness values that can be minimized.

In principle, this fitness assignment process resembles a sum of the rank-based fitness assignment (see Section 2.3.3) and the niche size fitness assignment (see Algorithm 2.7): $nsgaFitnessAssign \approx rankBasedFitnessAssign + nicheSizeFitnessAssign$.

Algorithm 2.8: $\mathfrak{f}(x) = nsgaFitnessAssign(X_{pop}, X_{arc})$

Input: X_{pop} the population to assign fitness values to
Input: X_{arc} the list of optimal individuals, normally ()
Input: Implicit: c_F the prevalence function
Data: X_a the unison of X_{pop} and X_{arc}
Data: $X_f \subseteq X_a$ the current non-prevailed front
Data: n, m maximum fitness values, used for the current/next set of non-prevailed individuals
Data: x, u individuals in X_a
Output: $\mathfrak{f}(x)$ the fitness assignment function which assigns a scalar fitness to all individuals in X_{pop} and X_{arc}

```
1  begin
2      X_a ⟵ appendList(X_arc, X_pop)
3      m ⟵ 1
4      while |X_a| > 0 do
5          X_f ⟵ extractOptimalSet(X_a)
6          X_a ⟵ X_a \ X_f
7          n ⟵ m
8          foreach x ∈ X_f do
9              f(x) ⟵ m (1 + Σ_{∀u∈X_f} Sh(x, u))
10             if f(x) > n then n ⟵ f(x)
11         m ⟵ n + 1
12     return f
13 end
```

2.3.8 NSGA2 Fitness Assignment

The NSGA2 algorithm [347] uses a fitness assignment process similar to the one in NSGA in Section 2.3.7, except for three improvements:

1. Elitism is applied in order to preserve the best individuals by taking both, the previous and the current population into account.

2. Instead of sharing, the crowding distance (see Section 35.8.3 on page 568) is used for niching.
3. The fitness assignment is only applied to the first p individuals, where p denotes the size of the next population.

The Algorithm 2.9 realizes the first two points by assigning a basic fitness value to each non-prevailed front of the unison of the two populations (similar to NSGA-fitness assignment) and offsetting it by the crowding distance $\mathfrak{cd}$ (Section 35.8.3 on page 568). It does however not consider the last point by performing the fitness assignment to all elements in the population, since that matches with our model. This will lead to a slightly higher computational time than the original algorithm has, and could be corrected in a real implementation.

Algorithm 2.9: $\mathfrak{f}(x) = nsga2FitnessAssign(X_{pop}, X_{arc})$

Input: X_{pop} the population to assign fitness values to, the unison of the current and previous population in $nsga2$
Input: X_{arc} the list of optimal individuals, normally ()
Input: Implicit: c_F the prevalence function
Data: X_a the unison of X_{pop} and X_{arc}
Data: $Z \subseteq X_a$ the current non-prevailed front
Data: n, m maximum fitness values, used for the current/next set of non-prevailed individuals
Data: x the current individual
Output: $\mathfrak{f}(x)$ the fitness assignment function which assigns a scalar fitness to all individuals in X_{pop} and X_{arc}

```
1  begin
2      X_a ⟵ appendList(X_arc, X_pop)
3      m ⟵ 1
4      while |X_a| > 0 do
5          Z ⟵ extractOptimalSet(X_a)
6          X_a ⟵ listToSet(X_a) \ Z
7          computeCrowdingDistance(Z)
8          n ⟵ m
9          foreach x ∈ Z do
10             f(x) ⟵ m + ρ_cd(x)
11             if f(x) > n then n ⟵ f(x)
12         m ⟵ n + 1
13     return f
14 end
```

2.3.9 RPSGAe Fitness Assignment

The Reduced Pareto Set Genetic Algorithm with Elitism (RPSGAe) [348] has another interesting fitness assignment process which is specified in Algorithm 2.10. A maximum count of ranks n is defined for the algorithm. The variable r is used in a loop as counter starting from $r = 1$ to $n - 1$. In this loop, the population X_a is reduced to $r\frac{|X_a|}{n}$ individuals by clustering. To all representative individuals $nucleus(b)$ in the resulting set of clusters B, the rank r is assigned if they do not have a rang yet. After the loop, the remaining individuals receive the rank n. The fitness $\mathfrak{f}(x)$ of an individual x is then assigned to a linear or exponential function of its rank divided by its niche count $m(x, X_a)$ (see Section 2.3.5 on page 69).

Algorithm 2.10: $\mathfrak{f}(x) = rpsgaeFitnessAssign_n(X_{pop}, X_{arc})$

Input: X_{pop} the population to assign fitness values to, the unison of the current and previous population in $nsga2$
Input: X_{arc} the list of optimal individuals, normally ()
Input: Implicit: n the total count of ranks to use
Input: Implicit: g a linear or exponential function used for ranking
Input: Implicit: *cluster* a clusterin algorithm
Data: X_a, x the unison of X_{pop} and X_{arc} and an element in that set
Data: $\mathfrak{s}$ the rank assignment function
Data: B, b the clustering result and a cluster
Data: r the rank counter
Data: l the count of individuals to reduce X_a to by clustering
Output: $\mathfrak{f}(x)$ the fitness assignment function which assigns a scalar fitness to all individuals in X_{pop} and X_{arc}

```
1  begin
2      X_a ⟵ listToSet(appendList(X_pop, X_arc))
3      forall x ∈ X_a do s(x) ⟵ 0
4      r ⟵ 1
5      while r < n do
6          l ⟵ ⌊r |X_a|/n + 0.5⌋
7          B ⟵ cluster_l(X_a)
8          foreach b ∈ B do
9              if s(nucleus(b)) = 0 then s(nucleus(b)) ⟵ r
10     forall x ∈ X_a do if s(x) = 0 then s(x) ⟵ n
11     forall x ∈ X_a do f(x) = g(s(x)) / (m(x, X_a)+1)
12     return f
13 end
```

2.3.10 SPEA Fitness Assignment

This fitness assignment process is used by the Strength Pareto Evolutionary Algorithm ([349], see Section 2.6.13 on page 110). It uses the optimal set in order to evaluate the individuals. The optimal set therefore has to be updated and pruned before the fitness assignment takes place. First, a strength value s is assigned to each member x_{arc} of the optimal set X_{arc} (where the optimal set $X^\star$ is passed in as archive $X_{arc} = X^\star$).

$$\mathfrak{f}(x_{arc} \in X_{arc}) = s(x_{arc}) = \frac{|x \in X_{pop} : x_{arc} \succ x|}{|X_{pop} + 1|} \tag{2.20}$$

These values represent the fitness of the optimal individuals, while the fitness of the non-optimal individuals is

$$\mathfrak{f}(x \in X_{pop}) = 1 + \sum_{\forall x_{arc} \succ x} \mathfrak{f}(x_{arc}) \tag{2.21}$$

Algorithm 2.11 describes this process more precisely.

2.3.11 SPEA2 Fitness Assignment

This fitness assignment process is utilized by the Strength Pareto Evolutionary Algorithm 2 ([350, 351] see Section 2.6.14 on page 111). Other than in the original SPEA algorithm, it uses both, the archive X_{arc} and the population X_{pop}, in order to evaluate the individuals. Note that SPEA2 (unlike SPEA) does not use a set of strictly optimal solution candidates as archive but a list of constant length containing the best individuals. First, a strength value $S(x)$ for a solution candidate $x \in setToList(appenList(X_{arc}, X_{pop}))$ is computed for all individuals equal to the number of individuals it dominates:

$$\begin{aligned} S(x) = \; & |\{\forall\, i \in 0 \ldots |X_{arc}| - 1 : x \succ X_{arc}[i]\}| + \\ & |\{\forall\, j \in 0 \ldots |X_{pop}| - 1 : x \succ X_{pop}[j]\}| \end{aligned} \tag{2.22}$$

Using that strength value, the raw fitness R of the individual x is determined:

$$\begin{aligned} R(x) = & \sum_{i=0}^{|X_{arc}|-1} \begin{cases} S(X_{arc}[i]) & \textit{if } X_{arc}[i] \succ x \\ 0 & \textit{otherwise} \end{cases} \\ & \sum_{j=0}^{|X_{pop}|-1} \begin{cases} S(X_{pop}[j]) & \textit{if } X_{pop}[j] \succ x \\ 0 & \textit{otherwise} \end{cases} \end{aligned} \tag{2.23}$$

The smaller the resulting fitness is, the better is the solution candidate. Non-prevailed individuals x_{arc} have a raw fitness of $R(x_{arc}) = 0$. This raw fitness already provides some sort of niching based the concept of prevalence, but may fail if most individuals of the population do not dominate each other. It is

Algorithm 2.11: $\mathfrak{f}(x) = speaFitnessAssign(X_{pop}, X_{arc})$

Input: X_{pop} the population to assign fitness values to
Input: X_{arc} the archive (which normally equals the set of optimal individuals $X^\star$)
Input: Implicit: c_F the prevalence function
Data: $x \in X_{pop}$ the current individual
Data: $x_{arc} \in X_{arc}$ the current archive individual
Data: *sum*, *counter* internal counters
Output: $\mathfrak{f}(x)$ the fitness assignment function which assigns a scalar fitness to all individuals in X_{pop} and X_{arc}

```
1  begin
2      i ⟵ |X_arc| − 1
3      while i ≥ 0 do
4          count ⟵ 0
5          j ⟵ |X_pop| − 1
6          while j ≥ 0 do
7              if X_arc[i] ≻ X_pop[j] then count ← count + 1
8              j ⟵ j − 1
9          f(X_arc[i]) ⟵ count / |X_pop|
10         i ⟵ i − 1
11     j ⟵ |X_pop| − 1
12     while i ≥ 0 do
13         if search_u(X_pop[j], X_arc) < 0 then
14             sum ⟵ 0
15             i ⟵ |X_arc| − 1
16             while j ≥ 0 do
17                 if X_arc[i] ≻ X_pop[j] then sum ← sum + f(X_arc[i])
18                 j ⟵ j − 1
19         f(X_pop[j]) ⟵ sum + 1
20         i ⟵ i − 1
21     return f
22 end
```

therefore complemented by a density estimate (see Section 35.8 on page 567), namely the kth nearest neighbor estimate $\rho_{nn,k}$ with $k = \sqrt{|X_{arc} \cup X_{pop}|}$ as introduced in Section 35.8.2 on page 567, is used to compute the fitness $\mathfrak{f}$. This density estimate (see line 20 in Algorithm 2.12 on the following page) can be replaced by any other suitable density estimate.

$$D(x) = \frac{1}{2 + \rho(x)} \tag{2.24}$$

$$\mathfrak{f}(x) = R(x) + D(x) \tag{2.25}$$

This process is specified in Algorithm 2.12 on the next page.

Algorithm 2.12: $\mathfrak{f}(x) = spea2FitnessAssign(X_{pop}, X_{arc})$

Input: X_{pop} the population to assign fitness values to
Input: X_{arc} the list of optimal individuals
Input: Implicit: c_F the prevalence function
Data: X_a the list containing unison of X_{pop} and X_{arc}
Data: k the k-value for the kth nearest neighbor density estimate $\rho_{nn,k}$ - other values for k could be chosen too
Data: i, j indexes into X_a used for computation
Data: S the list of the strength values
Data: sum, $counter$ internal counters
Output: $\mathfrak{f}(x)$ the fitness assignment function which assigns a scalar fitness to all individuals in X_{pop} and X_{arc}

```
1  begin
2      X_a ⟵ appendList(X_arc, X_pop)
3      k ⟵ √|X_a|
4      S ⟵ createList(|X_a|, 0)
5      i ⟵ |X_a| − 1
6      while i ≥ 0 do
7          count ⟵ 0
8          j ⟵ |X_a| − 1
9          while j ≥ 0 do
10             if X_a[i] ≻ X_a[j] then count ← count + 1
11             j ⟵ j − 1
12         S[i] ⟵ count
13         i ⟵ j − 1
14     i ⟵ |X_a| − 1
15     while i ≥ 0 do
16         sum ⟵ 0
17         j ⟵ |X_a| − 1
18         while j ≥ 0 do
19             if X_a[j] ≻ X_a[i] then sum ← sum + S[j]
20         f(X_a[i]) ⟵ sum + 1/(2+ρ_{nn,k}(X_a[i]))
21     return f
22 end
```

2.4 Selection

Definition 41 (Selection). The operation *select* is used to choose a list of individuals for reproduction, the so-called mating pool X_{mp}, from a list of solution candidates X_{sel} [179, 18, 352, 17].

Selection[13] may behave in a deterministic or in a randomized manner, according to its application-dependant implementation.

[13] `http://en.wikipedia.org/wiki/Selection_%28genetic_algorithm%29` [accessed 2007-07-03]

There exist two classes of selection algorithms: such *with replacement* (annotated with a subscript $_r$, Equation 2.27) and such *without replacement* (annotated with a subscript $_w$, Equation 2.27) [353]. In a selection algorithm without replacement, each individual from the input list X_{sel} is taken into consideration for reproduction at most once and therefore also will occur in the mating pool one time at most. The mating pool list returned by algorithms with replacement can contain the same individual multiple times. This is the reason why the mating pool is always represented as a list and not as a set. The parameter n of the selection algorithm corresponds to the size of the mating pool wanted. Notice that there may be restrictions on n in algorithms without replacement.

$$X_{mp} = select_w(X_{sel}, n) \Rightarrow \forall\, x_{mp} \in X_{mp} \Rightarrow countItemOccurrences(x_{mp}, X_{mp}) \leq countItemOccurrences(x_{mp}, X_{sel}) \wedge |X_{mp}| \leq n \tag{2.26}$$

$$X_{mp} = select_r(X_{sel}, n) \Rightarrow \forall\, x_{mp} \in X_{mp} \Rightarrow x_{mp} \in X_{sel} \wedge |X_{mp}| = n \tag{2.27}$$

Selection may work directly on the individuals using the prevalence comparator function c_F (see Section 1.3.5 on page 20) or by referring to a previously executed fitness assignment process (Section 2.3 on page 65). Selection algorithms that base on prevalence are annotated with a superscript ^{p} whereas algorithms that use fitness assignment are annotated with a superscript $^{\mathfrak{f}}$. When taking down the algorithms, we will omit that annotations, it just plays a role when wanting to specify exactly what selection scheme is used in an optimization process.

There exist selection algorithms which can only work on scalar fitness and thus need to rely on a fitness assignment process in multi-objective optimization. Algorithms that may exist in both variants will use the $x_1 > x_2$ operator for internal individual comparison. This operators will be replaced by $x_1 \succ x_2$ in directly prevalence-based selection and by $\mathfrak{f}(x_1) > \mathfrak{f}(x_2)$ in fitness assignment based realizations.

Selection can be used in conjunction with the other operations defined to extend single-solution algorithms to support a list of solutions.

Another possible classification of selection algorithms has already been discussed in Section 2.1.3 on page 54 and depends on the composition of this list X_{sel}. In (μ, λ)-selection, the selection algorithm returns μ elements of a set $|X_{sel}| = \lambda$ of the child individuals of the current generation. In $(\mu + \lambda)$-algorithms, X_{sel} contains λ children and μ parents of the current population and the selection returns μ elements.

Selection algorithms can be chained – 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.

2.4.1 Truncation Selection

Truncation selection[14], also called deterministic selection, returns the best elements of the list X_{sel}. Such a selection algorithm will usually not be a good choice since it does not preserve the diversity. It is especially bad if applied to an optimal set where no element is better than another one. Below, the algorithm in the form with replacement is noted – it simple returns an list containing n-times the minimal element of X_{sel}.

$$X_{mp} = truncationSelect_r(X_{sel}, n) \Leftrightarrow X_{mp} = createList(n, min_{\succ}\{X_{sel}\}) \tag{2.28}$$

The definition of Truncation selection without replacement is presented in Algorithm 2.13. Here, the n best individuals are extracted from the selectable population X_{sel} and placed into the mating pool X_{mp}. For n normally values like $\frac{|X_{sel}|}{2}$ or $\frac{|X_{sel}|}{3}$ are used.

Notice that computing the Truncation element of a set uses the $\cdot \succ \cdot$ comparison operator internally. Thus, the Truncation selection is available in both, prevalence and fitness assignment based flavors.

Algorithm 2.13: $X_{mp} = truncationSelect_w(X_{sel}, n)$

Input: X_{sel} the list of individuals to select from
Input: $n \leq |X_{sel}$ the number of individuals to be placed into the mating pool X_{mp}
Input: Implicit: c_F the prevalence comparator function
Data: $x_{sel} \in X_{sel}$ the next minimal element
Data: i a counter variable
Output: X_{mp} the individuals selected

```
1  begin
2      X_mp ⟵ ()
3      i ⟵ n − 1
4      while i ≥ 0 do
5          x_sel ⟵ min_≻{X_sel}
6          X_sel ⟵ removeListItem(X_sel, x_sel)
7          X_mp ⟵ addListItem(X_mp, x_sel)
8          i ⟵ i − 1
9      return X_mp
10 end
```

2.4.2 Random Selection

Random selection returns elements by chance. A possible preceding fitness assignment process as well as the objective values of the individuals play no

[14] `http://en.wikipedia.org/wiki/Truncation_selection` [accessed 2007-07-03]

role at all. This hinders the optimization algorithm to follow any gradient in the fitness landscape – it is effectively turned into a random walk. Random selection is thus not applied exclusively, but can serve as mating selection scheme in conjunction with a separate environmental selection. It maximally preserves the diversity and can be a good choice if used to pick elements from an optimal set.

Since random selection bases on uniformly distributed random numbers, it is a good and simple model to derive general properties of selection algorithms from.

Algorithm 2.14 and Algorithm 2.15 demonstrate how random selection with and without replacement can be performed. A more sophisticated random method which also preserves the order of the individuals is presented in [354].

Algorithm 2.14: $X_{mp} = rndSelect_r(X_{sel}, n)$

Input: X_{sel} the list of individuals to select from
Input: n the number of individuals to be placed into the mating pool X_{mp}
Input: Implicit: c_F the prevalence comparator function
Data: i a counter variable
Output: X_{mp} the individuals selected

```
1 begin
2 |   X_mp ⟵ ()
3 |   i ⟵ n − 1
4 |   while i ≥ 0 do
5 |   |   X_mp ⟵ addListItem(X_mp, X_sel [⌊random_u(|X_s|)⌋])
6 |   |   i ⟵ i − 1
7 |   return X_mp
8 end
```

2.4.3 Tournament Selection

In tournament selection[15] [355, 356], k elements will be picked out of a list X_{sel} and compete with each other. The winner of this competition will then enter mating pool X_{mp}. Although being a very simple strategy, it is very powerful and therefore used in many practical applications [357, 358, 359].

Consider a tournament selection (with replacement) with a tournament size of two, where the winners are allowed to enter the mating pool. For each single tournament, the contestants are chosen randomly according to an uniform distribution. Hence, each individual will on average participate in two tournaments. The best solution candidate of the population will win all the

[15] `http://en.wikipedia.org/wiki/Tournament_selection` [accessed 2007-07-03]

Algorithm 2.15: $X_{mp} = rndSelect_w(X_{sel}, n)$

Input: X_{sel} the list of individuals to select from
Input: $n \leq |X_{sel}|$ the number of individuals to be placed into the mating pool X_{mp}
Input: Implicit: c_F the prevalence comparator function
Data: i a counter variable
Data: j the index of the element to be selected next
Output: X_{mp} the individuals selected

```
1  begin
2      X_mp ⟵ ()
3      i ⟵ n − 1
4      while i ≥ 0 do
5          j ⟵ ⌊random_u(|X_sel|)⌋
6          X_mp ⟵ addListItem(X_mp, X_sel[j])
7          X_sel ⟵ deleteListItem(X_sel, j)
8          i ⟵ i − 1
9      return X_mp
10 end
```

contests it takes part in and, again on average, will contribute 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 one time on average. The worst individual in the population will loose all its challenges and thus will not be able to reproduce [360].

Tournament selection may directly work using the prevalence comparator function or it could make use of a previous fitness assignment process. If the list X_{sel} is already sorted, then a direct comparison between the k competitors is not needed anymore – generating k random indices and returning the element at the smallest of them will be sufficient.

Tournament selection with replacement is presented in Algorithm 2.16. Tournament selection without replacement [361] exists in two versions: normally, only the selected individuals are removed from the set X_{sel} (see Algorithm 2.17) - it is though possible to remove all the competitors of the tournaments (Algorithm 2.18 on page 85).

The algorithms introduced here should more specifically be entitled as *deterministic* tournament selection algorithms [356] since the winner of the k contestants that take part in each tournament enters the mating pool.

There also exists a non-deterministic variant of this selection type[16] where this is not necessarily the case. Therefore, 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 (i.e., the ith best with probability $p(1-p)^i$). Algorithm 2.19 on page 86

[16] `http://en.wikipedia.org/wiki/Tournament_selection` [accessed 2007-07-03]

Algorithm 2.16: $X_{mp} = tournamentSelect_{r,k}(X_{sel}, n)$

Input: X_{sel} the list of individuals to select from
Input: n the number of individuals to be placed into the mating pool X_{mp}
Input: Implicit: k the tournament size
Input: Implicit: c_F the prevalence comparator function
Data: a, b the indexes of the tournament contestants in X_{sel}
Data: i, j counter variables
Output: X_{mp} the winners of the tournaments which now form the mating pool

```
1  begin
2      X_mp ⟵ ()
3      i ⟵ n
4      while i > 0 do
5          j ⟵ k − 1
6          a ⟵ ⌊random_u(|X_sel|)⌋
7          while j > 0 do
8              b ⟵ ⌊random_u(|X_sel|)⌋
9              if X_sel[b] ≻ X_sel[a] then a ⟵ b
10             j ⟵ j − 1
11         X_mp ⟵ addListItem(X_mp, X_sel[a])
12         i ⟵ i − 1
13     return X_mp
14 end
```

realizes this behavior for a tournament selection with replacement. Notice that this algorithm is equivalent to Algorithm 2.16 if $p = 1$.

2.4.4 Crowded Tournament Selection

Crowded tournament selection is applied directly on optimal sets. It only relies on the crowding density estimate (see Section 35.8.3 on page 568) and is basically one example for fitness assignment process based tournament selection. It is used in algorithms where either no prevalence information is needed, for example in hill climbing or simulated annealing (see Chapter 8 and Chapter 10) or as sub-algorithm for higher level selection methods such as CNSGA-Selection in Section 2.4.10 on page 92. It is defined by Equation 2.29 and Equation 2.30.

$$crowdedTournamentSelect_{w,k} \equiv tournamentSelect_{w,k}^{\mathfrak{f}=\rho_{\mathfrak{c}\mathfrak{d}}} \tag{2.29}$$

$$crowdedTournamentSelect_{r,k} \equiv tournamentSelect_{r,k}^{\mathfrak{f}=\rho_{\mathfrak{c}\mathfrak{d}}} \tag{2.30}$$

Algorithm 2.17: $X_{mp} = tournamentSelect_{w1,k}(X_{sel}, n)$

Input: X_{sel} the list of individuals to select from
Input: $n \leq |X_{sel}|$ the number of individuals to be placed into the mating pool X_{mp}
Input: Implicit: k the tournament size
Input: Implicit: c_F the prevalence comparator function
Data: a, b the indexes of the tournament contestants in X_{sel}
Data: i, j counter variables
Output: X_{mp} the winners of the tournaments which now form the mating pool

```
1  begin
2      X_mp ⟵ ()
3      i ⟵ n
4      while i > 0 do
5          j ⟵ k − 1
6          a ⟵ ⌊random_u(|X_sel|)⌋
7          while j > 0 do
8              b ⟵ ⌊random_u(|X_sel|)⌋
9              if X_sel[b] ≻ X_sel[a] then a ⟵ b
10             j ⟵ j − 1
11         X_mp ⟵ addListItem(X_mp, X_sel[a])
12         X_sel ⟵ deleteListItem(X_sel, a)
13         i ⟵ i − 1
14     return X_mp
15 end
```

2.4.5 Roulette Wheel Selection

Roulette wheel selection[17], also known as proportionate selection, is one of the oldest selection methods. In this selection algorithm, an individual's chance of being selected is proportional to its fitness compared to the sum of the fitness of all individuals. Roulette wheel selection works only with scalar fitness values and therefore needs to rely on a preceding fitness assigned process if applied to multi-objective optimization.

You should have that we wrote *proportional to its fitness* ... well, in the context of this book we regard optimization processes per default as minimization. A high fitness would not be beneficial but a small one. In our case, we hence want the selection to work *inversely proportional* to the fitness values. In the roulette wheel algorithms introduced here, we do so.

Roulette wheel selection has a bad performance compared to other schemes like tournament selection [358, 359] or ranking selection [358, 352]. It is mainly included because of its fame since it was the original selection scheme for genetic algorithms defined by Holland [211].

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

Algorithm 2.18: $X_{mp} = tournamentSelect_{w2,k}(X_{sel}, n)$

Input: X_{sel} the list of individuals to select from
Input: $n \leq \frac{|X_{sel}|}{k}$ the number of individuals to be placed into the mating pool $|X_{mp}|$
Input: Implicit: k the tournament size
Input: Implicit: c_F the prevalence comparator function
Data: a, b the indexes of the tournament contestants in X_{sel}
Data: i, j counter variables
Output: X_{mp} the winners of the tournaments which now form the mating pool

```
1  begin
2      X_mp ⟵ ()
3      i ⟵ n
4      while i > 0 do
5          j ⟵ k − 1
6          a ⟵ ⌊random_u(|X_sel|)⌋
7          while j > 0 do
8              b ⟵ ⌊random_u(|X_sel|)⌋
9              if X_sel[b] > X_sel[a] then
10                 X_sel ⟵ deleteListItem(X_sel, a)
11                 a ⟵ b
12             j ⟵ j − 1
13         X_mp ⟵ addListItem(X_mp, X_sel[a])
14         X_sel ⟵ deleteListItem(X_sel, a)
15         i ⟵ i − 1
16     return X_mp
17 end
```

Again, we provide a version with replacement in Algorithm 2.20 and one without replacement in Algorithm 2.21.

2.4.6 Linear and Polynomial Ranking Selection

In polynomial ranking selection [205, 220], the probability for an individual to be selected is proportional to (a power of) its position (rank) in the sorted list of all individuals that can be selected. Therefore, it does not depend on a fitness assignment process. The implicit parameter $p \in \mathbb{R}^+$ of the polynomial ranking selection denotes the strictness of the selection: the bigger p gets, the higher is the probability that individuals which are non-prevailed i. e. have good objective values will be selected. $p < 1$ reverses that tendency: bad individuals will be preferred. Of course, we only speak of *linear* ranking if $p = 1$. For all other cases, the expression polynomial ranking is more appropriate. Since linear ranking is the most wide spread and analyzed version of polynomial ranking [352, 362, 358], we will focus on it here. Algorithm 2.22

Algorithm 2.19: $X_{mp} = ndTournamentSelect^{p}_{r,k}(X_{sel}, n)$

Input: X_{sel} the list of individuals to select from
Input: n the number of individuals to be placed into the mating pool X_{mp}
Input: Implicit: k the tournament size
Input: Implicit: $p \in [0, 1]$ the selection probability
Input: Implicit: c_F the prevalence comparator function
Input: A the tournament
Data: i, j counter variables
Output: X_{mp} the winners of the tournaments which now form the mating pool

```
1  begin
2      X_mp ⟵ ()
3      i ⟵ n
4      while i > 0 do
5          j ⟵ k
6          A ⟵ ()
7          while j > 0 do
8              A ⟵ addListItem(A, ⌊random_u(|X_sel|)⌋)
9              j ⟵ j − 1
10         A ⟵ sort_a(A, s(x_1, x_2) ≡ f(x_1) − f(x_2)))
11         j ⟵ 0
12         while j < k do
13             if random_u() < p then
14                 X_mp ⟵ addListItem(X_mp, A[j])
15                 j = ∞
16             j ⟵ j + 1
17         if j = k then X_mp ⟵ addListItem(X_mp, A[j])
18         i ⟵ i − 1
19     return X_mp
20 end
```

demonstrates how linear ranking selection with replacement works, ordered selection without replacement is described in Algorithm 2.23.

Exactly like tournament selection, polynomial ranking selection can use the prevalence comparator as well as a previous fitness assignment process.

2.4.7 VEGA Selection

The Vector Evaluated Genetic Algorithm's selection method [363, 364] is the first selection algorithm invented for multi-objective optimization. Here, one subpopulation of $\frac{p}{|F|}$ individuals is selected from the main population $|X_{sel}| = p$ for each objective $f_i \in F$ to be optimized. This subpopulation selection is performed using a proportional secondary selection scheme like roulette wheel

Algorithm 2.20: $X_{mp} = rouletteSelect_r(X_{sel}, n)$

Input: X_{sel} the list of individuals to select from
Input: n the number of individuals to be placed into the mating pool X_{mp}
Input: Implicit: $\mathfrak{f}$ the fitness assignment function
Data: i, j counter variables, randomized index
Data: z a temporary storage for the fitness values to be minimized
Data: d the fitness sum
Output: X_{mp} the mating pool

```
1  begin
2      X_sel ⟵ sort_a(X_sel)
3      z ⟵ createList(|X_sel|, 0
4      i ⟵ |X_sel| − 1
5      ofs ⟵ f(X_sel[|X_sel| − 1])
6      while (i ≥ 0) ∧ (ofs ≤ f(X_sel[i])) do i ⟵ i − 1
7      if (i < 0) ∨ (ofs ≤ f(X_sel[i])) then ofs ⟵ 0
8      else ofs ⟵ ofs + 0.5 ∗ (ofs − f(X_sel[i]))
9      i ⟵ 0
10     d ⟵ 0
11     while i < |X_sel| do
12         d ⟵ d + ofs − f(X_sel[i])
13         z[i] ⟵ d
14         i ⟵ i + 1
15     X_mp ⟵ ()
16     i ⟵ n
17     while i > 0 do
18         j ⟵ search_as(random_u(0, d), z)
19         if j < 0 then j ⟵ −(j + 1)
20         X_mp ⟵ addListItem(X_mp, X_sel[j])
21     return X_mp
22 end
```

selection (see Section 2.4.5 on page 84). The mating pool returned is then a mixture of these subpopulations.

2.4.8 𝕄IDEA Selection

The selection scheme applied in the 𝕄IDEA-algorithm [338] is very elitist since it only selects optimal individuals. If the optimal set is too large, a subset of maximum diversity is returned. This subset is created by first choosing one individual which is best in respect to a randomly drawn objective function (lines 4 and 5 in Algorithm 2.25). This drawing may be performed by taking all individuals and assuming that all objectives other than i are 0 and then applying prevalence comparator c_F. If the first individual x is chosen, it is placed into the mating pool. For each other individual, the distance to x is

Algorithm 2.21: $X_{mp} = rouletteSelect_w(X_{sel}, n)$

Input: X_{sel} the list of individuals to select from
Input: $n < |X_{sel}|$ the number of individuals to be placed into the mating pool X_{mp}
Input: Implicit: $\mathfrak{f}$ the fitness assignment function
Data: i, j, k counter/index variables
Data: z a temporary storage for the fitness values to be minimized
Data: d, δ fitness value sum calculation variables
Output: X_{mp} the mating pool

```
1  begin
2      X_sel ⟵ sort_a(X_sel)
3      z ⟵ createList(|X_sel|, 0)
4      i ⟵ |X_sel| − 1
5      ofs ⟵ f(X_sel[|X_sel| − 1])
6      while (i ≥ 0) ∧ (ofs ≤ f(X_sel[i])) do i ⟵ i − 1
7      if (i < 0) ∨ (ofs ≤ f(X_sel[i])) then ofs ⟵ 0
8      else ofs ⟵ ofs + 0.5 ∗ (ofs − f(X_sel[i]))
9      i ⟵ 0
10     d ⟵ 0
11     while i < |X_sel| do
12         d ⟵ d + ofs − f(X_sel[i])
13         z[i] ⟵ d
14         i ⟵ i + 1
15     X_mp ⟵ ()
16     i ⟵ n
17     while i > 0 do
18         j ⟵ search_as(random_u(d, 0), z)
19         if j < 0 then j ⟵ −(j + 1)
20         X_mp ⟵ addListItem(X_mp, X_sel[j])
21         if j > 0 then δ ⟵ z[j] − z[j − 1]
22         else δ ⟵ z[j]
23         if j < |X_sel| − 1 then
24             k ⟵ j
25             while k < |X_sel| do
26                 z[k] ⟵ z[k + 1] − δ
27                 X_sel[k] ⟵ X_sel[k + 1]
28         d ⟵ d − δ
29         X_sel ⟵ deleteListItem(X_sel, |X_sel| − 1)
30         z ⟵ deleteListItem(z, |z| − 1)
31         i ⟵ i − 1
32     return X_sel[i]
33 end
```

Algorithm 2.22: $X_{mp} = polynomialRankingSelect_{r,p}(X_{sel}, n)$

Input: X_{sel} the list of individuals to select from
Input: n the number of individuals to be placed into the mating pool X_{mp}
Input: $p \in \mathbb{R}$ the power value to be used for ordering
Input: Implicit: c_F the prevalence comparator function
Data: i a counter variable
Output: X_{mp} the individuals selected

```
1 begin
2     X_mp ⟵ ()
3     X_sel ⟵ sort_a(X_sel, c_F)
4     i ⟵ n − 1
5     while i ≥ 0 do
6         X_mp ⟵ addListItem(X_mp, X_sel [⌊(random_u())^p ∗ |X_sel|⌋])
7         i ⟵ i − 1
8     return X_mp
9 end
```

Algorithm 2.23: $X_{mp} = polynomialRankingSelect_{w,p}(X_{sel}, n)$

Input: X_{sel} the list of individuals to select from
Input: $n \leq |X_{sel}|$ the number of individuals to be placed into the mating pool X_{mp}
Input: $p \in \mathbb{R}$ the power value to be used for ordering
Input: Implicit: c_F the prevalence comparator function
Data: i a counter variable
Data: j the index of the element to be selected next
Output: X_{mp} the individuals selected

```
1  begin
2      X_mp ⟵ ()
3      X_sel ⟵ sort_a(X_sel, c_F)
4      i ⟵ n − 1
5      while i ≥ 0 do
6          j ⟵ ⌊(random_u())^p ∗ |X_sel|⌋
7          X_mp ⟵ addListItem(X_mp, X_sel[j])
8          X_sel ⟵ deleteListItem(X_sel, j)
9          i ⟵ i − 1
10     return X_mp
11 end
```

Algorithm 2.24: $X_{mp} = vegaSelect(X_{sel}, n)$

Input: X_{sel} the list of individuals to select from
Input: n the number of individuals to be placed into the mating pool X_{mp}
Input: Implicit: F the set of objective functions f_i
Data: i a counter variable
Output: X_{mp} the individuals selected

```
1 begin
2     X_mp ⟵ ()
3     i ⟵ |F|
4     while i > 0 do
5         X_mp ⟵ appendList(X_mp, rouletteWheelSelect^{ƒ=f_i}(X_sel, n/|F|))
6         i ⟵ i − 1
7     return X_mp
8 end
```

computed. The individual which is the farthest away from x is to be included next. After doing so, the distance of the rest of the selectable individuals to the mating pool are updated if they are nearer to the newly included element. This repeated until the mating pool is filled. Notice that MIDEA-selection is always without replacement and may result in a mating pool with less than n individuals - n now is a mere upper bound of its size ($|X_{mp}| \leq n$).

2.4.9 NPGA Selection

This selection scheme was introduced by Jeffrey Horn, Nicholas Nafpliotis, and David E. Goldberg for their Niched Pareto Genetic Algorithm in [345] as extension of normal tournament selection. Each time one new individual should be included in the mating pool, two candidates (x_1, x_2) are picked randomly. Then, these two are compared to an also randomly picked set of v individuals x_{tst}. If one of the two individuals x_1 or x_2 is not prevailed by any of the individuals x_{tst} while the other one is, it is selected. If either both are prevailed or non-prevailed, the one with the less other individuals in X_{sel} in its niche is used. This niching method is implicitly performed by the fitness assignment process ƒ which assigns the niche size as scalar fitness to each individual (see Section 2.3.6). Of course, any other fitness assignment process then this intended one could be used. In Algorithm 2.26 we introduce NPGA selection with replacement, Algorithm 2.27 on page 93 presents NPGA selection without replacement. Notice that this selection scheme can easily be extended to use a candidate set of the size k instead of two as propagated in the original work.

The NPGA selection scheme uses both, the prevalence comparator as well as a fitness assignment process at the same time.

Algorithm 2.25: $X_{mp} = mideaSelect(X_{sel}, n)$

Input: X_{sel} the list of individuals to select from
Input: n the number of individuals to be placed into the mating pool X_{mp}
Input: Implicit: c_F the prevalence comparator function and therefore:
Implifict F the objective functions
Data: i, j counter variables
Data: X_l the list representation of the optimal set in X_{sel}
Data: x the individual most recently included in X_{mp}
Data: dl the list assigning a distance from the mating pool to each individual
Data: md, mi the maximum distance to the mating pool and the index of the individual with this distance
Output: X_{mp} the individuals selected

```
1  begin
2      X_l = setToList(extractOptimalSet(X_sel))
3      if |X_l| ≤ n then return X_l
4      i ⟵ ⌈random_u(|F|)⌉
5      x ⟵ opt{f_i(x) ∀x ∈ X_l}
6      X_mp ⟵ createList(1, x)
7      X_l ⟵ remove(|X_l|, x)
8      dl ⟵ createList(|X_l|, ∞)
9      i ⟵ n − 1
10     while i > 0 do
11         md ⟵ −∞
12         j ⟵ |X_l| − 1
13         while j ≥ 0 do
14             if dist(X_l[j], x) < dl[j] then dl[j] ⟵ dist(X_l[j], x)
15             if dl[j] > md then
16                 md ⟵ dl[j]
17                 mi ⟵ j
18             j ⟵ j − 1
19         x ⟵ X_l[mi]
20         X_l ⟵ deleteListItem(X_l, mi)
21         dl ⟵ deleteListItem(d_l, mi)
22         X_mp ⟵ addListItem(X_mp, x)
23         i ⟵ i − 1
24     return X_mp
25 end
```

Algorithm 2.26: $X_{mp} = npgaSelect_{r,v}(X_{sel}, n)$

Input: X_{sel} the list of individuals to select from
Input: n the number of individuals to be placed into the mating pool X_{mp}
Input: v the size of the test set
Input: Implicit: c_F the prevalence comparator function
Data: i, j counter variables
Data: x_{tst} the test individual
Data: x_1, x_2 the candidates for inclusion into X_{mp}
Data: d_1, d_2 boolean variables checking if x_1, x_2 are prevailed
Output: X_{mp} the individuals selected

```
1  begin
2      X_mp ⟵ ()
3      i ⟵ n − 1
4      while i ≥ 0 do
5          X_tst ⟵ ∅
6          x_1 ⟵ X_sel[⌊random_u(|X_sel|)⌋]
7          x_2 ⟵ X_sel[⌊random_u(|X_sel|)⌋]
8          d_1 ⟵ false
9          d_2 ⟵ false
10         j ⟵ v
11         while j > 0 do
12             x_tst ⟵ X_sel[⌊random_u(|X_sel|)⌋]
13             if x_tst ≻ x_1 then d_1 ⟵ true
14             if x_tst ≻ x_2 then d_2 ⟵ true
15             j ⟵ j − 1
16         if ¬d1 ∧ d2 then X_mp ⟵ addListItem(X_mp, x_1)
17         else if d1 ∧ ¬d2 then X_mp ⟵ addListItem(X_mp, x_2)
18         else if f(x_1) > f(x_2) then X_mp ⟵ addListItem(X_mp, x_1)
19         else X_mp ⟵ addListItem(X_mp, x_2)
20         i ⟵ i − 1
21     return X_mp
22 end
```

2.4.10 CNSGA Selection

The Controlled Non-dominated Sorting Genetic Algorithm (CNSGA) [365] by Deb and Goell uses a selection scheme which preserves individual diversity along the optimal (non-prevailed) frontiers as well as in depth of the prevailed individuals. It extends the measures applied in NSGA2-fitness assignment processes (see Section 2.3.8 on page 73). Basing on the $prevalenceFitnessAssign_2$ fitness assignment process (Section 2.3.2 on page 66), the frontiers where each element has the same integer fitness are enumerated. This will be the non-dominated frontiers Z_i, which can be defined by:

$$Z_0 = extractOptimalSet(X_{sel}) \tag{2.31}$$

Algorithm 2.27: $X_{mp} = npgaSelect_{w,v}(X_{sel}, n)$

Input: X_{sel} the list of individuals to select from
Input: $n \leq |X_{sel}|$ the number of individuals to be placed into the mating pool X_{mp}
Input: v the size of the test set
Input: Implicit: c_F the prevalence comparator function
Data: i, j counter variables
Data: x_{tst} the test individual
Data: x_1, x_2 the candidates for inclusion into X_{mp}
Data: d_1, d_2 boolean variables checking if x_1, x_2 are prevailed
Output: X_{mp} the individuals selected

```
1  begin
2      X_mp ⟵ ()
3      i ⟵ n − 1
4      while i ≥ 0 do
5          X_tst ⟵ ∅
6          x_1 ⟵ X_sel[⌊random_u(|X_sel|)⌋]
7          x_2 ⟵ X_sel[⌊random_u(|X_sel|)⌋]
8          d_1 ⟵ false
9          d_2 ⟵ false
10         j ⟵ v
11         while j > 0 do
12             x_tst ⟵ X_sel[⌊random_u(|X_sel|)⌋]
13             if x_tst ≻ x_1 then d_1 ⟵ true
14             if x_tst ≻ x_2 then d_2 ⟵ true
15             j ⟵ j − 1
16         if ¬(¬d1 ∧ d2) then
17             if (d1 ∧ ¬d2) ∨ (𝔣(x_2) > 𝔣(x_1)) then x_1 ⟵ x_2
18         X_sel ⟵ removeListItem(X_sel, x_1)
19         X_mp ⟵ addListItem(X_mp, x_1)
20         i ⟵ i − 1
21     return X_mp
22 end
```

$$Z_1 = extractOptimalSet(X_{sel} \setminus Z_0) \tag{2.32}$$
$$Z_2 = extractOptimalSet(X_{sel} \setminus (Z_0 \cup Z_1)) \tag{2.33}$$
$$\dots$$
$$Z_i = extractOptimalSet\left(X_{sel} \setminus \left(\cup_{j=0}^{i-1} Z_j\right)\right) \tag{2.34}$$
$$\tag{2.35}$$

It is attempted to maintained an adaptively computed number p_i of individuals from each such front Z_i, where p_i is computed using the geometric distribution:

$$p_i = v\, p_{i-1} (v \in [0, 1)) \tag{2.36}$$

p_i is the maximum number of individuals allowed in the ith frontier F. If K is the total count of frontiers in the set of selectable individuals X_{sel} and the first frontier has the index 0 (thus $i \in \{0, 1, 2, \ldots, K-1\}$), the count of individuals allowed in frontier Z_i can be computed by

$$p_i = \lfloor |X_{sel}| \frac{1-v}{1-v^K} v^i + 0.5 \rfloor \tag{2.37}$$

The number of individuals selected from the first front is highest, decreasing exponentially with i. Of course, it may be possible that in a frontier j are not enough individuals for selection ($|Z_j| < p_j$). If that is the case, the number of remaining individuals is used as a *bonus* of additional individuals allowed in the next front, therefore changing the equation of p_i:

$$bonus_0 = 0 \tag{2.38}$$

$$bonus_i = \max\{0, p_{i-1} - |Z_{i-1}|\} \tag{2.39}$$

$$p_i = \lfloor |X_{sel}| \frac{1-v}{1-v^K} v^i + 0.5 \rfloor + bonus_i \tag{2.40}$$

If there are more individuals in Z_i than needed ($|Z_i| > p_i$), a secondary selection algorithm, in the original paper crowded tournament selection $crowdedTournamentSelect_2$ (see Section 2.4.4 on page 83) is applied. Crowded tournament selection serves here to preserve diversity. This special selection algorithm solely bases on the crowding distance $\mathfrak{cd}$ and the density estimate $\rho_{\mathfrak{cd}}$ (see Section 35.8.3 on page 568). It is still possible that we cannot fill up the mating pool with one pass of the selection algorithm, since it may happen that we have very much individuals in the first fronts and only very few in the last frontiers. If that is the case, the whole process is simple repeated. We have specified CNSGA selection with replacement in Algorithm 2.28 and without replacement in Algorithm 2.29.

2.4.11 PESA Selection

The selection algorithm employed by PESA [366] relies strongly on the adaptive grid archiving technique which is introduced as Algorithm 1.6 on page 38 in this book. Basically, PESA selection selects only individuals of the optimal set using a binary tournament where the individual in the less crowded grid region wins and ties are broken randomly. For more information about how the hyper-grid used is generated, take a look at Section 1.7.3 on page 36. The PESA selection can be performed with (in Algorithm 2.30) and without (in Algorithm 2.31) replacement, but usually works directly on the objective function values since these are internally used to build the hyper-grid.

2.4.12 PESA-II Selection

In PESA-II [367], selection is no longer performed individual- but hyper-box based. This selection algorithm is a two-level method: first, the individuals

Algorithm 2.28: $X_{mp} = cnsgaSelect^{\mathfrak{f}}_{r,v}(X_{sel}, n)$

Input: X_{sel} the list of individuals to select from
Input: n the number of individuals to be placed into the mating pool X_{mp}
Input: Implicit: v the frontier ratio
Data: Z the remaining/optimal front/unused individual lists
Data: i, K the front counter and the expected count of fronts
Data: j a counter variable
Data: m, p the remaining/current count of individuals to select
Data: $bonus$ the count of individuals missing in the previous fronts
Data: $select_r$ the secondary selection scheme, normally $crowdedTournamentSelect_r$
Output: X_{mp} the individuals selected

```
1  begin
2      X_mp ⟵ ()
3      X_sel ⟵ sort_a(X_sel, s(x_1, x_2) ≡ 𝔣(x_1) − 𝔣(x_2))
4      K ⟵ ⌊𝔣(X_sel[|X_sel| − 1])⌋ − ⌊𝔣(X_sel[0])⌋ + 1
5      while |X_mp| < n do
6          i ⟵ 0
7          j ⟵ 0
8          bonus ⟵ 0
9          m ⟵ n − X_mp
10         while (|X_sel| > 0) ∧ (|X_mp| < n) do
11             Z ⟵ ()
12             f ⟵ ⌊𝔣(X_sel[j])⌋
13             while (j < |X_sel|) ∧ (f = ⌊𝔣(X_sel[j])⌋) do
14                 Z ⟵ listAdd(Z, X_sel[j])
15                 j ⟵ j + 1
16             p ⟵ max{⌊m (1−v)/(1−v^K) v^i + 0.5⌋ + bonus, 1}
17             if |Z| ≤ p then
18                 X_mp ⟵ appendList(X_mp, Z)
19                 bonus ⟵ |Z| − p
20             else
21                 X_mp ⟵ appendList(X_mp, select_r(listToSet(Z), p))
22                 bonus ⟵ 0
23             i ⟵ i + 1
24     return X_mp
25 end
```

Algorithm 2.29: $X_{mp} = cnsgaSelect^{\mathfrak{f}}_{w,v}(X_{sel}, n)$

Input: X_{sel} the list of individuals to select from
Input: $n \leq |X_{sel}|$ the number of individuals to be placed into the mating pool X_{mp}
Input: Implicit: v the frontier ratio
Data: Z, G the remaining/optimal front/unused individual lists
Data: i, K the front counter and the expected count of fronts
Data: m, p the remaining/current count of individuals to select
Data: $bonus$ the count of individuals missing in the previous fronts
Data: $select_w$ the secondary selection scheme, normally $crowdedTournamentSelect_w$
Output: X_{mp} the individuals selected

begin
  $X_{mp} \longleftarrow ()$
  $X_{sel} \longleftarrow sort_a(X_{sel}, s(x_1, x_2) \equiv \mathfrak{f}(x_1) - \mathfrak{f}(x_2))$
  while $|X_{mp}| < n$ **do**
    $G \longleftarrow ()$
    $i \longleftarrow 0$ $bonus \longleftarrow 0$
    $K \longleftarrow \lfloor \mathfrak{f}(X_{sel}[|X_{sel}| - 1]) \rfloor - \lfloor \mathfrak{f}(X_{sel}[0]) \rfloor + 1$
    $m \longleftarrow n - X_{mp}$
    while $(|X_{sel}| > 0) \wedge (|X_{mp}| < n)$ **do**
      $Z \longleftarrow ()$
      $f \longleftarrow \lfloor \mathfrak{f}(X_{sel}[0]) \rfloor$
      while $(|X_{sel}| > 0) \wedge (f = \lfloor \mathfrak{f}(X_{sel}[0]) \rfloor)$ **do**
        $Z \longleftarrow listAdd(Z, X_{sel}[0])$
        $X_{sel} \longleftarrow listDelete(X_{sel}[0])$
      $p \longleftarrow \max\{\lfloor m \frac{1-v}{1-v^K} v^i + 0.5 \rfloor + bonus, 1\}$
      if $|Z| \leq p$ **then**
        $X_{mp} \longleftarrow appendList(X_{mp}, Z)$
        $bonus \longleftarrow |Z| - p$
      else
        $X_{mp} \longleftarrow appendList(X_{mp}, select_w(listToSet(Z), p))$
        $bonus \longleftarrow 0$
      $G \longleftarrow appendList(G, Z)$
      $i \longleftarrow i + 1$
    $X_{sel} \longleftarrow G$
  return X_{mp}
end

Algorithm 2.30: $X_{mp} = pesaSelect_r(X_{sel}, n)$

Input: X_{sel} the list of individuals to select from
Input: n the number of individuals to be placed into the mating pool X_{mp}
Data: lst a list assigning grid coordinates to the elements of X_{sel}
Data: cnt containing the count of elements in the grid locations defined in lst
Data: a, b, i index variables
Output: X_{mp} the individuals selected

```
1  begin
2      X_mp ⟵ ()
3      (X_sel, lst, cnt) ⟵ agaDivide(extractOptimalSet(X_sel), d)
4      (lst, cnt) ⟵ agaNormalize(lst, cnt)
5      i ⟵ n
6      while i > 0 do
7          a ⟵ ⌊random_u(|X_sel|)⌋
8          b ⟵ ⌊random_u(|X_sel|)⌋
9          if cnt[b] < cnt[a] then a ⟵ b
10         X_mp ⟵ addListItem(X_mp, X_sel[a])
11         i ⟵ i − 1
12     return X_mp
13 end
```

Algorithm 2.31: $X_{mp} = pesaSelect_w(X_{sel}, n)$

Input: X_{sel} the list of individuals to select from
Input: $n < |X_{sel}|$ the number of individuals to be placed into the mating pool X_{mp}
Data: lst a list assigning grid coordinates to the elements of X_{sel}
Data: cnt containing the count of elements in the grid locations defined in lst
Data: a, b, i index variables
Output: X_{mp} the individuals selected

```
1  begin
2      X_mp ⟵ ()
3      (X_sel, lst, cnt) ⟵ agaDivide(extractOptimalSet(X_sel), d)
4      (lst, cnt) ⟵ agaNormalize(lst, cnt)
5      i ⟵ n
6      while i > 0 do
7          a ⟵ ⌊random_u(|X_sel|)⌋
8          b ⟵ ⌊random_u(|X_sel|)⌋
9          if cnt[b] < cnt[a] then a ⟵ b
10         X_mp ⟵ addListItem(X_mp, X_sel[a])
11         X_sel ⟵ deleteListItem(X_sel, a)
12         cnt ⟵ deleteListItem(cnt, a)
13         i ⟵ i − 1
14     return X_mp
15 end
```

are divided into hyper-boxes according to Algorithm 1.6 on page 38 and a secondary selection is performed on these hyper-boxes on line 12. Therefore, any fitness assignment process based selection could be chosen (be default, a binary tournament selection is used). As fitness subject to minimization, the inhabitant count is assigned to each box. From each box chosen by the secondary selection algorithm, one individual is drawn by random. This selection method is only defined as algorithm with replacement and only selection algorithms with replacement should be chosen as secondary selection schemes. Like the PESA-selection, this selection algorithm is also strictly elitist.

Algorithm 2.32: $X_{mp} = pesa2Select(X_{sel}, n)$

Input: X_{sel} the list of individuals to select from
Input: n the number of individuals to be placed into the mating pool X_{mp}
Input: Implicit: $select^{\mathfrak{f}}$ a fitness assignment based selection scheme to be used for hyper-box selection, binary tournament selection by default
Data: lst a list assigning grid coordinates to the elements of X_{sel}
Data: cnt containing the count of elements in the grid locations defined in lst
Data: i, j index variables
Data: X'_{sel}, X'_{mp} the hyper-box selection and mating pools
Data: $\mathfrak{f}$ a fitness function that assigns the count of occupying individuals to the hyper-boxes
Output: X_{mp} the individuals selected

```
1  begin
2      X_mp ⟵ ()
3      (X_sel, lst, cnt) ⟵ agaDivide(extractOptimalSet(X_sel), d)
4      (lst, cnt) ⟵ agaNormalize(lst, cnt)
5      X'_sel ⟵ ∅
6      i ⟵ |lst| − 1
7      while i ≥ 0 do
8          if lst[i] ∉ X'_sel then
9              X'_sel ⟵ X'_sel ∪ lst[i]
10             f(lst[i]) ⟵ cnt[i] + 2
11         i ⟵ i − 1
12     X'_mp ⟵ select^f(X'_sel, n)
13     i ⟵ |X'_mp|
14     while i > 0 do
15         j ⟵ ⌊random_u(|X_sel|)⌋
16         while lst[j] ≠ X'_mp[i] do  j ⟵ (j + 1) mod |X_sel|
17         X_mp ⟵ listAdd(X_mp, X_sel[j])
18         i ⟵ i − 1
19     return X_mp
20 end
```

2.4.13 Prevalence/Niching Selection

Fernando Jiménez et al. introduced in [224] another interesting selection algorithm [368]. Although it is used for constraint-based genetic algorithms, the constraint-related parts can be striped so only a simple selection scheme remains in Algorithm 2.33. The basic idea is to draw two individuals a and b from the selectable set X_{sel}. If one of them prevails over the other, the non-prevailed individual is included into the mating pool. If neither one is prevailing, then a test set X_t of the size k is drawn. If a is non-prevailed by this test set but b is, then a is put into the mating pool. The same goes for b vice versa. If again either both are prevailed or non-prevailed, the one of them with the smaller niche count $m(x, X_{sel})$ wins (see Section 2.3.5 on page 69).

2.5 Reproduction

Optimization algorithms use the information gathered up to step t to create the solution candidates to be evaluated in step $t + 1$. There exist different methods to do so, but basically, they can be reduced to four reproduction operations. Although their names are strongly inspired by genetic algorithms and the biological reproduction mechanisms[18] of mother nature [187], the definitions given in this chapter are general enough to fit for all global optimization algorithms. Note that all the operations defined may application-dependently be implemented in a deterministic or a randomized way.

In the following definitions, the operators are applied to elements x of the solution space $\tilde{X}$. This is not always the case – in genetic algorithms for example, they will work on the genotypes $g \in \mathbb{G}$ which are only an intermediate representation of the phenotypes x. In these cases, we assume that an implicit translation takes place.

Definition 42 (Creation). The creation operation *create* is used to produce a new solution candidate which is not related to the existing ones. When starting up the optimization process, this operation may be used to create randomized individuals.

$$x_{new} = create(),\ x_{new} \in \tilde{X} \tag{2.41}$$

Definition 43 (Duplication). The duplication operation *duplicate* is used to create an exact copy of an existing solution candidate x. Duplication may be useful to increase the share of a given type of individual in a population for population-based algorithms or if the evaluation criteria have changed.

$$x_{new} = x = duplicate(x)\ :\ x \in \tilde{X},\ x_{new} \in \tilde{X} \tag{2.42}$$

[18] `http://en.wikipedia.org/wiki/Reproduction` [accessed 2007-07-03]

Algorithm 2.33: $X_{mp} = prevalenceNicheSelect(X_{sel}, n)$

Input: X_{sel} the list of individuals to select from
Input: n the number of individuals to be placed into the mating pool X_{mp}
Input: Implicit: k the test set size
Data: X_t the test set
Data: a, b individuals competing against each other
Data: i, j index variables
Output: X_{mp} the individuals selected

```
1  begin
2      X_mp ⟵ ()
3      i ⟵ n
4      X_sel ⟵ setToList(X_sel)
5      while i > 0 do
6          a ⟵ X_sel[⌊random_u(|X_sel|)⌋]
7          b ⟵ X_sel[⌊random_u(|X_sel|)⌋]
8          if a ≻ b then
9              X_mp ⟵ listAdd(X_mp, a)
10         else if b ≻ a then
11             X_mp ⟵ listAdd(X_mp, b)
12         else
13             j ⟵ k
14             X_t ⟵ ∅
15             while j > 0 do
16                 X_t ⟵ X_t ∪ X_sel[⌊random_u(|X_sel|)⌋]
17                 j ⟵ j − 1
18             if ¬∃x ∈ X_t : x ≻ a ∧ (∃x ∈ X_t : x ≻ b) then
19                 X_mp ⟵ listAdd(X_mp, a)
20             else if ∃x ∈ X_t : x ≻ a ∧ ¬∃x ∈ X_t : x ≻ b then
21                 X_mp ⟵ listAdd(X_mp, b)
22             else
23                 if m(a, X_sel) < m(b, X_sel) then X_mp ⟵ listAdd(X_mp, a)
24                 else X_mp ⟵ listAdd(X_mp, b)
25         i ⟵ i − 1
26     return X_mp
27 end
```

Definition 44 (Mutation). The mutation[19] operation *mutate* is used to create a new solution candidate by modifying an existing one. This modification may application-dependently happen in a randomized or in a deterministic fashion.

$$x_{new} = mutate(x) \ : \ x \in \tilde{X}, \ x_{new} \in \tilde{X} \tag{2.43}$$

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

Definition 45 (Crossover). The crossover[20] (or recombination[21]) operation *crossover* is used to create a new solution candidate by combining the features of two existing ones. This modification may application-dependently happen in a randomized or in a deterministic fashion.

$$x_{new} = crossover(x_1, x_2) \ : \ x_1, x_2 \in \tilde{X}, \ x_{new} \in \tilde{X} \tag{2.44}$$

Notice that the term recombination is more general than crossover (which is often used for linear representations of solution candidates only) and is therefore maybe a better choice for the function that produces and offspring of two parents. Crossover is however more common and therefore used here.

These operations may be combined arbitrarily, $mutate\left(crossover\left(x_1, x_2\right)\right)$ for example may produce a mutated offspring of x_1 and x_2. All operators together are used to reproduce whole populations of individuals.

Definition 46 (*reproducePop*). While a single *reproducePop*-function creates exactly one solution candidate, we introduce the $reproducePop(X_{mp}, k)$ operation which creates a new population of k individuals from a list of X_{mp} individuals (the mating pool, obtained for example from one of the selection operators, see Section 2.4).

$$X_{new} \ = \ reproducePop(X_{mp}, k) \tag{2.45}$$

$$\forall x_{old} \in X_{mp} \Rightarrow x_{old} \in \tilde{X} \tag{2.46}$$

$$|X_{new}| \ = \ k \tag{2.47}$$

$$\forall x_{new} \in X_{new} \Rightarrow x_{new} \in \tilde{X} \tag{2.48}$$

$$\forall x_{old} \in X_{mp} \ \exists \ x_{new} \in \tilde{X} : reproduce(x_{old}) = x_{new} \tag{2.49}$$

Furthermore, we define a macro for creating a set of n random individuals:

2.5.1 NCGA Reproduction

The NCGA evolutionary algorithm [369] uses a special reproduction method. Crossover is performed only neighboring individuals which will lead to child individuals close to their parents. This so-called neighborhood cultivation shifts the crossover/recombination-operator more into exploitation. It can furthermore be argued that parents that do not differ very much from each other are more likely to be compatible in order to produce functional offspring than parents that have nothing in common. Neighborhood cultivation is achieved in Algorithm 2.35 by sorting the mating pool along one *focused* objective and then recombine elements situated directly besides each other. The focus on the objective rotates in a way that in a three-objective optimization the first

[20] http://en.wikipedia.org/wiki/Crossover_%28genetic_algorithm%29 [accessed 2007-07-03]

[21] http://en.wikipedia.org/wiki/Recombination [accessed 2007-07-03]

Algorithm 2.34: $X_{pop} = createPop(n)$

Input: n the size of the population to be created
Data: i a counter variable
Output: X_{pop} the new, random population ($|X_{pop}| = n$)

```
1 begin
2     X_pop ⟵ ()
3     i ⟵ n
4     while i > 0 do
5         X_pop ⟵ appendListItem(X_pop, create())
6         i ⟵ i − 1
7     return X_pop
8 end
```

objective is focused at the beginning, then the second, then the third and after that again the first. The algorithm shown here has the implicit parameter foc denoting that focused parameter. The original publication is not very clear about how the mutation is applied, so we simple mutate each individual (see line 12), which could be changed in any real implementation.

Algorithm 2.35: $X_{pop} = ncgaReproducePop_{foc}(X_{mp}, p)$

Input: X_{mp} the mating pool list
Input: Implicit: p the count of individuals to be created
Input: Implicit: foc the objective currently focussed
Input: Implicit: *crossover*, *mutate* - crossover and mutation routines
Output: X_{pop} the new population

```
1  begin
2      X_mp ⟵ sort_a(X_mp, s(x_1, x_2) ≡ f_foc(x_1) − f_foc(x_2))
3      X_p ⟵ ()
4      i ⟵ p − 1
5      while i ≥ 0 do
6          X_p ⟵ appendListItem(X_p, crossover(X_mp [i mod |X_mp|],
7                                              X_mp [(i + 1) mod |X_mp|]))
8          i ⟵ i − 1
9      X_mp ⟵ ()
10     i ⟵ |X_p| − 1
11     while i ≥ 0 do
12         X_mp ⟵ appendListItem(X_mp, mutate(X_p[i]))
13         i ⟵ i − 1
14     return X_pop
15 end
```

2.6 Algorithms

Besides the standard evolutionary algorithms introduced in Section 2.1.1 on page 51, there exists a variety of other, more sophisticated approaches. Many of them deal especially with multi-objectivity 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 [363, 364] invented in 1985 by Schaffer. In VEGA, the selection algorithm has been modified (see Section 2.4.7 on page 86). If it is applied to $|F|$ objectives, the selection will first create $\frac{p}{|F|}$ subpopulations, each filled with a proportional secondary selection algorithm working on one single objective. These subpopulations are then shuffled together again in order to perform reproduction. The VEGA is specified here as Algorithm 2.36.

Richardson et.al. [370, 371] argue the selection scheme of VEGA would be approximately the same as if computing a weighted sum of the fitness values [372].

Algorithm 2.36: $X^\star = vega(c_F)$

Input: c_F the comparator function which allows us to compare the fitness of two solution candidates, used by *updateOptimalSet*
Input: Implicit: p the population size
Data: X_{pop} the population
Data: X_{mp} the mating pool
Output: $X^\star \subseteq \tilde{X}$ the set of the best elements found

```
1 begin
2   X_pop ⟵ createPop(p)
3   while ¬terminationCriterion() do
4     X_mp ⟵ vegaSelect(X_pop, p)
5     X_pop ⟵ reproducePop(X_mp, p)
6   return extractOptimalSet(X_pop)
7 end
```

2.6.2 MIDEA

The naive mixture-based, multi-objective iterated density-estimation evolutionary algorithm [337, 338, 373] uses a mixture of probability distributions in order to estimate multi-objective criteria. It has a good performance in function minimization and in solving combinatorial problems [227]. In this

part of the book we are not concerned about special applications (like pure numeric parameter approximation) so we will solely concentrate on the evolutionary properties of MIDEA.

MIDEA uses a special selection scheme specified in Section 2.4.8 on page 87. In the population, the individuals which have not been selected are replaced by offspring of the selected individuals - MIDEA is thus an elitist algorithm. As far as general problem spaces are regarded, that is the major difference to other EA. The specification of (the generalized version of) MIDEA can be found in Algorithm 2.37.

Algorithm 2.37: $X^\star = midea(c_F)$

Input: c_F the comparator function which allows us to compare the fitness of two solution candidates, used by *updateOptimalSet*
Input: Implicit: p the population size
Data: X_{pop} the population
Data: X_{new} the individuals of the next generation
Data: X_{mp} the mating pool
Output: $X^\star \subseteq \tilde{X}$ the set of the best elements found

```
1 begin
2   X_pop ⟵ createPop(p)
3   while ¬terminationCriterion() do
4     X_mp ⟵ mideaSelect(X_pop, p)
5     X_new ⟵ reproducePop(X_mp, p − |X_mp|)
6     X_pop ⟵ appendList(X_mp, X_new)
7   return extractOptimalSet(X_pop)
8 end
```

2.6.3 NPGA

The Niched Pareto Genetic Algorithm for multi-objective optimization by Horn, Nafpliotis, and Goldberg [374, 345, 375] uses a special sort of tournament selection, the *npgaSelect*-algorithm (see page 90) and a fitness assignment strategy which puts pressure against crowded niches (see Section 2.3.6 on page 71) in order to obtain a broad scan of the optimal frontier. It works as described in Algorithm 2.38 on the facing page.

2.6.4 NPGA2

Erickson, Mayer and Horn improved the Niched Pareto Genetic Algorithm in order to make the Pareto domination sampling used in the tournaments less lossy [376]. In principle, they now apply a Pareto-ranking scheme, as discussed in [372] and defined as rank-based fitness assignment processes in this book

Algorithm 2.38: $X^\star = npga(c_F)$

Input: c_F the comparator function which allows us to compare the fitness of two solution candidates, used by *updateOptimalSet*
Input: Implicit: p the population size
Data: X_{pop} the population
Data: X_{mp} the mating pool
Output: $X^\star \subseteq \tilde{X}$ the set of the best elements found

```
1 begin
2 |   X_pop ⟵ createPop(p)
3 |   while ¬terminationCriterion() do
4 |   |   nichSizeFitnessAssign(X_pop, ())
5 |   |   X_mp ⟵ npgaSelect(X_pop, p)
6 |   |   X_pop ⟵ reproducePop(X_mp, p)
7 |   return extractOptimalSet(X_pop)
8 end
```

(see Section 2.3.3 on page 67). The NPGA2 utilizes (deterministic) tournament selection with the tournament size k. For each tournament, we check if one individual has the lowest rank of all k candidates. If so, it wins the tournament. Otherwise, the population density around the candidates will break the tie – the individual with the fewest other individuals in its near will win. This approach is equivalent to performing a fitness assignment process which orders the individuals according to their rank and using the values of the sharing function Sh as increments while ensuring that individuals of lower rank have always lower fitness values regardless of how crowded their surrounding is. Exactly this is done in the $nsgaFitnessAssign$ fitness assignment process, Algorithm 2.8 on page 73. The result of this specification is Algorithm 2.39 on the next page, which was used by the authors to optimize groundwater remediation systems.

2.6.5 NSGA

The non-dominated sorting genetic algorithm by Srinivas and Deb [346, 23] employs a special fitness assignment procedure (see Algorithm 2.8 on page 73) that successively removes the non-prevailed individuals from the population and relates it to fitness values using sharing to enhance diversity. After fitness assignment, the original paper mentions the use of a *stochastic remainder proportionate selection scheme* – seemingly a roulette wheel method. NSGA has been applied in a variety of multi-objective optimization problems, for example in polymer reaction engineering, catalytic reactors, membrane modules, cyclone separators and venturi scrubbers in chemical engineering [121]. The NSGA algorithm is specified here as Algorithm 2.40. Many similarities between NSGA and the NPGA2 algorithm (Section 2.6.4 on the preceding page) can be observed.

Algorithm 2.39: $X^\star = npga2(c_F)$

Input: c_F the comparator function which allows us to compare the fitness of two solution candidates, used by *updateOptimalSet*
Input: Implicit: p the population size
Data: X_{pop} the population
Data: X_{mp} the mating pool
Output: $X^\star \subseteq \tilde{X}$ the set of the best elements found

```
1 begin
2     X_pop ⟵ createPop(p)
3     while ¬terminationCriterion() do
4         nsgaFitnessAssign(X_pop, ())
5         X_mp ⟵ tournamentSelect(X_pop, p)
6         X_pop ⟵ reproducePop(X_mp, p)
7     return extractOptimalSet(X_pop)
8 end
```

Algorithm 2.40: $X^\star = nsga(c_F)$

Input: c_F the comparator function which allows us to compare the fitness of two solution candidates, used by *updateOptimalSet*
Input: Implicit: p the population size
Data: X_{pop} the population
Data: X_{mp} the mating pool
Output: $X^\star \subseteq \tilde{X}$ the set of the best elements found

```
1 begin
2     X_pop ⟵ createPop(p)
3     while ¬terminationCriterion() do
4         nsgaFitnessAssign(X_pop, ())
5         X_mp ⟵ rouletteWheelSelect(X_pop, p)
6         X_pop ⟵ reproducePop(X_mp, p)
7     return extractOptimalSet(X_pop)
8 end
```

2.6.6 NSGA2

The nondominated sorting genetic algorithm 2 by Deb, Agrawal, Pratab and Meyarivan [347] is an extension of the NSGA algorithm. It improves convergence with elitism and employs a new, improved fitness assignment process (see Algorithm 2.9 on page 74). After fitness assignment, a binary tournament selection is used. NSGA2 is specified as Algorithm 2.41 on the next page.

2.6.7 CNSGA

The Controlled Non-dominated Sorting Genetic Algorithm (CNSGA) [365] by Deb and Goell is an extension of the NSGA2. Whereas NSGA2 may

Algorithm 2.41: $X^\star = nsga2(c_F)$

Input: c_F the comparator function which allows us to compare the fitness of two solution candidates
Input: Implicit: p the population size
Data: X_{pop}, X_{old} the current and the previous population
Data: X_{mp} the mating pool
Output: X_{sel} the individuals list of selectable individuals

```
1  begin
2      X_old ⟵ ()
3      X_pop ⟵ createPop(p)
4      while ¬terminationCriterion() do
5          X_sel ⟵ appendList(X_old, X_pop)
6          nsga2FitnessAssign(X_sel, ())
7          X_sel ⟵ sort_a(X_sel, s(x1, x2) = f(x1) − f(x2))
8          if p < |X_sel| then X_sel ⟵ deleteListRange(X_sel, p, |X_sel| − p)
9          X_mp ⟵ tournamentSelect^f_{r,2}(X_sel, p)
10         X_old ⟵ X_pop
11         X_pop ⟵ reproducePop(X_mp, p)
12     return extractOptimalSet(appendList(X_pop, X_old))
13 end
```

be too explorative in some circumstances and thus loose the individual diversity to optimally solve problems, CNSGA applies a controlled approach for diversity preservation. CNSGA preserves individual diversity by a special selection algorithm (see Section 2.4.10 on page 92) which uses the $prevalenceFitnessAssign_2$ fitness assignment process (Section 2.3.2 on page 66). It preserves diversity inside the optimal frontiers by using the crowding density estimate and across the frontiers by adaptively including individuals. While the properties of the selection algorithm are described in Section 2.4.10, the rest of the algorithm remains the same as in NSGA2:

2.6.8 PAES

The Pareto Archived Evolutionary Strategy (PAES) [377] by Knowles and Corne is very similar to multi-objective hill climbing presented in Chapter 8 on page 223. It introduces a new method of pruning the optimal set [120] (see Algorithm 1.7 on page 39). An individual is included in the non-prevailed set if it is non-prevailed. If this optimal set reaches its maximum size, a new, non-prevailed individual is only included if it can replace one inside the archive that resides in a more crowded region. This means that new individuals enter the archive only if they may add diversity (see Section 1.7.3 on page 36).

It is also possible to create $(1 + \lambda)$ or $(\mu + \lambda)$ variants of PAES, which, in its pure version, is a $(1 + 1)$-algorithm and specified here as Algorithm 2.43 on the next page.

Algorithm 2.42: $X^\star = cnsga(c_F)$

Input: c_F the comparator function which allows us to compare the fitness of two solution candidates
Input: Implicit: p the population size
Data: X_{pop}, X_{old} the current and the previous population
Data: X_{mp} the mating pool
Output: X_{sel} the individuals list of selectable individuals

```
1  begin
2      X_old ⟵ ()
3      X_pop ⟵ createPop(p)
4      while ¬terminationCriterion() do
5          X_sel ⟵ appendList(X_old, X_pop)
6          prevalenceFitnessAssign_2(X_sel, ())
7          X_mp ⟵ cnsgaSelect^f_w(X_sel, p)
8          X_old ⟵ X_pop
9          X_pop ⟵ reproducePop(X_mp, p)
10     return extractOptimalSet(appendList(X_pop, X_old))
11 end
```

Algorithm 2.43: $X^\star = paes(c_F)$

Input: c_F the comparator function which allows us to compare the fitness of two solution candidates, used by *updateOptimalSet*
Data: $x_{new} \in \tilde{X}$ the new element created
Output: $X^\star \subseteq \tilde{X}$ the set of the best elements found

```
1  begin
2      X* ⟵ ∅
3      x_new ⟵ create()
4      while ¬terminationCriterion() do
5          X* ⟵ updateOptimalSet(X*, x_new)
6          X* ⟵ pruneOptimalSet_aga(X*)
7          x_new ⟵ select(setToList(X*), 1)[0]
8          x_new ⟵ mutate(x_new)
9      return X*
10 end
```

2.6.9 PESA

The Pareto Envelope-based Selection Algorithm [366] (Algorithm 2.44 on the facing page) is an evolutionary algorithm with many parallels to the PAES. It also uses a hyper-grid in order to determine if a solution is situated in a crowded region or not. PESA is very elitist by only using the optimal set for selection (see Section 2.4.11 on page 94 for more details).

Algorithm 2.44: $X^\star = pesa(c_F)$

Input: c_F the comparator function which allows us to compare the fitness of two solution candidates, used by *updateOptimalSet*
Input: Implicit: p the population size
Data: X_{pop} the population
Data: X_{mp} the mating pool
Output: $X^\star \subseteq \tilde{X}$ the set of the best elements found

```
1  begin
2    X* ⟵ ∅
3    X_pop ⟵ createPop(p)
4    while ¬terminationCriterion() do
5      foreach x ∈ X_pop do  X* ⟵ updateOptimalSet(X*, x)
6      X* ⟵ pruneOptimalSet_aga(X*)
7      X_mp ⟵ pesaSelect(X*, p)
8      X_pop ⟵ reproducePop(X_mp, p)
9    return X*
10 end
```

2.6.10 PESA-II

The Pareto Envelope-based Selection Algorithm II [367] (Algorithm 2.45 on the following page) improves PESA by using a more sophisticated selection method (see Section 2.4.12 on page 94). Basically, instead of selecting individuals, boxes in the hyper-grid are now selected basing on the count of their inhabitants. From these boxes, individuals are then drawn randomly. In all other features, PESA-II equals PESA.

2.6.11 RPSGAe

The Reduced Pareto Set Genetic Algorithm with Elitism (RPSGAe) [348, 378] improves the RPSG algorithm [379] by adding elitism in a very decent way and is specified here as Algorithm 2.46 on page 111. The algorithm works on a local population X_{pop} and an external population/archive X_{arc}. It uses a rank-based fitness assignment strategy defined in Section 2.3.9 on page 75 with a maximum number of n ranks. The archive is initially empty, but in each generation, the best $2\frac{p}{n}$ (where p is the population size) individuals from the local population X_{pop} are copied to it. If the archive reaches size $2p$, it is truncated back to size $\frac{p}{n}$, only keeping the best solution candidates. This truncated archive is now incorporated back into the local population by replacing the $\frac{p}{n}$ weakest individuals there.

2.6.12 SFGA and PSFGA

The Single Front Genetic Algorithm and its parallelized version the Parallel Single Front Genetic Algorithm are introduced in [380] by de Toro, Ortega

Algorithm 2.45: $X^\star = pesa2(c_F)$

Input: c_F the comparator function which allows us to compare the fitness of two solution candidates, used by *updateOptimalSet*
Input: Implicit: p the population size
Data: X_{pop} the population
Data: X_{mp} the mating pool
Output: $X^\star \subseteq \tilde{X}$ the set of the best elements found

```
1  begin
2  |  X* ⟵ ∅
3  |  X_pop ⟵ createPop(p)
4  |  while ¬terminationCriterion() do
5  |  |  foreach x ∈ X_pop do  X* ⟵ updateOptimalSet(X*, x)
6  |  |  X* ⟵ pruneOptimalSet_aga(X*)
7  |  |  X_mp ⟵ pesa2Select(X*, p)
8  |  |  X_pop ⟵ reproducePop(X_mp, p)
9  |  return X*
10 end
```

et al. It is very similar to the basic elitist evolutionary Algorithm 2.2 on page 56.

Algorithm 2.47 on page 112 specifies the behavior of the SFG algorithm using a grid-based pruning technique as stated in the original paper. While SFGA runs locally, a distributed form, called PSFGA is also introduced in the paper. There, the populations of the GA are divided and each runs on a processor for its own. Then global loop restarts after some time when the populations are joined and divided again.

2.6.13 SPEA

The Strength Pareto Evolutionary Algorithm (SPEA [349]) by Zitzler and Thiele for multi-objective optimization, extended for prevalence, works as illustrated in Figure 2.6 on page 112 and is characterized as follows:

- Besides the population a set of the optimal (non-prevailed) individuals generated so far is maintained.
- This set is used to evaluate the fitness of individuals according to the prevalence relationship.
- The population's diversity is preserved using a prevalence based mechanism.
- A clustering method is incorporated in order to reduce the Pareto set without losing its characteristics. ()

The developers of SPEA used the experiments of SPEA uses the average linkage method (see Section 36.3.4 on page 579) in order to reduce the optimal set and a binary tournament for selection (see Section 2.4.3 on

Algorithm 2.46: $X^\star = rpsgae(c_F)$

Input: c_F the comparator function which allows us to compare the fitness of two solution candidates, used by *updateOptimalSet*
Input: Implicit: p the population size
Input: Implicit: n the count of ranks
Data: X_{pop}, X_{pl} the population and its list representation
Data: X_{arc}, X_{al} the archive and its list representation
Data: rc the archive truncation size
Data: i a counter variable
Output: $X^\star \subseteq \tilde{X}$ the set of the best elements found

```
1  begin
2    X_arc ⟵ ()
3    X_pop ⟵ createPop(p)
4    rc ⟵ ⌊p/n + 0.5⌋
5    while ¬terminationCriterion() do
6      rpsgaeFitnessAssign(X_pop, ())
7      X_pl ⟵ sort_a(X_pop, s(x1, x2) ≡ f(x1) − f(x2))
8      i ⟵ ⌊2p/n + 0.5⌋ − 1
9      while i ≥ 0 do
10       X_arc ⟵ appendListItem(X_arc, X_pl[i])
11       i ⟵ i − 1
12     if |X_arc| ≥ 2p then
13       rpsgaeFitnessAssign(X_arc, ())
14       X_al ⟵ sort_a(X_arc, s(x1, x2) ≡ f(x1) − f(x2))
15       X_al ⟵ deleteListRange(X_al, rc, |X_al| − rc)
16       X_pl ⟵ deleteListRange(X_pl, |X_pl| − rc, rc)
17       X_pop ⟵ appendList(X_pl, X_al)
18       X_arc ⟵ X_al
19     X_pop ⟵ reproducePop(X_pop, p)
20   return extractOptimalSet(appendList(X_pop, X_arc))
21 end
```

page 81). For fitness assignment, a customized algorithm introduced in section sec:speaFitnessAssignment is used. The Strength Pareto Evoluationary Algorithm thus works as illustrated in Algorithm 2.48 on page 113.

Except for SPEA2, which is discussed in the next section, there are also other suggestions for improvements of SPEA, for example the usage of unbounded archives [118, 381].

2.6.14 SPEA2

The Strength Pareto Evolutionary Algorithm 2 [350, 351] is an improvement of the original SPEA algorithm. The new algorithm (depicted in Figure 2.7 on page 114) comes with three major modifications:

Algorithm 2.47: $X^\star = sfga(c_F)$

Input: c_F the comparator function which allows us to compare the fitness of two solution candidates, used by *updateOptimalSet*
Input: Implicit: p the population size
Data: X_{pop} the population
Data: X_{mp} the mating pool
Output: $X^\star \subseteq \tilde{X}$ the set of the best elements found

```
1 begin
2     X_pop ⟵ createPop(p)
3     while ¬terminationCriterion() do
4         X_pop ⟵ extractOptimalSet(X_pop)
5         if |X_pop| ≥ p then X_pop ⟵ pruneOptimalSet_aga(X_pop)
6         X_mp ⟵ select(X_pop, p)
7         X_pop ⟵ X_pop ∪ reproducePop(X_mp, p − |X_pop|)
8     return extractOptimalSet(X_pop)
9 end
```

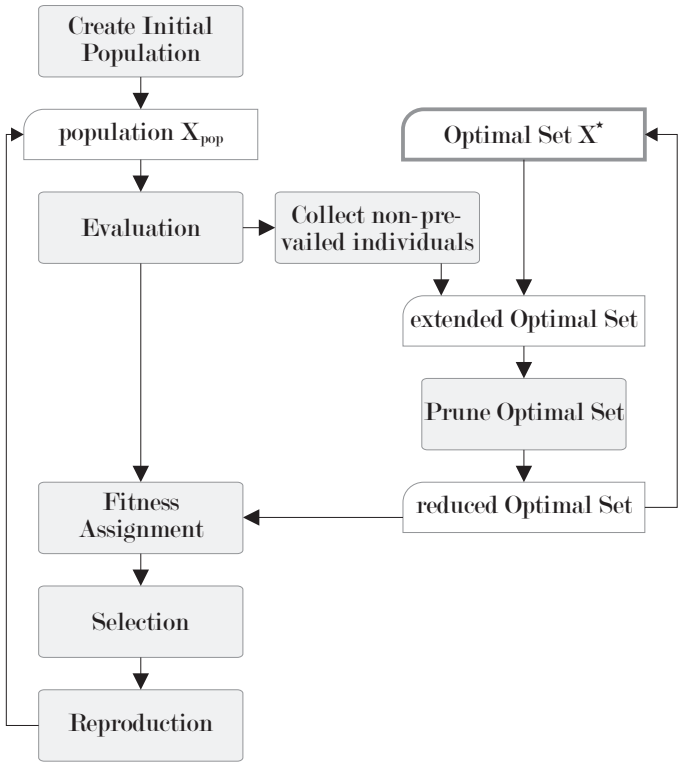


Fig. 2.6: The cycle of the SPEA

Algorithm 2.48: $X^\star = spea(c_F)$

Input: c_F the comparator function which allows us to compare the fitness of two solution candidates, used by *updateOptimalSet*
Input: Implicit: p the population size
Data: X_{pop} the population
Data: X_{sel} the selection pool
Data: X_{mp} the mating pool
Output: $X^\star \subseteq \tilde{X}$ the set of the best elements found

```
1  begin
2      X* ⟵ ∅
3      X_pop ⟵ createPop(p)
4      while ¬terminationCriterion() do
5          foreach x ∈ X_pop do  X* ⟵ updateOptimalSet(X*, x)
6          X* ⟵ pruneOptimalSet(X*)
7          assignFitness(X_pop, X*)
8          X_sel ⟵ appendList(X_pop, X*)
9          X_mp ⟵ tournamentSelect^f_{r,2}(X_sel, p)
10         X_pop ⟵ reproducePop(X_mp, p)
11     return X*
12 end
```

1. A new fitness assignment process (see Section 2.3.11 on page 76) now takes the whole population into consideration when computing the fitness while in SPEA only the optimal set is used for fitness assignment.
2. This fitness assignment process employs density estimation in order to avoid too many individuals with the same fitness. This feature is particularly useful if most of the solution candidates do not prevail each other, especially in situations where the optimization process is just starting up.
3. SPEA2 uses an archive X_{arc} of the fixed-size n as source for selection.

The archive construction algorithm (Algorithm 2.49 on page 115) creates an archive X_{arc} of the fixed size n using an old archive X_{old} and the current population X_{pop}. First, the optimal individuals are copied into this archive. If the archive size is now exactly n, everything is fine. If it contains less than n individuals, it is filled up using the best individuals of the rest of the population and the previous archive. If it already contains too many ($> n$) individuals, it is truncated to the proper size. When truncating, we use the kth nearest neighbor method (see Section 35.8.2 on page 567) successively. First, the set of individuals which are nearest to their 1st nearest neighbor r is chosen. This will contain at least two individuals. Now we look for the individuals which are closest to their 2nd nearest neighbor amongst in r. We iterate this way until we've reached a state where r contains only one individual. This individual is the removed from X_{arc}. If we do not reach this state, we simple remove the first best individual in r from X_{arc}.

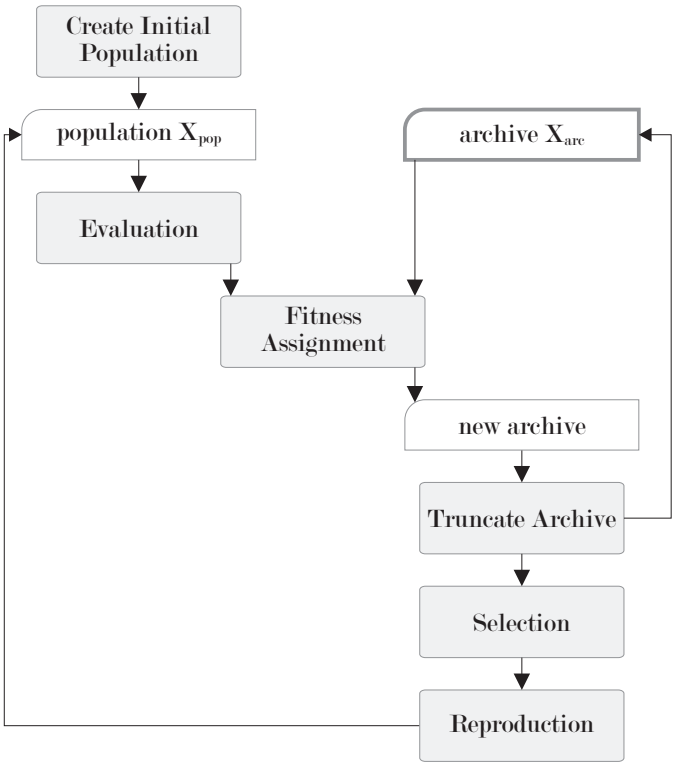


Fig. 2.7: The cycle of the SPEA2

Algorithm 2.48 on the preceding page illustrates the mean loop of SPEA2. First, a scalar fitness value is computed for all individuals in the population and the archive (using the *spea2FitnessAssign*-fitness assignment process introduced as Algorithm 2.12 on page 78 on page 78 per default). Then, the new archive is constructed using the before mentioned Algorithm 2.49 on the facing page. From that archive, the population of the next iteration is produced, using a binary tournament selection scheme with replacement as default.

2.6.15 NCGA

The Neighborhood Cultivation Genetic Algorithm for Multi-Objective Optimization Problems (NCGA) [369, 307] makes use of the archive maintenance feature of SPEA2 but extends it by neighborhood cultivation. Neighborhood cultivation means that crossover only occurs between parents that are similar to each other. NCGA achieves this using a special reproduction scheme (Algorithm 2.35 on page 102) that simply sorts the mating pool according to

Algorithm 2.49: $X_{arc} = constructArchiveSPEA2(X_{old}, X_{pop}, n)$

Input: X_{old} the archive of the previous iteration
Input: X_{pop} the current population
Input: n the wanted archive size
Data: x_{all} the combination of the old archive and the current population
Data: x, z, x_s some individuals used internally
Data: i, k counter variables
Data: r the set used for individual removal
Output: X_{arc} the newly constructed archive

```
1  begin
2      X_all ⟵ appendList(X_pop, X_old)
3      X_arc ⟵ extractOptimalSet(X_all)
4      i ⟵
5      while |X_arc| < n do
6          x_s ⟵ x : f(x) = min{f(z) : z ∈ X_all}
7          X_arc ⟵ appendListItem(X_arc, x_s)
8          X_all ⟵ removeListItem(X_all, x_s)
9          i ⟵ i − 1
10     while |X_arc| > n do
11         k ⟵ 1
12         r ⟵ X_arc
13         while k < |r| do
14             r ⟵ {x : ρ_nnk(x) = min{ρ_nn,k(z) : z ∈ r}}
15             if |r| = 1 then
16                 X_arc ⟵ removeListItem(X_arc, r[0])
17                 k ⟵ |X_arc|
18             k ⟵ k + 1
19         if k = |X_arc| then X_arc ⟵ removeListItem(X_arc, r[0])
20     return X_arc
21 end
```

one *focused* objective first and then recombines individuals which are direct neighbors. The focused objective changes every generation in a cyclic manner. NCGA is specified as Algorithm 2.51 on the following page.

Algorithm 2.50: $X^\star = spea2(c_F)$

Input: c_F the comparator function which allows us to compare the fitness of two solution candidates, used by *updateOptimalSet*
Input: Implicit: n the wanted archive size
Input: Implicit: p the population size
Data: X_{arc} the archive
Data: X_{pop} the population
Data: X_{mp} the mating pool
Output: $X^\star \subseteq \tilde{X}$ the set of the best elements found

```
1  begin
2      X_arc ⟵ ()
3      X_pop ⟵ createPop(p)
4      while ¬terminationCriterion() do
5          assignFitness(X_pop, X_arc)
6          X_arc ⟵ constructArchiveSPEA2(X_arc, X_pop, n)
7          X_mp ⟵ tournamentSelect^f_{r,2}(X_arc, p)
8          X_pop ⟵ reproducePop(X_mp, p)
9      return extractOptimalSet(X_arc)
10 end
```

Algorithm 2.51: $X^\star = ncga(c_F)$

Input: c_F the comparator function which allows us to compare the fitness of two solution candidates, used by *updateOptimalSet*
Input: Implicit: n the wanted archive size
Input: Implicit: p the population size
Input: Implicit: F the objectives functions used by c_F
Data: X_{arc} the archive
Data: X_{pop} the population
Data: X_{mp} the mating pool
Output: $X^\star \subseteq \tilde{X}$ the set of the best elements found

```
1  begin
2      foc ⟵ 0 X_arc ⟵ ()
3      X_pop ⟵ createPop(p)
4      while ¬terminationCriterion() do
5          assignFitness(X_pop, X_arc)
6          X_arc ⟵ constructArchiveSPEA2(X_arc, X_pop, n)
7          X_mp ⟵ tournamentSelect^f_{r,2}(X_arc, p)
8          X_pop ⟵ ncgaReproducePop_foc(X_mp, p)
9          foc ⟵ (foc + 1) mod |F|
10     return extractOptimalSet(X_arc)
11 end
```

3

Genetic Algorithms

3.1 Introduction

Beginning in the 1950s, biologists like Barricelli began to apply computer-aided simulations to study the artificial selection of organisms [382, 383, 384, 385]. Scientists like Bremermann [386] and Bledsoe [387, 388, 389, 390] began to use evolutionary approaches based on binary genomes to solve problems like function optimization and balance weights for neural networks [391] in the early 1960s. At the end of that decade, important research on these genomes was contributed by Bagley [392], Cavicchio [393], and Frantz [394] and conducted by John Holland at the University of Michigan. As a result of his work, Genetic Algorithms[1] (GA) as a new approach for problem solving finally became widely recognized and popular [395, 211, 396, 69]. Today there are many applications of genetic algorithms in science, economy, and research and development [397].

Genetic Algorithms are a subclass of evolutionary algorithms that employs two different representations for each solution candidate. The genotype is used in the reproduction operations whereas the phenotype is the form of the individual which can be used for the determining its fitness [1, 360]. The genotypes in Genetic Algorithms are usually binary strings of fixed or variable length.

Definition 47 (Genotype). A genotype[2] $g \in \mathbb{G}$ is the individual representation on which the reproduction operators work, both in biology as well in evolutionary algorithms.

If only a single objective functions is applied in a Genetic Algorithm, it is often called fitness function, which should not be mixed up with the fitness assignment processes mentioned in Section 2.3 on page 65. The objective

[1] `http://en.wikipedia.org/wiki/Genetic_algorithm` [accessed 2007-07-03]
[2] `http://en.wikipedia.org/wiki/Genotype` [accessed 2007-07-03]

functions do not work directly on the genotypes g but on the phenotypes $x = hatch(g) \in \tilde{X}$ (see Figure 3.1 on the facing page and Definition 57).

Definition 48 (Phenotype). The fitness of an individual is determined on basis of its phenotype[3] $x \in \tilde{X}$, both in biology as well in evolutionary algorithms.

Phenotypes are the solution models, which can for example be evaluated in simulations. There are two reasons for the existence of genotypes during the reproduction process

1. The genotypic form is chosen in a way that can especially easily be handled be reproduction operations.
2. There exists a set of highly efficient, well-studied, and simple operators for data types like bit strings. If we can reuse them, we lower the risk of making errors because we do not need to design them on our own. Additionally, we can rely on the mathematical features of these operators which have been validated in years of research.

If we, for instance, are able to encode the design of an electronic circuit to a variable-length binary string [398, 399, 400], we can use the default reproduction operators for such strings. Creating new operators that process the circuit designs directly on the other hand would be error prune, time-consuming, and no necessarily yield better results.

The best phenotypes found in the domain $\tilde{X}$ are the outputs of the genetic algorithms whereas the genotypes are just an internal formats used in the reproduction operations (see Section 2.5 on page 99).

The process of transforming the genotypic representations to their corresponding phenotype is called genotype-phenotype mapping and is discussed thoroughly in Section 3.5 on page 127 and subject to active research.

There exist various forms of genetic algorithms [401]. Some even allow/require human interaction or evaluation of the individuals, like interactive genetic algorithms[4] [402, 403] or human-based genetic algorithms[5] (HBGA) [404, 405].

From the algorithmic point of view, there is not much of a difference between GA and EA – the only major issue is the explicit option in GA to employ different representations for genotypes and phenotypes. One could for example consider evolutionary algorithms as genetic algorithms with equal genotypes and phenotypes[6].

[3] `http://en.wikipedia.org/wiki/Phenotype` [accessed 2007-07-03]

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

[5] `http://en.wikipedia.org/wiki/HBGA` [accessed 2007-07-03]

[6] see Definition 49 on page 122

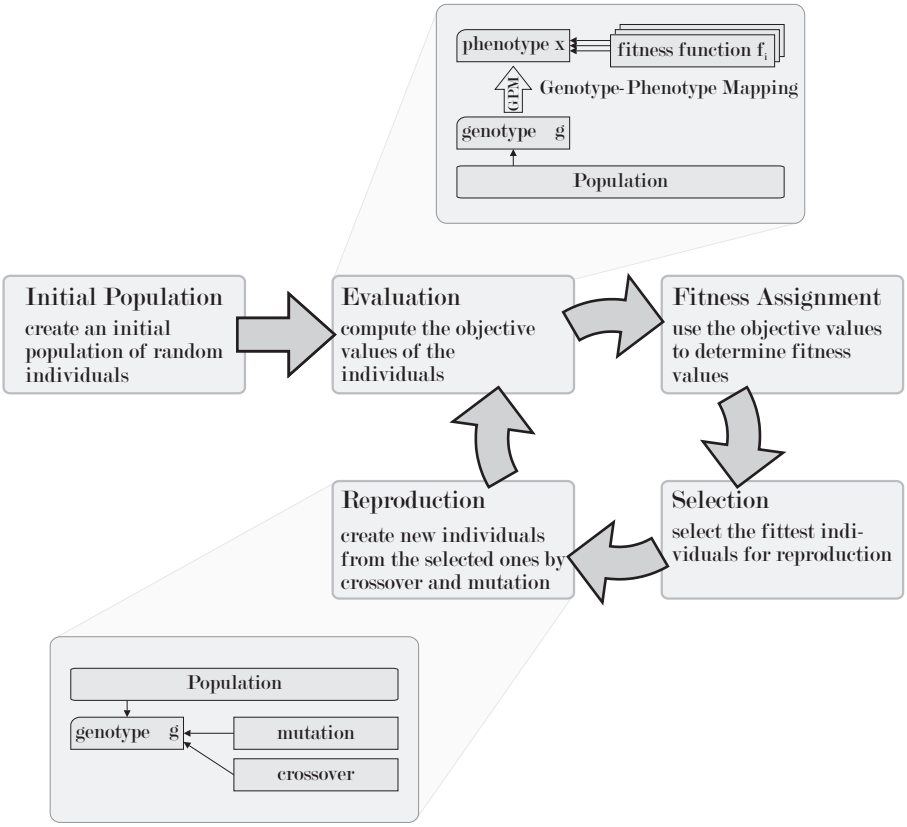


Fig. 3.1: The basic cycle of genetic algorithms.

3.2 General Information

3.2.1 Areas Of Application

Some example areas of application of genetic algorithms are:

Application	References
scheduling problems	[406, 407, 408, 409]
chemistry and chemical manufacturing	[348, 410, 411, 412, 413]
medicine	[414, 415, 416, 417]
data mining and data analysis	[418, 419, 420, 421, 231]

geometry	[422, 423, 424, 425, 426, 427]
finance and trade	[428]
optimizing distributed protocols	[429, 430]

For more information see also [397].

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 61

FOGA: Foundations of Genetic Algorithms
`http://www.sigevo.org/` [accessed 2007-09-01]
History: 2007: Mexico City, Mexico, see [431]
2005: Aizu-Wakamatsu City, Japan, see [432]
2002: Torremolinos, Spain, see [433]
2000: Charlottesville, VA, USA, see [434]
1998: Madison, WI, USA, see [435]
1996: San Diego, CA, USA, see [436]
1994: Estes Park, Colorado, USA, see [437]
1992: Vail, Colorado, USA, see [438]
1990: Bloomington Campus, Indiana, USA, see [439]

GALESIA: International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications
now part of *CEC*, see Section 2.2.2 on page 60
History: 1997: Glasgow, UK, see [440]
1995: Scheffield, UK, see [441]

GECCO: Genetic and Evolutionary Computation Conference
see Section 2.2.2 on page 62

ICGA: International Conference on Genetic Algorithms
Now part of GECCO, see Section 2.2.2 on page 62
History: 1997: East Lansing, Michigan, USA, see [442]
1995: Pittsburgh, PA, USA, see [443]
1993: Urbana-Champaign, IL, USA, see [444]
1991: San Diego, CA, USA, see [445]
1989: Fairfax, Virginia, USA, see [371]
1987: Cambridge, MA, USA, see [446]

1985: Pittsburgh, PA, USA, see [447]

ICANNGA: International Conference on Adaptive and Natural Computing Algorithms
see Section 2.2.2 on page 63

Mendel: International Conference on Soft Computing
see Section 1.8.2 on page 42

3.2.3 Books

Some books about (or including significant information about) genetic algorithms are (ordered alphabetically):

Goldberg: *Genetic Algorithms in Search, Optimization and Machine Learning* (see [69])

Mitchell: *An Introduction to Genetic Algorithms* (see [17])

Holland: *Adaptation in Natural and Artificial Systems* (see [211])

Gen and Chen: *Genetic Algorithms (Engineering Design and Automation)* (see [448])

Quagliarella, Periaux, Poloni, and Winter: *Genetic Algorithms and Evolution Strategy in Engineering and Computer Science: Recent Advances and Industrial Applications* (see [397])

Gwiazda: *Crossover for single-objective numerical optimization problems* (see [449])

3.3 Genomes

The term *Genome*[7] was coined in 1920 by the German biologist Hans Winkler [450] as a portmanteau of the words gene and chromosome [451]. It identifies the whole hereditary information of an organism. This includes both, the genes and the non-coding sequences of the Deoxyribonucleic acid (DNA[8]), which is illustrated in Figure 3.2.

Simply put, the Dna is a string of base pairs that encodes the phenotypical characteristics of the creature it belongs to. When reproducing sexually, the genes of the two parents genotypes will recombine. Additionally, small variations (mutations) will modify the chromosomes during this process. In asexual reproduction, mutations are the only changes that occur. After the genes have been copied this way, life begins with a single cell which divides[9] time and again until a mature individual is formed[10] The emergence of an

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

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

[9] `http://en.wikipedia.org/wiki/Cell_division` [accessed 2007-07-03]

[10] Matter of fact, cell division will continue until the individual dies, but I think you got my point here.

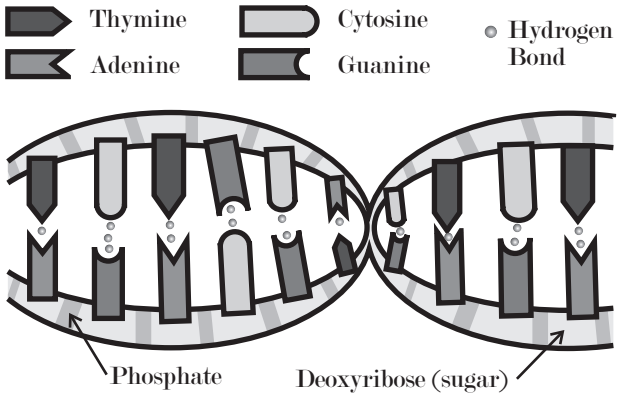


Fig. 3.2: A sketch of a part of a DNA molecule.

individual phenotype from its genotypic representation is called embryogenesis, its artificial counterpart, as for example used in genetic algorithms, is discussed in Section 3.5.1 on page 128.

Definition 49 (Genome). In genetic algorithms, the genome (or chromosome[11]) $\mathbb{G}$ is the space of possible individual representations $g \in \mathbb{G}$ that can be processed by the reproduction operations. It encompasses the set of parameters that define a possible solution.

Definition 50 (Solution Space). While the genome represents the *search space* $\mathbb{G}$ where the genetic operations take place, the *solution space* (or *problem space*) is the space $\tilde{X}$ of all solution candidates. In genetic algorithms, both differ and are connected with the aid of a genotype-phenotype mapping[12]. In other optimization algorithms like for example genetic programming, they may as well be equal $\mathbb{G} = \tilde{X}$.

The genome $\mathbb{G}$ of in genetic algorithms is normally different from the space of possible solutions (phenotypes[13]) $\tilde{X}$.

Definition 51 (Gene). A gene is the basic informational unit in a genome. This 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[14].

[11] `http://en.wikipedia.org/wiki/Chromosome_%28genetic_algorithm%29` [accessed 2007-07-03]

[12] The subject of mapping genotypes to phenotypes is elaborated on in Section 3.5 on page 127.

[13] see Definition 47 and Definition 48 on page 118

[14] `http://en.wikipedia.org/wiki/Gene` [accessed 2007-07-03]

Definition 52 (Allele). An allele[15] is a value of specific gene. A gene which is a bit for example can have the values $\{0, 1\}$, a gene representing a real number can take on real values in $\mathbb{R}$, and so on. Thus, as in nature, an allele is a specific instance of a gene.

Definition 53 (Locus). The locus is the position where a specific gene can be found in a chromosome[16].

Definition 54 (Intron). In biology, an intron[17] is a section of the DNA that has no (obvious) function [452]. Corresponding to this natural phenomenon, we refere to the parts of a genotype $g \in \mathbb{G}$ that do not contribute to the phenotype $x = hatch(g) \in \tilde{X}$ also 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 are now not used anymore. It has however recently been discovered that introns are not as useless as initially assumed. Instead, they seem to provide support for efficient splicing.

Figure 3.3 illustrates the relations between the aforementioned entities in a bit string genome $\mathbb{G}$ of the length 5. Additionally, it shows that a gene in the genotype g corresponds to one feature of the phenotype x in the randomly chosen example solution space $\tilde{X}$. If additional bits were appended to the genotype – for example because our computer always uses full bytes instead of bits – these occur as introns and will not influence the phenotype.

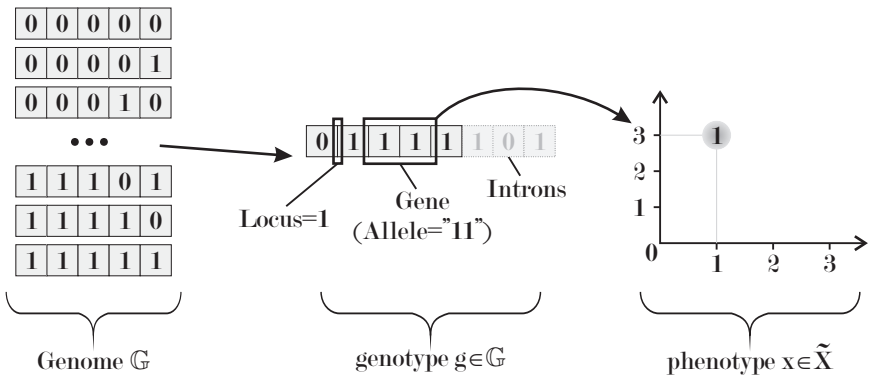


Fig. 3.3: A five bit string genome $\mathbb{G}$ and a fictitious phenotype $\tilde{X}$.

[15] http://en.wikipedia.org/wiki/Allele [accessed 2007-07-03]
[16] http://en.wikipedia.org/wiki/Locus_%28genetics%29 [accessed 2007-07-03]
[17] http://en.wikipedia.org/wiki/Intron [accessed 2007-07-05]

3.4 String Chromosomes

In genetic algorithms, we most often use chromosomes that are strings of one and the same data type, for example bits or real numbers [449, 448, 17].

Definition 55 (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 positions of the genes are constant and hence, the tuples may contain elements of different types.

$$\mathbb{G} = \{\forall(g_1, g_2, \ldots, g_n) : g_1 \in \mathbb{G}_1, g_2 \in \mathbb{G}_2, \ldots, g_n \in \mathbb{G}_n\} \tag{3.1}$$

This is not given in variable-length string genomes. Here, the positions of the genes may shift due to the reproduction operations. Thus, all elements of such genotypes must have the same type $\mathbb{G}_l$.

$$\mathbb{G} = \{\forall \mathit{lists}\ g : g[i] \in \mathbb{G}_l\ \forall 0 \leq i < |g|\} \tag{3.2}$$

String chromosomes are often bit strings or vectors of real numbers. Genetic algorithms with such real vector genomes in their *natural representation* are called *real encoded* [453]. As already said, string genomes of fixed length may also contain mixed elements, for example like $\mathbb{G} = \{\forall(g_1 \in \mathbb{R}, g_2 \in [0,1], g_2 \in \{a,b,c,d\})\}$.

In the case of bit string genomes, it is common practice to use gray coding[18] when transforming them into their phenotypic representations. This ensures that small changes in the genotype will also lead to small changes in the phenotype. This method is discussed in [454].

3.4.1 Fixed-Length String Chromosomes

Especially widespread are fixed-length genomes. A lot of research has been conducted investigating the properties of their crossover and mutation operations [455].

Creation

Creation of fixed-length string individuals means simple to create a new tuple of the structure defined by the genome and initialize it randomized.

Mutation

Mutation is an important method of preserving individual diversity. In fixed-length string chromosomes it can be achieved by modifying the value of one

[18] `http://en.wikipedia.org/wiki/Gray_coding` [accessed 2007-07-03]

element of the genotype, as illustrated in Figure 3.4. More generally, a mutation may change $0 \leq n < |g|$ locations in the string. In binary coded chromosomes for example, the elements are bits which are simply toggled. For real-coded 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 $g_i^{new} \sim N(g_1, \sigma)$.

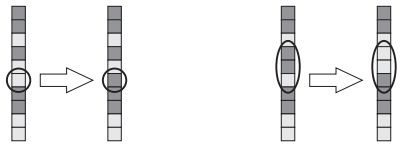


Fig. 3.4: Value-altering mutation of string chromosomes.

Permutation

The permutation operation is an alternative mutation method where the alleles of two genes are exchanged. 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 traveling salesman problem. Here, a genotype could encode the sequence in which the cities are visited. Exchanging two alleles then equals of switching two cities in the route.

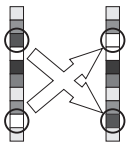


Fig. 3.5: Permutation applied to a string chromosome.

Crossover

Figure 3.6 outlines the recombination of two string chromosomes. This operation is called crossover and is done by swapping parts of the genotypes between the parents.

When performing crossover, both parental chromosomes are split at a randomly determined crossover point. Subsequently, a new child chromosome is created by appending the first part of the first parent with the second part of the second parent. This method is called one-point crossover. In two-point crossover, both parental genotypes are split at two points, constructing a new offspring by using parts number one and three from the first, and the middle part from the second ancestor. The generalized form of this technique is n-point crossover. For fixed-length strings, the crossover points for both parents are always identical.

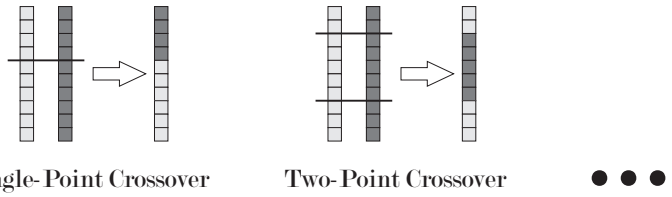


Fig. 3.6: Crossover (recombination) of fixed-length string chromosomes.

3.4.2 Variable-Length String Chromosomes

Creation

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.

Mutation

If the string chromosomes are of variable length, the set of mutation operations introduced in Section 3.4.1 can be extended by two additional methods. On one hand, one could insert a couple of elements at any given position into a chromosome. One the other hand, this operation can be reversed by deleting elements from the string. These operations are one of the reasons why variable-length strings need to be constructed of elements the same set because the positions of the elements are no longer be coupled to their types anymore - one would need to including additional logic into the operators and the evaluation functions otherwise. It should be noted that both, insertion and deletion, are also implicitly 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.

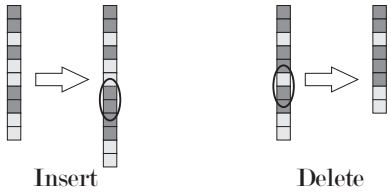


Fig. 3.7: Mutation of variable-length string chromosomes.

Crossover

For variable-length string chromosomes, the same crossover operations are available as for fixed-length strings except that the strings now are not necessarily split at the same positions. The length of the new strings resulting from such a *cut and splice* operation may now differ from the length of the parents (which itself may also differ, see Figure 3.8).

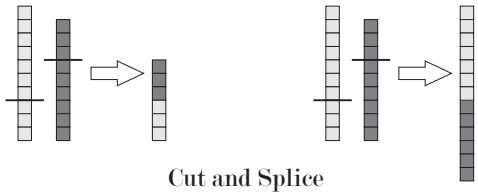


Fig. 3.8: Crossover of variable-length string chromosomes.

3.5 Genotype-Phenotype Mapping

Genetic algorithms often use string genomes to encode the phenotypes of the individuals that represent the possible candidate solutions. These phenotypes however do not necessarily need to be one-dimensional strings too. Instead, they can be construction plans, or trees[19].

Definition 56 (Genotype-Phenotype Mapping).
The process of translating genotype into a corresponding phenotype is called genotype-phenotype mapping (GPM in short) [456].

GPM is often also revered to as ontogenic mapping [188] which has its reason in the natural analog discussed in Section 3.5.1. In the context of

[19] See for example Section 4.5.5 on page 165

this book, we define the two operations *hatch* and *regress* considering this translation.

Definition 57 (*hatch*). The operation *hatch* transforms one instance of the genotype $g \in \mathbb{G}$ into an instance of the phenotype $x \in \tilde{X}$. It is possible that different genomes $g_1, g_2 \in \mathbb{G}, g_1 \neq g_2$ may result in the same phenotype $x \in \tilde{X}, x = hatch(g_1) = hatch(g_2)$ since the function *hatch* is not bijective.

$$hatch(g) = x \ : \ g \in \mathbb{G}, x \in \tilde{X} \tag{3.3}$$

Computing the corresponding phenotype of a given genotype may involve arbitrary complicated calculations (see especially artificial embryogeny in Section 3.5.1). There are also mappings that require further restrictions or corrections [457] in order to produce *valid* phenotypes for all possible genotypes $g \in \mathbb{G}$.

Definition 58 (*regress*). The operation *regress* transforms one instance of the phenotype $x \in \tilde{X}$ into one instance of the genotype $g' \in \mathbb{G}$. It is possible that different phenotypes $x_1, x_2 \in \tilde{X}, x_1 \neq x_2$ may result in the same genome $g' \in \mathbb{G}, g' = regress(x_1) = regress(x_2)$ since the function *hatch* is not bijective.

$$regress(x) = g' \ : \ x = hatch(g'), g' \in \mathbb{G}, x \in \tilde{X} \tag{3.4}$$

regress is the reverse operation of *hatch*. Notice that for some forms of GPM, it may not be possible to specify this inverse operation. In other cases, it may only be defined for a subset $\tilde{X}' \subseteq \tilde{X}$ of $\tilde{X}$.

3.5.1 Artificial Embryogeny

Embryogenesis is the process in nature in which the embryo forms and develops[20]. The genotype-phenotype mapping in genetic algorithms and genetic programming corresponds to this natural process. Most of even the more sophisticated mappings have however an implicit one-to-one mapping in terms of complexity. In the grammar-guided genetic programming approach Gads[21], 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 often more complex. Among other things, the Dna for instance encodes the structural design information of the human brain. For over 100 trillion neural connections in our cerebrum, there are only about 30 thousand active genes in the human genome (2800 million amino acids) [458]. A huge manifold of information is hence decoded from "data" which is of a much lower magnitude. This process 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.

[20] `http://en.wikipedia.org/wiki/Embryogenesis` [accessed 2007-07-03]
[21] See Section 4.5.4 on page 162 for more details.

Definition 59 (Artificial Embryogeny). We subsume all methods of transforming a genotype into a phenotype of (much) higher complexity under the subject of *artificial embryogeny* [458, 459, 460] (also known as computational embryogeny [461, 462]).

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 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.

An example for gene reuse is the non-trivial mapping is the genotype-phenotype mapping performed in grammatical evolution which is discussed in Section 4.5.5 on page 166. Although the resulting phenotypes are not much more complicated than the genotypes, it can be viewed a bridge method between normal GPM and artificial embryogeny.

3.6 Schema Theorem

The schema theorem is a special case of forma analysis (discussed in Section 2.1.4 on page 56) in genetic algorithms. Matter of fact, it is older than its generalization and was first stated by Holland in 1975 [211, 209, 54].

3.6.1 Schemata and Masks

Most generally said, in genetic algorithms the genotypes g of individuals in the search space $\mathbb{G}$ are often represented as strings of a fixed-length l over an alphabet[22] Σ, i. e. $g \in \mathbb{G} = \Sigma^l$. Most often, Σ is the binary alphabet $\Sigma = \{0, 1\}$.

From forma analysis we know that properties can be defined on the genotypic or the phenotypic space. When we have 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 60 (Mask). For a fixed-length string genome $\mathbb{G} = \Sigma^l$ we define the set of all genotypic masks Φ_l as the power set[23] of the valid loci $\Phi_l = \mathcal{P}(\{0, \ldots, l-1\})$ [212]. Every mask $\phi_i \in \Phi_l$ defines a property p_i and an equivalence relation:

$$g \sim_{\phi_i} h \Leftrightarrow g_j = h_j \; \forall \; j \in \phi_i \tag{3.5}$$

[22] Alphabets and such and such are defined in Section 37.3 on page 614.
[23] The power set you can find described in Definition 94 on page 504.

The order $o(\phi_i)$ of the mask ϕ_i is the number of loci defined by it:

$$o(\phi_i) = |\phi_i| \tag{3.6}$$

The defined length $\delta(\phi_i)$ of a mask ϕ_i is the maximum distance between two indices in the mask:

$$\delta(\phi_i) = \max\{|j-k| \; \forall \; j,k \in \phi_i\} \tag{3.7}$$

A mask contains the indices of all characters in a string that are interesting in terms of the property it defines. Assume we bit strings of the length $l = 3$ as genotypes ($\mathbb{G} = \{0,1\}^3$). The set of valid masks Φ is then $\Phi = \{\{0\},\{1\},\{2\},\{0,1\},\{0,2\},\{1,2\},\{0,1,2\}\}$. The mask $\phi_1 = \{1,2\}$ for specifies that the values at the loci 0 and 1 of a genotype denote the value of a property p_1 and the value of the bit at position 2 is irrelevant. Therefore, it defines four formae $A_{p_1=(0,0)} = \{(0,0,0),(0,0,1)\}$, $A_{p_1=(0,1)} = \{(0,1,0),(0,1,1)\}$, $A_{p_1=(1,0)} = \{(1,0,0),(1,0,1)\}$, and $A_{p_1=(1,1)} = \{(1,1,0),(1,1,1)\}$.

Definition 61 (Schema). A forma defined on a string genome concerning the values of the characters at specified loci is called *Schema* [211, 463].

3.6.2 Wildcards

The second method of specifying such schemata is to use do not care symbols (wildcards) to create "blueprints" H of their member individuals. Therefore, we place the do not care character $*$ at all irrelevant positions and the characterizing values of the property at the others.

$$H_i \in \Sigma \cup \{*\} \tag{3.8}$$

$$H_i = \begin{cases} g_j & \textit{if } j \in \phi_i \\ * & \textit{otherwise} \end{cases} \tag{3.9}$$

We now can redefine the aforementioned schemata like: $A_{p_1=(0,0)} \equiv H_1 = (0,0,*)$, $A_{p_1=(0,1)} \equiv H_2 = (0,1,*)$, $A_{p_1=(1,0)} \equiv H_3 = (1,0,*)$, and $A_{p_1=(1,1)} \equiv H_4 = (1,1,*)$.

These schemata mark hyperplanes in the genome space, as illustrated in Figure 3.9 for the three bit genome.

3.6.3 Holland's Schema Theorem

The original schema theorem[24] was defined by Holland [211, 209, 54] for genetic algorithms where fitness is to be maximized and that use fitness-proportionate selection (see Section 2.4.5 on page 84).

$$m(H,t+1) \geq \frac{m(H,t)\overline{f}(H,t)}{\overline{f}(t)}(1-p) \tag{3.10}$$

where

[24] http://en.wikipedia.org/wiki/Holland%27s_Schema_Theorem [accessed 2007-07-29]

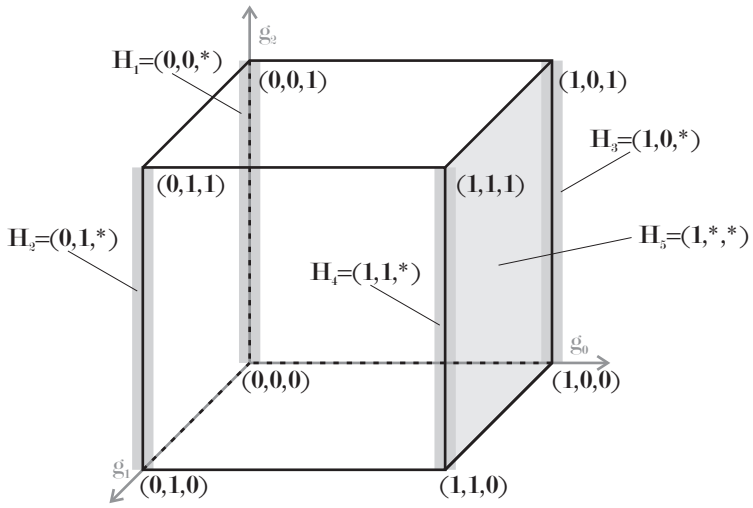


Fig. 3.9: An example for schemata in a three bit genome.

- $m(H,t)$ is the number of instances of a given Schema A defined by the blueprint H in the population of time step t,
- $\overline{f}(H,t)$ is the average fitness of these individuals (observed in time step t),
- $\overline{f}(t)$ is the average fitness of the population in time step t, and
- p is the probability that 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.

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. Generally, the schema theorem represents a lower bound that will only hold for one generation [360]. Trying to derive predictions for more than one or two generations using the schema theorem as is will lead to misleading or wrong results [220, 464].

Furthermore, the population of a genetic algorithm only represents a sample of limited size of the genotypic space $\mathbb{G}$. This limits the reproduction of the

schema but also makes statements about probabilities in general more complicated. We also have only samples of the schemata H and cannot be sure if $\overline{f}(H,t)$ really represents the average fitness of all the members of the schema. Even reproduction operators that preserve the instances of the schema may lead to a decrease of $\overline{f}(H,t)$. This becomes especially critical if 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.

We also do not 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 vs. exploitation topic and the importance of diversity in Section 1.4.1 on page 22.

Another issue is that we 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.

It can also be argued that there are properties for which we cannot specify schema blueprints or masks. If we take the set D_3 of numbers divisible by three, for example $D_3 = \{3, 6, 9, 12, \ldots\}$. Representing them as binary strings will lead to $D_3 = \{0011, 0110, 1001, 1100, \ldots\}$ if we have a bit-string genome of the length 4. Obviously, we cannot seize these individuals in a schema using the discussed approach. They may, however, be gathered in a forma, but the schema theorem cannot hold then since the probability of destruction may be different from forma instance to instance.

3.6.5 The Building Block Hypothesis

The building block hypothesis (BBH) [69, 211] 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. [69]

Although it seems as if the building block hypothesis was 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 [465] (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 trail of thought, it can be regarded as not (yet) proven sufficiently.

3.7 Principles for Individual Representations

In software engineering, there are some design patterns[25] 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 evolutionary algorithms, we need to define a representation for the solution candidates, the structure of the elements in $\mathbb{G}$. The individual representation along with the genotype-phenotype mapping is a vital part of genetic algorithms and has major impact on the chance of finding good solutions. Like in software engineering, there are some principles that lead to better solutions if considered in this process [334, 466].

Two general design patterns for genotypes are [69, 466]

1. The representations for the formae[26] and schemata should be as short as possible and the representations of different, compatible 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.

Some other simple rules have been defined for tree-representations in [467, 468] and generalized in [466]:

3. A good genome should be able to represent all phenotypes, i.e.

$$\forall\, x \in \tilde{X} \Rightarrow \exists\, g \in \mathbb{G} : x = hatch(g) \tag{3.11}$$

4. $\mathbb{G}$ should be unbiased in the sense that all phenotypes are represented by the same number of phenotypes, i.e.

$$\forall\, x, y \in \tilde{X} \Rightarrow |\{g_x \in \mathbb{G} : x = hatch(g_x)\}| \approx |\{g_y \in \mathbb{G} : y = hatch(g_y)\}| \tag{3.12}$$

5. The transformation from genotypes to phenotypes (the genotype-phenotype mapping, the artificial embryogeny) should always yield *valid* phenotypes. The meaning of valid in this context is that if our problem space $\tilde{X}$ is the space of real vectors with three elements, $\tilde{X} \subseteq \mathbb{R}^3$, only such vectors are the result of the . No vectors with fewer or more elements will be produced. 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.
7. The genotypic representation should possess locality (or causality), i. e. small changes in the genotype lead to small changes in the phenotype. Optimally, this would mean that[27]:

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

[26] See Section 2.1.4 on page 56 for more details on formae analysis.

[27] Here, *reproduce* stands for any given reproduction operation

$$\forall\, x, x' \in \tilde{X} : x' = reproduce(x) \Rightarrow x' \approx x \tag{3.13}$$

[469, 466] summarize additional rules:

8. The genotypic representation should be aligned to a set of reproduction operators in a way that good configurations of schemata, the building blocks, are preserved when creating offspring.
9. The representations should minimize epistasis (see Section 3.7.2 on the facing page).
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 (see also rule 6).

3.7.1 Locality and Causality

The 7^{th} rule discussed is very basic and can easily be justified: Generally we can assume that the individuals that are processed by reproduction operations have previously been selected. The chance of being selected is higher, the fitter an individual is. Reversing this statement suggests that individuals which have been selected are likely to have a good fitness. The fitness of a solution candidate depends on its properties, and we can assume that these, in turn, depend on their genotypic representation. If small changes in this representation lead to small changes in the properties, as outlined in Equation 3.13, we can assume that also the objective values (the utility of the solution candidate) changes only slightly. Hence, we have a smooth fitness landscape and the optimization algorithm can perform a gradient descent. If the phenotypes, on the other hand, change wildly in each reproduction cycle, the fitness landscape will become rugged and the optimizer has no hint in which direction to move. This problem has been discussed in Section 1.4.2 on page 25.

Definition 62 (Locality). The principle of strong locality (causality) states that small changes in an object lead to small changes in its behavior [470, 471].

In natural genomes, the same principle can be observed. Small modifications in the genotype of a fish induced by mutation will more probably lead to a change of the color of the scales of its offspring than producing totally different creatures.

Apart from its straightforward, informal explanation here, causality has been investigated thoroughly in different fields of artificial evolution, such as evolution strategy [470, 472], structure evolution based on evolution strategy [473], Genetic Programming [471, 474, 472], genotype-phenotype mapping [475], reproduction operators [472], and evolutionary algorithms in general [476, 334, 472].

3.7.2 Epistasis

Another very important aspect of encoding solution candidates is picked up in rule 9 and already indirectly mentioned in rule 1: The representations for compatible formae should not influence each other. In biology, *epistasis* is defined as a form of interaction between different genes. It was coined by Bateson [477] in order to describe how one gene can suppress the phenotypical expression of another gene. According to Lush [478, 479], the interaction between genes is epistatic if the effect on the fitness from altering one gene depends on the allelic state of other genes.

Definition 63 (Epistasis). Epistasis in evolutionary algorithms means that a change in one property of a solution candidate, introduced by an reproduction operation for instance, also leads to a change in some of its other properties [480, 481].

We speak of minimal epistasis when every gene is independent of every other gene and of maximal epistasis when no proper subset of genes is independent of any other gene [439, 482].

Such behavior violates the locality previously discussed, since changes in the phenotypes and the objective values resulting from changes in the genotypes should be small. In a genome with high epistasis, a modification in a genotype will alter multiple properties of a phenotype. Hence, we should try to avoid epistasis in the design of the genome [480].

In [483], Naudts and Verschoren have shown on the example of length-two binary string genomes that deceptiveness[28] does not occur in situations with low epistasis. However, even fitness functions with high epistasis are note necessarily deceptive.

3.7.3 Redundancy

The degree of redundancy in the context of genetic representations denotes how many genotypes $g \in \mathbb{G}$ represent the same phenotype $x \in \tilde{X}$. Different from epistasis or locality, redundancy may have both, positive and negative effects [466].

Redundancy can have much impact on the explorability of the problem space. If we imagine a one-to-one mapping, the translation of a slightly modified genotype will always result in a different phenotype. If the number n of loci where a genotype can be modified is finite (as it normally is), there are also exactly n different phenotypes that can be reached by altering one genotype. If there exists a many-to-one mapping between genotypes and phenotypes, reproduction operations may create offspring genotypes different from the parent, which still translate to the same phenotype. The evolution may now walk a neutral path leading to new, unexplored regions of the search space.

[28] See Section 1.4.3 on page 25 for a introduction of deceptive fitness landscapes.

If all genotypes along this path can be modified to n different offsprings, and only a fraction of them leads to the same phenotypes, many more new solution candidates can be reached. Positive effects of redundancy have been observed in [484, 485, 486, 487, 81]. In the Cartesian Genetic Programming method, neutrality is explicitly introduced in order to increase the evolvability (see Section 4.7.2 on page 183).

Neutrality however can also have very negative effects, as already outlined in Section 1.4.4 on page 26. [334, 486] show that uniform neutrality is harmful for the evolutionary process [466]. If redundancy exists in a genome, it is vital that only some fraction of the possible results of the reproduction operations produce the same phenotypes than their parental genotype. Otherwise, the negative effects described in Section 1.4.4 will occur, since the optimization algorithm has no hint in which direction it should proceed.

The effects of neutrality induced by redundancy of natural genomes are also part of scientific discussion and are also considered to have positive and negative effects [71].

It is not possible to state a simple rule-of-thumb on how to deal with redundancy in genomes. In principle, one should avoid it where it seems useless (for example, when encoding real numbers) and permit it, where it seems beneficial (for problems with rugged fitness landscapes, for instance).

3.7.4 Implications of the Forma Analysis

In this section, we will discuss some of the implications of forma analysis for representations in genetic algorithms as stated in [213, 212].

Formae in Genotypic and Phenotypic Space

The major idea here is that most arguments considering the Building Block Hypothesis or the Schema theorem focus on the genotypic representations $g \in \mathbb{G}$ of the individuals, the bit-strings. The fitness of the schemas however depends on their expression in the solution space $\tilde{X}$. This representation in turn results from the genotype-phenotype mapping (see Section 3.5 on page 127). The GPM may map single schemas to single properties, but phenotypic properties could also be completely unrelated to any schema. Hence, *intrinsic parallelism*, as discussed in Section 2.1.4 on page 56, exists only in phenotypic space.

From this point of view, it becomes clear that useful separate properties in phenotypic space can only be combined by the reproduction operations properly if they are separate in genotypic space too. In other words, formae in phenotypic space should also be formae in genotypic space. Then, implicit parallelism also applies to genotypic space.

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.

Inheritance of Formae

From a good recombination (crossover) operation we can expect that the offspring of two members x, y of a forma A are also always members of the forma.

$$\forall\, x, y \in A \subseteq \tilde{X} \Rightarrow crossover(x, y) \in A \tag{3.14}$$

If we furthermore can assume that all instances of all formae A with minimal precision $mini$ of an individual (which was created via crossover) are inherited by at least one parent, crossover performs a pure recombination.

$$\forall\, z = crossover(x, y) \in \tilde{X},\ \forall; A \in mini : z \in A \Rightarrow x \in A \vee y \in A \tag{3.15}$$

Otherwise, crossover may also perform an implicit mutation. This property is not necessarily needed, more important is that the recombination operation is able to combine two parents which are instances of different but compatible forma in a way that their offspring is an instance of both formae.

Reachability of Formae

The mutation operator should be able to reach all possible formae. If crossover is purely recombinatorial, mutation is the only genetic operation able to introduce new formae not yet present in the population. Hence, it should be able to find any given forma.

4

Genetic Programming

4.1 Introduction

The term *Genetic Programming*[1] [11, 1] has two possible meanings. On one hand, it is often used to subsume all evolutionary algorithms that produce tree data structures as phenotypes. On the other hand, we can also define it 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 involves discussing tree-shaped genomes.

The conventional well-known input-processing-output model[3] of information processing in computer science states that a running instance of a program uses its input information to compute and return output data. In Genetic Programming, we already know (or are able to produce) some inputs or situations and corresponding output data samples and we want 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.

In 1958, Friedberg left the first footprints in this area by using a learning algorithm to stepwise improve a program [488, 489]. The program was represented as a sequence of instructions[4] for a theoretical computer called *Herman*. Friedberg did not use an evolutionary, population-based approach for searching the programs. This may be because the idea of evolutionary algorithms wasn't fully developed yet[5] and also because of the limited computational capacity of the computers of that era.

Twenty years later, a new generation of scientists began to look for ways to evolve programs. First new results were reported by Smith in his PhD

[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 37.1.1 on page 585.

[3] see Section 37.1.1 on page 587

[4] Linear Genetic Programming is discussed in Section 4.6 on page 177.

[5] Compare with Section 3.1 on page 117.

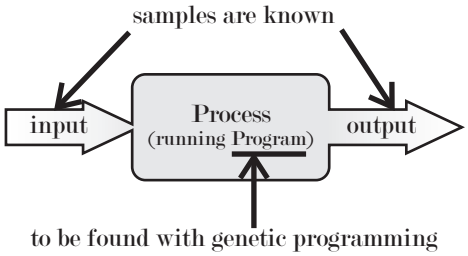


Fig. 4.1: Genetic programming in the context of the IPO model.

thesis [490] in 1980. Forsyth evolved trees denoting fully bracketed Boolean expressions for classification problems in 1981 [491, 492, 493]. Four years later, Cramer applied a Genetic Algorithms in order to evolve a program written in a subset of the programming language `PL`.[6] This GA used a string of integers as genome and employed a genotype-phenotype mapping that recursively transformed them into program trees [494]. At the same time, the undergraduate student Schmidhuber 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 [495, 496].

Genetic Programming became fully accepted at the end of this productive decade mainly because of the work of John R. Koza. One of the most important of the approaches that he developed was symbolic regression[7], a method for obtaining mathematical expressions that match given data samples. Koza further formalized (and patented [497, 498]) 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 encode `Lisp` S-expressions[8] where a node stands for an operation and its child nodes are the parameters of the operation. Leaf nodes are terminal symbols like numbers or variables. This form of Genetic Programming is called *Standard Genetic Programming* or SGP. With it, not only mathematical functions but also more complex programs can be expressed as well.

Generally, a tree can represent a rule set [499, 500], a mathematical expressions, a decision tree (of course) [501], or even the blueprint of an electrical circuit [502]. 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, alternatives, and such and such. Not only does this form normally constitute a tree – compilers even use tree representations internally. When reading the source code of a program,

[6] Cramer's approach is discussed in Section 4.4.1 on page 154.
[7] More information on symbolic regression is presented in Section 19.1 on page 329 in this book.
[8] List S-expressions are discussed in Section 37.3.11 on page 627

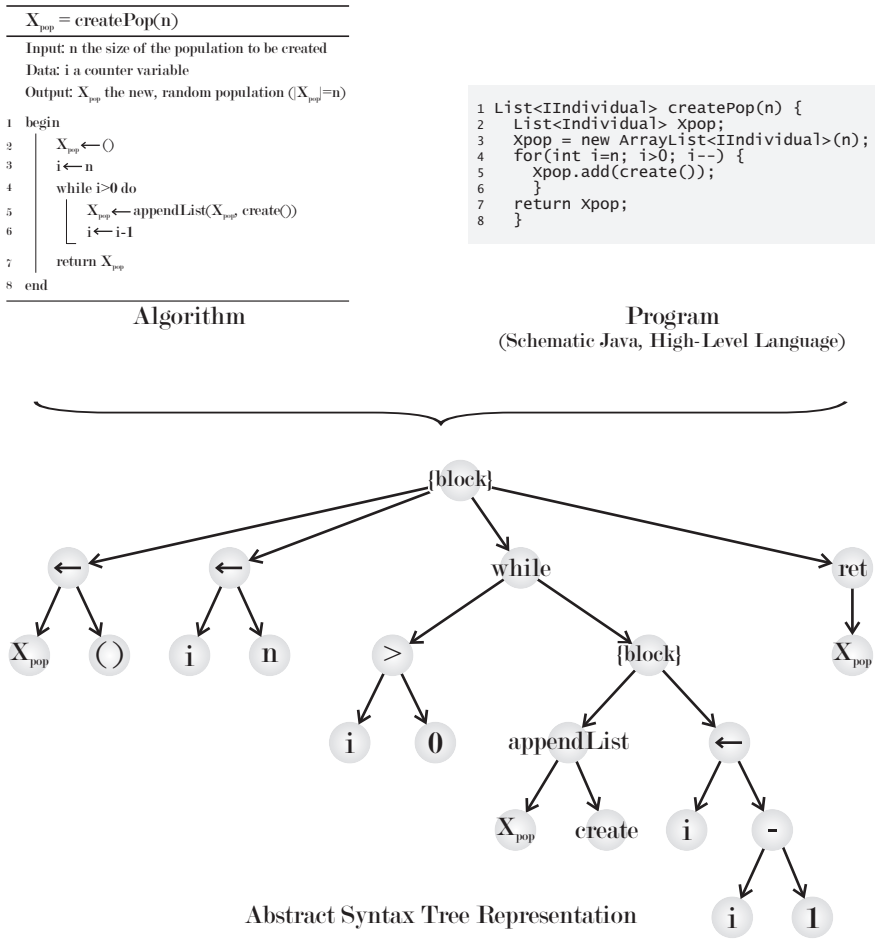


Fig. 4.2: The AST representation of algorithms/programs.

they first split it into tokens[9], parse[10] these tokens, and finally create an abstract syntax tree[11] (AST) [503, 504]. The internal nodes of ASTs are labeled by operators and the leaf nodes contain the operands of these operators. In principle, we can illustrate almost every[12] program or algorithm as such an AST (see Figure 4.2).

[9] http://en.wikipedia.org/wiki/Lexical_analysis [accessed 2007-07-03]

[10] http://en.wikipedia.org/wiki/Parse_tree [accessed 2007-07-03]

[11] http://en.wikipedia.org/wiki/Abstract_syntax_tree [accessed 2007-07-03]

[12] Excluding such algorithms and programs that contain jumps that would produce crossing lines in the flowchart (http://en.wikipedia.org/wiki/Flowchart [accessed 2007-07-03]).

Tree-based Genetic Programming now directly evolves individuals in this form, which also provides a very intuitive representation for mathematical functions it has initially been used for by Koza (see Figure 19.1 on page 330).

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 is also 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 approaches, for example artificial neural networks, generate black-box outputs, which are highly complicated if not impossible to fully grasp [505].

4.2 General Information

4.2.1 Areas Of Application

Some example areas of application of Genetic Programming are:

Application	References
symbolic regression	[11, 506, 507, 508] Section 19.1
grammar induction	[509, 510, 511, 512]
data mining and data analysis	[513, 514, 515, 516, 231, 501, 517] Section 18.1.2
logic function synthesis	[11, 518, 519]
circuit design and layout	[502, 520, 521, 522, 523, 524]
high-level circuit design, for FPGAs ect.	[525]
medicine	[526, 527, 528, 529]
breeding financial and trading rules	[530, 531, 532]
microwave antenna design	[533]
finding cellular automata rules	[534, 535, 536, 537]
learning of rules for geometric structures	[538]
automated programming	[539, 18, 540, 541, 542, 543]

automated programming of robots	[544, 542, 545, 546, 547]
evolving aggregation protocols	Section 20.1 on page 337 and [548]
deriving distributed algorithms and protocols	[549, 550, 551, 552, 547]
evolving agent behaviors	[553, 554, 555, 556, 557, 547, 558, 559, 560, 561, 562]
learning rules for OCR systems	[563, 564]
biochemistry, detecting proteins, and such and such	[565, 566]
evolving game players	[567]

See also Section 4.4.3 on page 158, Section 4.5.5 on page 168, and Section 4.7.2 on page 185.

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: 2007: Valencia, Spain, see [568]
2006: Budapest, Hungary, see [569]
2005: Lausanne, Switzerland, see [570]
2004: Coimbra, Portugal, see [571]
2003: Essex, UK, see [572]
2002: Kinsale, Ireland, see [573]
2001: Lake Como, Italy, see [574]
2000: Edinburgh, Scotland, UK, see [575]
1999: Göteborg, Sweden, see [576]
1998: Paris, France, see [577, 578]

GECCO: Genetic and Evolutionary Computation Conference
see Section 2.2.2 on page 62

GP: Annual Genetic Programming Conference
Now part of GECCO, see Section 2.2.2 on page 62
History: 1998: Madison, Wisconsin, USA, see [579, 580]
1997: Stanford University, CA, USA, see [581, 582]
1996: Stanford University, CA, USA, see [583, 584]

GPTP: Genetic Programming Theory Practice Workshop

http://www.cscs.umich.edu/gptp-workshops/ [accessed 2007-09-28]
History: 2007: Ann Arbor, Michigan, USA, see [585]
2006: Ann Arbor, Michigan, USA, see [586]
2005: Ann Arbor, Michigan, USA, see [587]
2004: Ann Arbor, Michigan, USA, see [588]
2003: Ann Arbor, Michigan, USA, see [589]

ICANNGA: International Conference on Adaptive and Natural Computing Algorithms
see Section 2.2.2 on page 63

Mendel: International Conference on Soft Computing
see Section 1.8.2 on page 42

4.2.3 Journals

Some journals that deal (at least partially) with Genetic Programming are (ordered alphabetically):

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.

4.2.5 Books

Some books about (or including significant information about) Genetic Programming are (ordered alphabetically):

Koza: *Genetic Programming, On the Programming of Computers by Means of Natural Selection* (see [11])

Koza: *Genetic Programming II: Automatic Discovery of Reusable Programs: Automatic Discovery of Reusable Programs* (see [590])
Koza, Bennett III, Andre, Keane: *Genetic Programming III: Darwinian Invention and Problem Solving* (see [591])
Koza, Keane, Streeter, Mydlowec, Yu, Lanza: *Genetic Programming IV: Routine Human-Competitive Machine Intelligence* (see [543])
Langdon, Polli: *Foundations of Genetic Programming* (see [18])
Langdon: *Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming!* (see [592])
Banzhaf, Nordin, Keller, Francone: *Genetic Programming: An Introduction – On the Automatic Evolution of Computer Programs and Its Applications* (see [539])
Kinnear: *Advances in Genetic Programming, Volume 1* (see [593])
Angeline, Kinnear: *Advances in Genetic Programming, Volume 2* (see [594])
Spector, Langdon, O'Reilly, Angeline: *Advances in Genetic Programming, Volume 3* (see [595])
Wong, Leung: *Data Mining Using Grammar Based Genetic Programming and Applications* (see [517])
Geyer-Schulz: *Fuzzy Rule-Based Expert Systems and Genetic Machine Learning* (see [596])

4.3 (Standard) Tree Genomes

Tree-based Genetic Programming (TGP) is, not alone for historical reasons, the most widespread Genetic Programming variant (see also Section 4.6 on page 177). In this section, we introduce the well-known reproduction operations applicable to tree genomes.

4.3.1 Creation

Before the evolutionary process can begin, we need an initial, randomized population. In genetic algorithms, we therefore simple created a set of random bit strings. For Genetic Programming, we do the same with trees instead of such one-dimensional sequences.

Normally (but not necessarily), there is a maximum depth d specified that the tree individuals are not allowed to surpass. Thus, the creation operation will return only return trees with a longest path between the root and the most distant node of not more than d. There three different ways for realizing the *create*() operation (see Definition 42 on page 99) for trees which can be distinguished according to the depth of the produced individuals.

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 (Figure 4.4) on the other hand creates trees where each non-backtracking path from the root to the leaf nodes is no longer than d

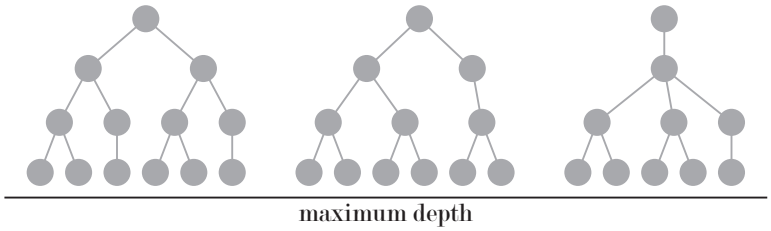


Fig. 4.3: Tree creation by the *full* method.

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 may be attached to.

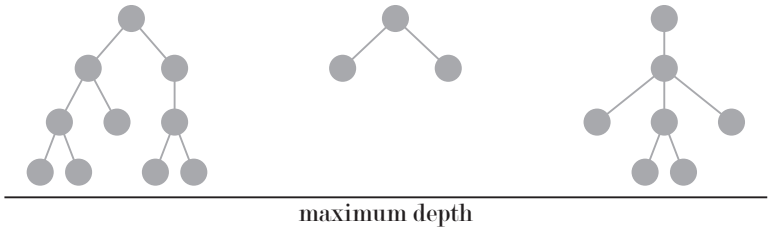


Fig. 4.4: Tree creation by the *grow* method.

Koza additionally introduced a mixture method called *ramped half-and-half* [11]. For each tree to be created, this algorithm draws a number r uniformly distributed between 2 and d: $(r = \lfloor random_u(2, d+1) \rfloor)$. Now either *full* or *grow* is chosen to finally create a tree with the maximum depth r (instead of d). This method is preferable since it produces an especially wide range of different tree depths and shapes and thus provides a great initial diversity.

4.3.2 Mutation

Analogously to nature, tree genotypes may undergo small variations called mutations during the reproduction process. Mutation on a tree is defined as randomly selecting a node, removing this node and all its children and replacing it with a new, randomly created one [11]. This general definition subsumes

- insertions of new nodes or small trees,
- replacement of existing nodes with others, or
- the deletion of nodes, as illustrated in Figure 4.5.

It is performed by the operation *mutate* which is introduced in Definition 44 on page 100.

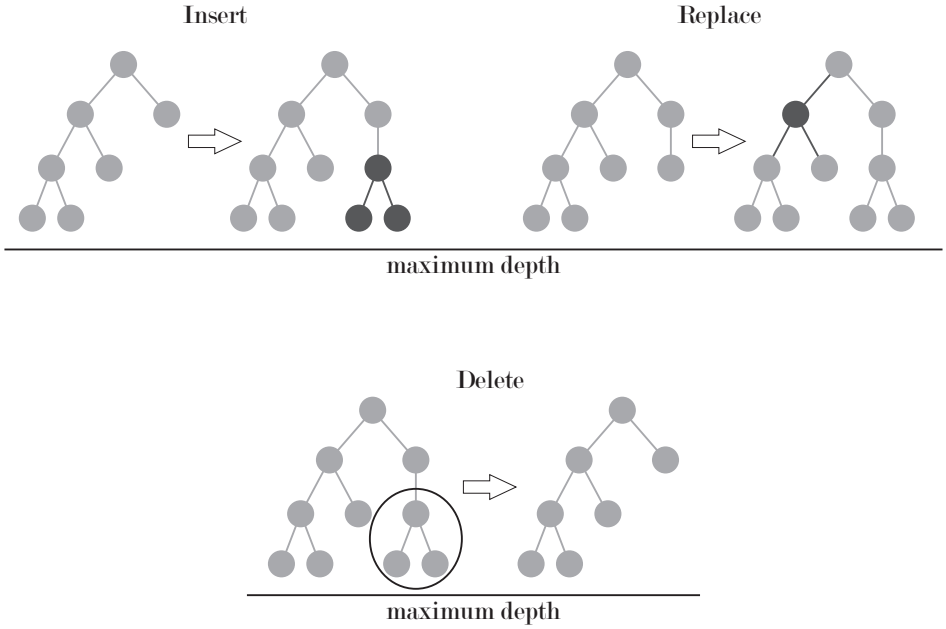


Fig. 4.5: Possible tree mutation operations.

4.3.3 Crossover

The mating process in nature, the recombination of the genotypes of two individuals, also takes place in tree-based Genetic Programming. Applying the *crossover*-operator (see Definition 45 on page 101) to two trees means to exchange sub-trees between them as illustrated in Figure 4.6. Therefore, one single sub-tree is selected randomly from each of the parents. They are cut out and reinserted in the partner genotype.

The intent of using the crossover 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 combined with good genes of different solution candidates. On the other hand, crossover in standard Genetic Programming has also a very destructive effect on the individual fitness [466, 597, 539].

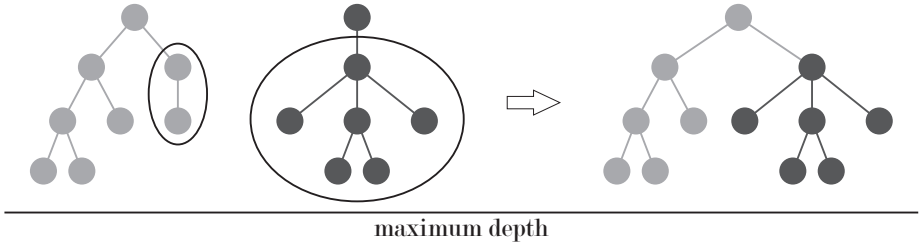


Fig. 4.6: Tree crossover by exchanging sub-trees.

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.

4.3.4 Permutation

The tree permutation operation illustrated in Figure 4.7 somewhat resembles the inversion operation of string genomes. Like mutation, it is used to reproduce one single tree asexually. 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 picked is commutative, it has 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 [11].

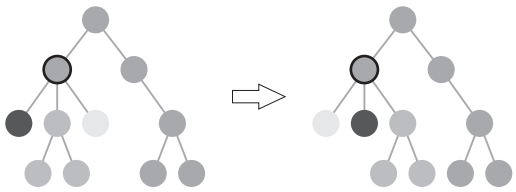


Fig. 4.7: Tree permutation – asexually shuffling sub-trees.

4.3.5 Editing

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 time that we need to compute it 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.

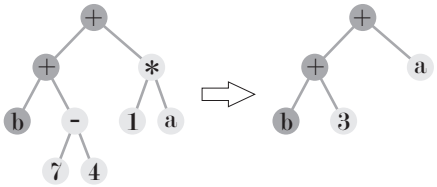


Fig. 4.8: Tree editing – asexual optimization.

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 crossover operations to pick "important" building blocks. At the same time, the expression $(7 - 4)$ is now less likely to be destroyed by crossover 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 reducing 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 [11].

4.3.6 Encapsulation

The idea behind the encapsulation operation is to a potentially useful sub-tree and making it an atomic building block as sketched in Figure 4.9. To put it plain, we create a new terminal symbol that (internally hidden) is a tree with multiple nodes. This way it 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. Again, this operation has no substantial effect but may be useful in special applications like the evolution of neural networks [11].

4.3.7 Wrapping

Applying the wrapping operation means to first select an arbitrary node n in the tree. Additionally, we create a new node non-terminal m outside of the

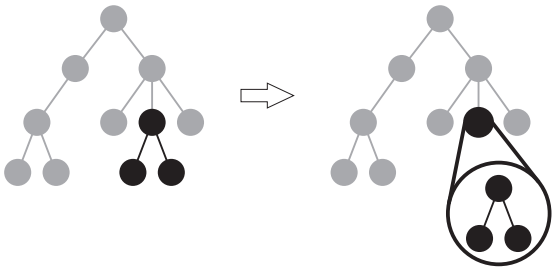


Fig. 4.9: An example for tree encapsulation.

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.

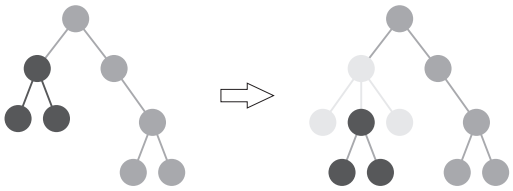


Fig. 4.10: An example for tree wrapping.

As 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, simple 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 is possible like $(b{+}3) + a \longrightarrow ((b{+}\mathbf{1}){+}3) + a$.

The wrapping operation is introduced by the author – at least, I have not seen another source where it is used.

4.3.8 Lifting

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.

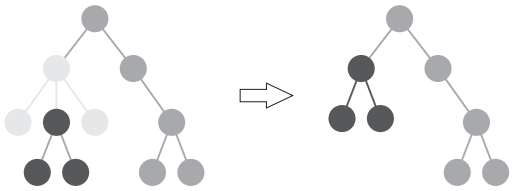


Fig. 4.11: An example for tree lifting.

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 20.1 on page 337). I, however, have not yet found other sources using a similar operation.

4.3.9 Automatically Defined Functions

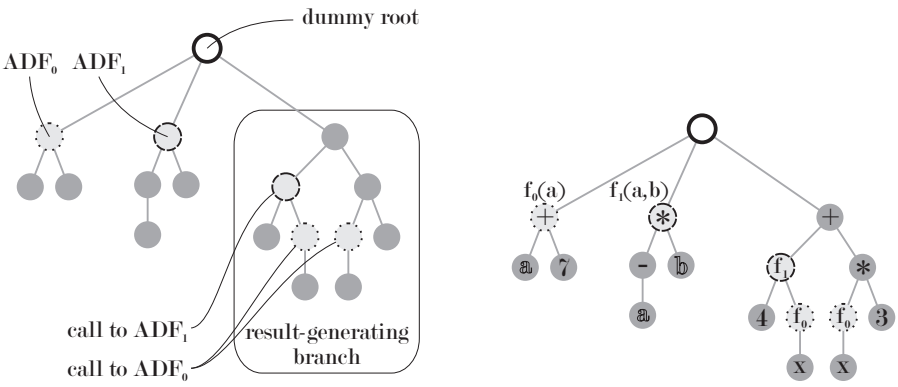
Automatically defined functions (ADF) [11] introduce some sort of pre-specified modularity into Genetic Programming. Finding a way to evolve modules and reusable building blocks is one of the key issues in using GP to derive higher-level abstractions and solutions to more complex problems [567, 590].

If ADFs are used, a certain structure is defined for the genome. The root of the trees looses its functional responsibility and now serves only as glue that holds the individual together. It has a fixed number of n children, from which $n-1$ are automatically defined functions and one is the result-generating branch. When evaluating the fitness of the individual, only this result-generating 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, 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 using an example from function approximation[13], since this is the area where the idea originally stems from. In Figure 4.12b we illustrate such a concrete example while Figure 4.12a outlines how a genotype with s could look like.

With this example, we want to approximate a function g with the one parameter x. We use a genome where two functions (lets call them f_0 and f_1)

[13] A very common example for function approximation, Genetic Programming-based symbolic regression, is discussed in Section 19.1 on page 329.



(a) general structure (b) example

Fig. 4.12: Automatically defined functions in Genetic Programming.

are automatically defined. f_0 has a single formal parameter a and f_1 has two formal parameters a and b. The genotype Figure 4.12a encodes the following mathematical functions:

$$\begin{aligned} f_0(a) &= a + 7 \\ f_1(a,b) &= (-a) * b \\ g &\approx f_1(4, f_0(x)) + (f_0(x) * 3) \end{aligned}$$

Hence, $g \approx ((-4) * (x + 7)) + (x + 7) * 3$. As you can see, 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 [11], they can also be applied to a variety of other problems like in the evolution of agent behaviors [555, 556, 598, 560], electrical circuit design [522], or the evolution of robotic behavior [545].

4.3.10 Node Selection

In most reproduction operations, in mutation as well a crossover, 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 crossover, 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 a very 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 should select all nodes c and n in the tree t with exactly the same probability as done by the method $uniformSelectNode$.

$$P(uniformSelectNode(c)) = P(uniformSelectNode(n)) \; \forall \; c, n \in t \quad (4.1)$$

Therefore, we define the weight of a tree node n, $nodeWeight(n)$, to be the total count of all of its descendants (children, grandchildren, grand-grandchildren, ...).

$$nodeWeight(n) = 1 + \sum_{i=1}^{|n.children|} nodeWeight(n.children[i]) \quad (4.2)$$

$$P(uniformSelectNode(c)) = \frac{1}{nodeWeight(t)} \; \forall \; c \in t \quad (4.3)$$

Thus, the $nodeWeight$ of the root of a tree is the number of all nodes in the tree and the $nodeWeight$ of its leafs is exactly 1. We can now reach each node in the tree t with an uniform probability (see Equation 4.3) by descending it from the root according to Algorithm 4.1.

Algorithm 4.1: $n = uniformSelectNode(t)$

Input: t the tree to select a node from
Data: c the currently investigated node
Data: $c.children$ the list of child nodes of c
Data: b, d two boolean variables
Data: r a value uniformly distributed in $[0, getWeight(c)]$
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) + 1)⌋
6          if r >= nodeWeight(c) then b ⟵ false
7          else
8              i ⟵ |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 i ⟵ i − 1
15     return c
16 end
```

A tree descend where with probabilities different from these defined here will lead to unequal node selection probability distributions. Then, the re-

production 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 (with also 0.5 probability). Then, the root t would have a 50% chance of being the starting point of reproduction operation. Its direct children have at most probability $\frac{0.5^2}{|t.children|}$ each, and their children even $\frac{0.5^3}{|t.children||t.children[i].children|}$ and so on. Hence, the leaves would almost never take actively part in reproduction.

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 3.5 on page 127) are used in many different Genetic Programming approaches. Here we illustrate them by the means of two examples: binary Genetic Programming and Gene Expression Programming. Moreover, many of the grammar-guided genetic approaches discussed in Section 4.5 on page 159 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 [494]. 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:

```
1 ;;Set variable V0 to have the value of V1
2 (:ZERO V0)
3 (:LOOP V1 (:INC V0))
4
5 ;;Multiply V3 by V4 and store the result in V5
6 (:ZERO V5)
7 (:LOOP V3 (:LOOP V4 (:INC V5)))
```

Listing 4.1: Two examples for the PL dialect used by Cramer for GP

Using a Genetic Algorithm working on integer strings for evolving his programs. He proposed two ideas on how to convert these strings to valid program trees.

The JB Mapping

The first approach is to divide the integer string to tuples of a fixed length 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, are ignored.

```
1 (0  4 2) → (:BLOCK AS4 AS2)
2 (1  6 0) → (:LOOP V6 AS0)
3 (2  1 9) → (:SET V1 V9)
4 (3 17 8) → (:ZERO V17) ;;the 8 is ignored
5 (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 example 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, for instance, be encoded as `(0 0 1 3 5 8 1 3 2 1 4 3 4 5 9 9 2)` which translates to

```
1 (0 0 1) ;;main statement       → (:BLOCK AS0 AS1)
2 (3 5 8) ;;auxiliary statement 0 → (:ZERO V5)
3 (1 3 2) ;;auxiliary statement 1 → (:LOOP V3 AS2)
4 (1 4 3) ;;auxiliary statement 2 → (:LOOP V4 AS3)
5 (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[14] – 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 and not to destroy the genotypes passed to them.

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.

[14] We come back to positional epistasis in Section 4.8.2 on page 187.

4.4.2 Binary Genetic Programming

With their Binary Genetic Programming (BGP) approach [456, 599, 600, 601], Keller and Banzhaf further explore the utility of explicit genotype-phenotype mappings and neutral variations in the genotype. 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[15], 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 a sentence in the target language works as follows:

1. $x \longleftarrow \epsilon$
2. Take the next gene (codon) $\mathtt{g}$ 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. 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[16] to the codon $\mathtt{g}$.
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 may still be an invalid sentence since there could have been 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 a sentence 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 [602, 603, 604] and are discussed in Section 3.7.3 on page 135 in detail.

From the form of the genome we assume the number of corrections needed (especially for larger grammars) in the genotype-phenotype mapping 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 185.

4.4.3 Gene Expression Programming

Gene Expression Programming (GEP) by Ferreira [605, 606, 607, 608, 609] introduces an interesting method for dealing with remaining unsatisfied function

[15] See Figure 3.2 on page 122 for more information on the DNA.
[16] see Definition 186 on page 574

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.

A gene in GEP is composed of a head and a tail [605] 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 identify parameterless expressions only. 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 4.4, where n is the arity (the number of arguments) of the function with the most arguments.

$$t = h(n-1)+1 \tag{4.4}$$

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 sentences 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 [605]. 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 4.13 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[17] of the nodes of the expression tree. In 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 tree 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 here because the number of allowed symbols is known before.

[17] `http://en.wikipedia.org/wiki/Tree_traversal` [accessed 2007-07-15]

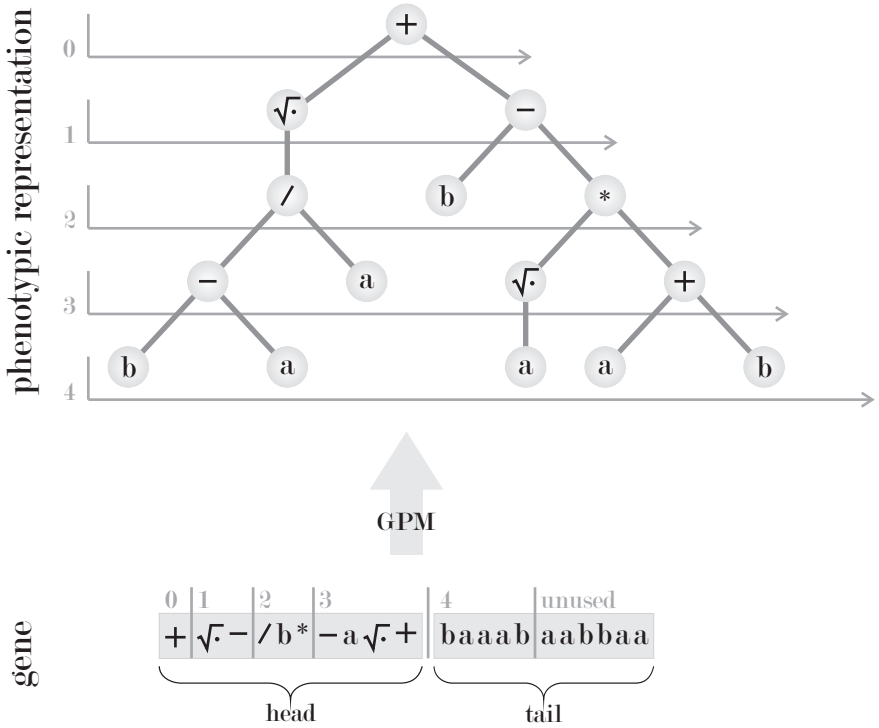


Fig. 4.13: A GPM example for Gene Expression Programming.

This fixed mapping is a disadvantage of Gene Expression Programming in comparison with the methods introduced in the following which have variable input grammars. On the other side 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 this chapter 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 185 for more details.)

General Information

Areas Of Application

Some example areas of application of Gene Expression Programming are:

Application	References
boolean function discovery	[610]
mathematical function discovery	[611, 612]
building classification and predicate association rules	[499, 500, 613, 614, 615, 616]
modeling electronics circuits	[617]
neural network design	[618, 619]
physical modeling	[620, 621, 622]

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.

4.5 Grammars in Genetic Programming

We have learned that the most common genotypic and phenotypic representations in Genetic Programming are trees. Furthermore, we have discussed the reproduction operations that are available for tree-based genomes. In this discussion we left one out important point: Reproduction cannot occur freely. In almost all applications, there are certain restrictions to the structure and shape of the trees that must not be violated. Take our pet-example symbolic regression[18] 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 to be seen in Figure 4.14.

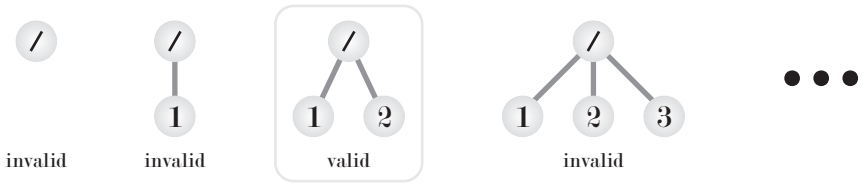


Fig. 4.14: Example for valid and invalid trees in symbolic regression.

There are four general methods how to avoid invalid configurations under these limitations:

[18] See Section 19.1 on page 329.

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 superfluous arguments (like c in Figure 4.14) would simple 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 mapping methods, we can prevent the creation of invalid trees from the start.

In this section, we will introduce some general considerations mostly regarding the latter approach.

A very natural way to express structural and semantic restrictions of a search space are formal grammars which are elaborated on in Section 37.3 on page 614. 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).

4.5.1 Trivial Approach

Standard Genetic Programming as introduced by Koza 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 GP, 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 yields a context-free grammar[19] 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)[20]. Here, the non-terminal symbol 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.

4.5.2 Strongly Typed Genetic Programming

Strongly typed Genetic Programming (STGP) developed by Montana [623, 624, 625] is an approach still very close to standard Genetic Programming

[19] see Section 37.3.2 on page 616 for details
[20] The Backus-Naur form is discussed in Section 37.3.4 on page 617.

```
<Z> ::= (<Z> + <Z>)
<Z> ::= (<Z> - <Z>)
<Z> ::= (<Z> * <Z>)
<Z> ::= (<Z> / <Z>)
<Z> ::= (sin <Z>)
<Z> ::= X
```

Listing 4.4: A trivial symbolic regression grammar.

that solves the problem of illegal parse trees that can occur when using data types in the evolved programs as illustrated in Figure 4.15.

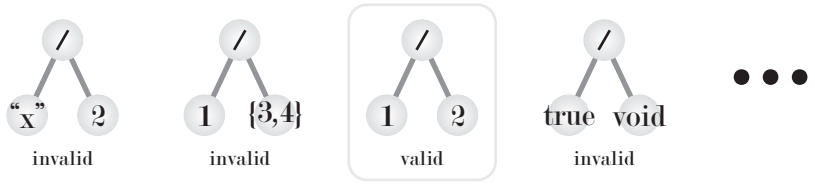


Fig. 4.15: Example for valid and invalid trees in typed Genetic Programming.

As already mentioned in Section 4.5.1 on the preceding page, in Standard Genetic Programming such errors are circumvented by only using representations that are type-safe per definition. In standard 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, so inconsistencies like in Figure 4.15 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 yet 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 mutation and creation operators now create valid individuals per default. Crossover 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 its type matches. If this is not possible, either the parents are returned directly or nothing is returned.

STGP also introduces interesting new concepts like generic functions and data types, very much like in Ada or C [625] and hierarchical type systems,

comparable to object-oriented programming in their inheritance structure [626]. This way, STGP increases the reusability and modularity of individual parts in GP which is needed for solving more complex problems [567, 590].

4.5.3 Early Research in GGGP

Research steps into grammatically driven program evolution can be traced to the early 1990s where Antonisse developed his Grammar-based Genetic Algorithm [627]. 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[21] of that grammar. Then, crossover was performed similar 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 [505]. Antonisse's early example was followed by others like Stefanski [628], Roston [629], and Mizoguchi et al. [630].

In the mid-1990s [505, 631], more scientists began to concentrate on this topic. The LOGENPRO system developed by Wong and Leung [632, 633, 634, 635, 636, 637, 517] used PROLOG Definite Clause Grammars to derive first-order logic programs. The system proposed by Whigham applied context-free grammars [638, 639, 640, 641] in order to generate populations of derivation trees. His system additionally had the advantage that it allowed the user to bias the evolution into the direction of certain parts of the grammar [641]. Schulz derived a similar system [596], differing mainly in the initialization procedure [642, 505], for learning rules for expert systems. The Genetic Programming Kernel (GPK) by Hörner [643] used tree-genomes where each genotype was a deviation tree generated from a BNF definition.

4.5.4 Gads 1

The Genetic Algorithm for Deriving Software 1 (Gads 1) by Paterson and Livesey [644, 645] was one of the basic research projects that paved the way for other more sophisticated approaches like grammatical evolution. Like the binary Genetic Programming system of Keller and Banzhaf, it uses a clear distinction between genotype and phenotype. The genotypes $g \in \mathbb{G}$ in Gads are fixed-length integer strings which are transformed to character string phenotypes $x \in \tilde{X}$ (representing program syntax trees) by a genotype-phenotype mapping (see Section 3.5 on page 127). Because of this genome, Gads can use a conventional genetic algorithm engine[22] 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

[21] An elaboration on derivation trees can be found in Section 37.3.3 on page 616.

[22] Gads 1 uses the genetic algorithm C++ class library GAGS (`http://geneura.ugr.es/GAGS/` [accessed 2007-07-09]) release 0.95.

```
 1  (0) <expr>    ::= <expr> <op> <expr>
 2  (1) <expr>    ::= (<expr> <op> <expr>)
 3  (2) <expr>    ::= <pre-op> (<expr>)
 4  (3) <expr>    ::= <var>
 5
 6  (4) <op>      ::= +
 7  (5) <op>      ::= -
 8  (6) <op>      ::= /
 9  (7) <op>      ::= *
10
11  (8) <pre-op>  ::= log
12  (9) <pre-op>  ::= tan
13 (10) <pre-op>  ::= sin
14 (11) <pre-op>  ::= cos
15
16 (12) <var>     ::= X
17
18 (13) <func>    ::= double func(double x){
19                          return <expr>;
20                    }
```

Listing 4.5: A simple grammar for C functions that could be used in Gads.

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.

Although Gads was primarily tested with LISP S-expressions it can evolve sentences according to all possible 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 $\Sigma = \{\texttt{sin}, \texttt{cos}, \texttt{tan}, \texttt{log}, \texttt{+}, \texttt{-}, \texttt{*}, \texttt{/}, \texttt{X}, \texttt{()}, \ldots\}$ 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 may be defined for the same non-terminal symbol.

Every gene in a Gads genotype contains the index of the productions in G to be applied next. For now, let us assume 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

```
1 double func(double x){
2   return <expr>;
3 }
```

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 expr-non-terminal 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[23]. 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:

```
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 spare 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 exprs to vars 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[24] [188] 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.

[23] An allele is a value of specific gene, see Definition 52 on page 123.

[24] Introns are genes or sequences of genes (in the genotype) that do not contribute to the phenotype or its behavior, see Definition 54 on page 123

```
1  (A) <expr>    ::= <expr> <op> <expr>     (0)
2                   | (<expr> <op> <expr>)  (1)
3                   | <pre-op> (<expr>)     (2)
4                   | <var>                 (3)
5
6  (B) <op>      ::= +                      (0)
7                   | -                      (1)
8                   | /                      (2)
9                   | *                      (3)
10
11 (C) <pre-op>  ::= log                    (0)
12                  | tan                    (1)
13                  | sin                    (2)
14                  | cos                    (3)
15
16 (D) <var>     ::= X                      (0)
17
18 (E) <func>    ::= double func(double x){
19                       return <expr>;
20                  }                       (0)
```

Listing 4.6: A simple grammar for C functions that could be used by GE.

4.5.5 Grammatical Evolution

Like Gads, grammatical evolution[25] (GE), developed by Ryan, Collins, and O'Neill [631], creates expressions in a given language by iteratively applying the rules of a grammar specified in the Backus-Naur form [631, 646, 647].

In order to discuss how grammatical evolution works, we re-use the example of C-style mathematical functions [631] as in Section 4.5.4. Listing 4.6 shows the 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 with `|`). In each rule, options are numbered started with `0`. When the symbol `<exp>` for example is expanded, 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. However, 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 during the genotype-phenotype mapping by the non-terminal symbol to be expanded and we only need to

[25] `http://en.wikipedia.org/wiki/Grammatical_evolution` [accessed 2007-07-05]

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 be evolved using genetic algorithms [631, 648] (like in Gads) or with other techniques, like particle swarm optimization [649, 650] or differential evolution [651]. As illustrated in Figure 4.16, 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 [652].

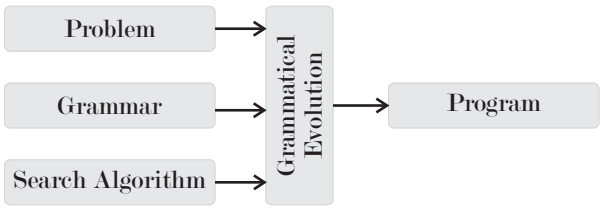


Fig. 4.16: The structure of a grammatical evolution system [652].

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 $n \in N$ 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 [631], 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 granularity for the genome. As we may have less than 256 options, we apply modulo arithmetics to get the index of the option. This way, the sequence $g = (193, 47, 51, 6, 251, 88, 63)$ would be 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

```
1 double func(double x){
2  return <expr>;
3 }
```

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. There are three possible approaches how to proceed:

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 GE. It has, of course, 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 the aforementioned default rules.

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[26], but on basis of derivation trees[27]. Therefore, the number of the choices made during a random grammatical rule expansion beginning at the start symbol is 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
solving trigonometric equalities (vs. standard GP: [11])	[648, 631]
creating program source code in different programming languages	[647, 653, 654]
the evolution of robotic behavior (vs. standard GP: [542, 544])	[546, 655]
breeding financial and trading rules (vs. standard GP: [530, 531])	[532, 656, 657, 658]

There even exists an approach called "Grammatical Evolution by Grammatical Evolution" ($(GE)^2$) where the grammar defining the structure of the solution candidates itself is co-evolved with the individuals represented in it [659].

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]

[26] An initialization method of standard, tree-based Genetic Programming that creates a good mixture of various tree shapes [11], see Section 4.3.1 on page 146 for more details.

[27] see Section 37.3.3 on page 616

History: 2004: Seattle, WA, USA, see [660]
2003: Chicago, IL, USA, see [661]
2002: New York, NY, USA, see [309]

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.

4.5.6 Gads 2

In Gads 2, Paterson 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;
```

It is obviously not a well-typed program although syntactically correct. Context-sensitive grammars[28] allow productions like $\alpha A\beta \rightarrow \alpha\gamma\beta$ 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 $\gamma \neq \varepsilon$ 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 with a context-sensitive grammar. Paterson argues that the application of existing approaches like two-level grammars and standard attribute grammars[29] in Genetic Programming is infeasible [188] and introduces an approach based on *reflective attribute grammars*.

Definition 64 (Reflective Attribute Grammar). A reflective attribute grammar (rag[30]) [188] is a special form of attribute grammar. 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.

[28] See Section 37.3.2 on page 616 where we discuss the Chomsky Hierarchy of grammars.

[29] See Section 37.3.6 on page 619 for a discussion of attribute grammars.

[30] Notice that the shortcut of this definition *rag* slightly collides with the one of recursive attribute grammars (*RAG*) introduced by Shut in [662] and discussed in Section 37.3.8 on page 623, although their letter cases differ. To the knowledge of the author, rags are exclusively used in Gads 2.

The transformation of a genotype 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 compared 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 genotypes with genes in one-byte granularity and encounter a production with four options while the next gene has the value 45. In GE, 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 [663]. The experimental results in [188], although successful, do not provide substantial benefits compared to the simpler grammatical evolution approach.

Here 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 and reaching good fitness values with overfitting.

Like grammatical evolution, the Gads 2 idea can be realized with arbitrary genetic algorithm engines. The experiments in [188] used the Java-based evolutionary computation system ECJ by Luke et al. [664] as genetic algorithm engine.

4.5.7 Christiansen Grammar Evolution

Christiansen grammars (as described in Section 37.3.9 on page 624) have many similarities to reflective grammars as used in Gads 2. They are both (extended) attribute grammars[31] 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 [665].

Building on their previous work [666], Ortega, de la Cruz, and Alfonseca place the idea of Gads 2 on the solid foundation of Christiansen grammars with their Christiansen grammar evolution approach (CGE) [667]. 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 $G = (N, \Sigma, P, S)$ 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 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

[31] See Section 37.3.7 on page 621 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.

`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 37.11 on page 624. `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 4.5.5 on page 165.

4.5.8 Tree-Adjoining Grammar-Guided Genetic Programming

A different approach to grammar-driven Genetic Programming has been developed by Nguyen [466, 668, 669, 670, 671, 672] with his Tree-Adjoining Grammar-Guided Genetic Programming (TAG3P) system. 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, discussed in detail in Section 37.3.10 on page 625).

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.

Let us take a look back on the tree-adjoining representation of our earlier example grammar G in listing 4.6 on page 165 for mathematical functions in C. Figure 4.17 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).

The TAG3P approach has in common with the other grammar-guided methods that it 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 37.3.3 on page 616.

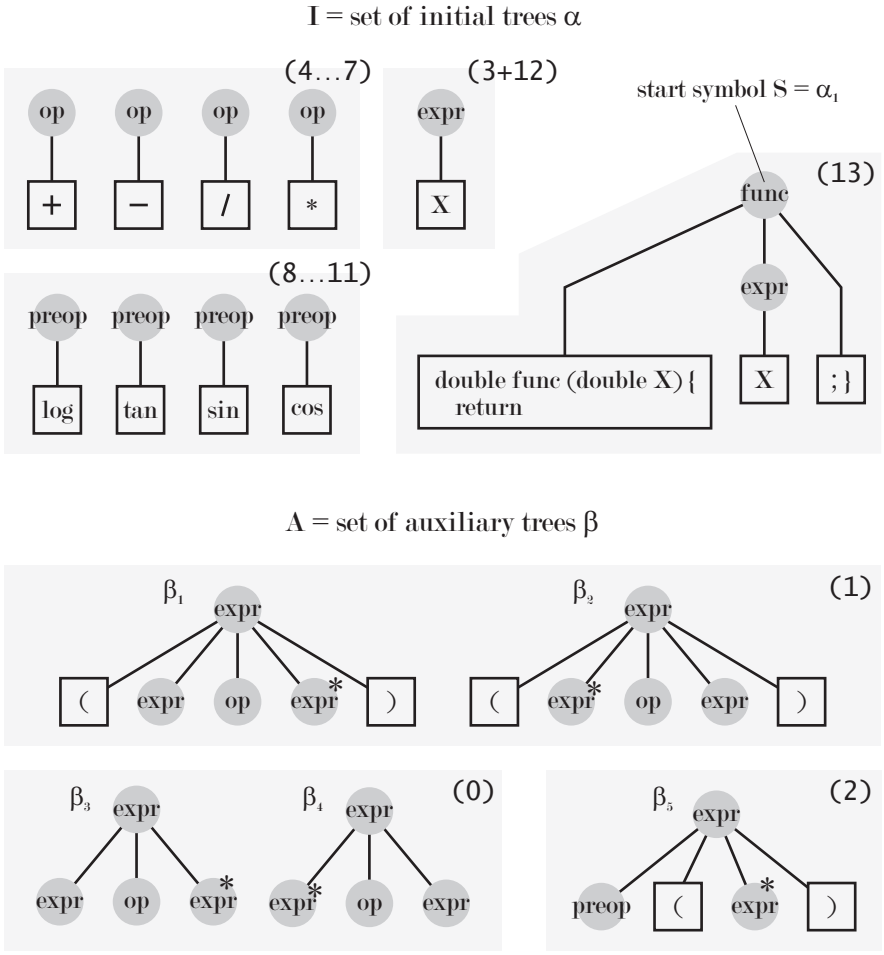


Fig. 4.17: An TAG realization of the C-grammar of listing 4.6.

Derivation Trees

For tree-adjoining grammars, there exist different types of derivation trees [466]. In Weir's definition in [673], 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[32]. The number of

[32] `http://en.wikipedia.org/wiki/Tree_traversal` [accessed 2007-07-18]

adjunctions performed with each node is limited to one. Substitution operations, not possible with the Weir method, are enabled by Joshi's and Schabes' extension [674]. In their notation (not illustrated here) a solid connection between two nodes in the derivation tree stands for "and" adjunction, whereas a broken line denotes a substitution.

In TAG3P, Nguyen uses a restricted form of such TAG derivation trees [466] 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 4.18 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 4.17 we 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 according derivation tree, they are simple attached as a list of lexemes. A similar substitution can be performed 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 and 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.

Summary

Tree-adjoining grammar-guided Genetic Programming is a different approach to grammar-based Genetic Programming which has some advantages compared with the other methods. Maybe its biggest plus 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

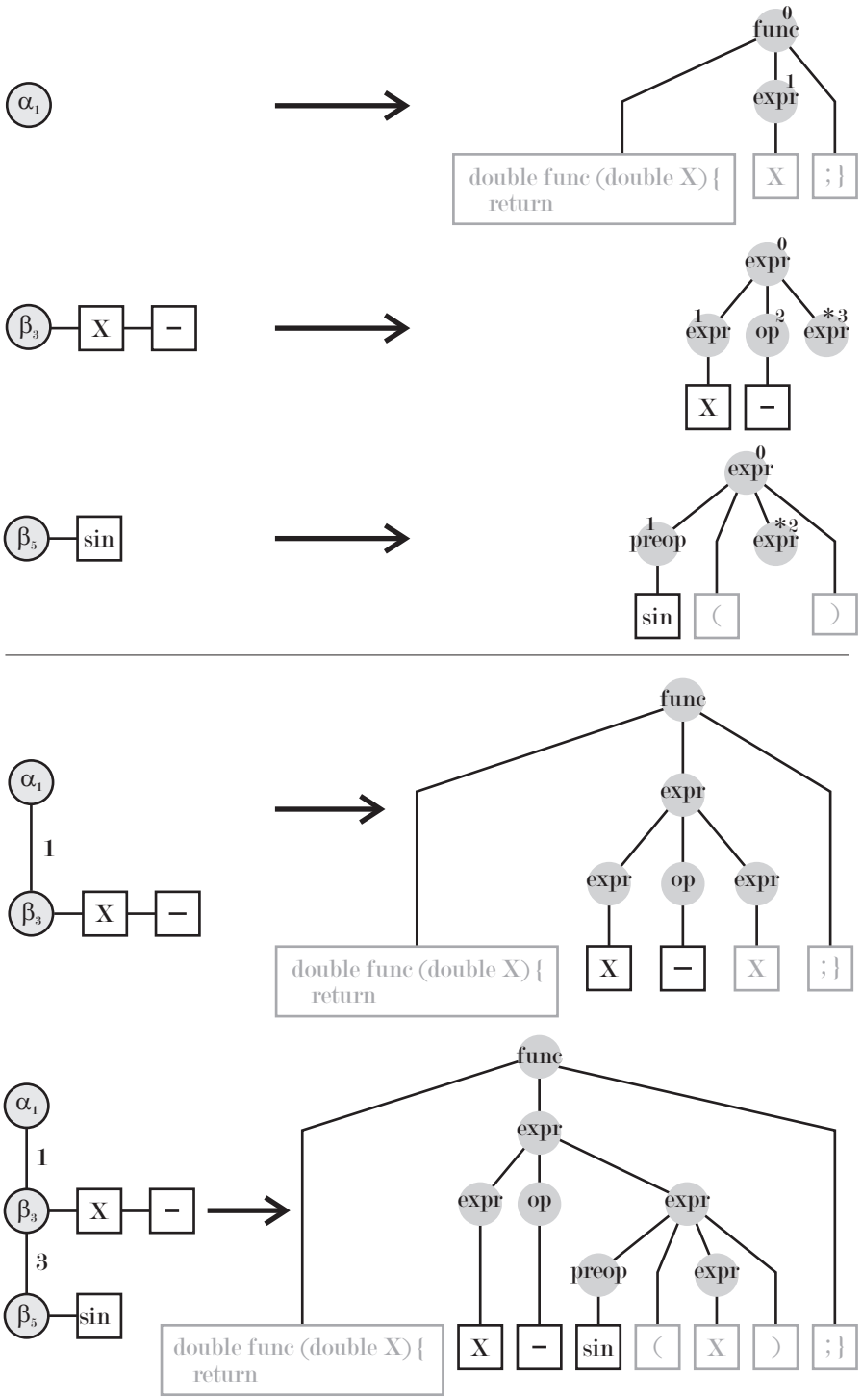


Fig. 4.18: One example genotype-phenotype mapping in TAG3P.

one bit in a grammar 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 general principle holds that small changes in the genotype should 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 149 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 4.18 with itself, for example.

General Information

Areas Of Application

Some example areas of application of tree-adjoining grammar-guided genetic programminn are:

Application	References
symbolic regression	[669, 670]
finding trigonometric identities	[675, 676]
logical function synthesis	[671]

Online Resources

Some general, online available ressources on tree-adjoining grammar-guided genetic programminn 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

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.

Trees are however not the only way of representing programs. Matter of fact, a computer processes them in form of a sequence of instructions. This

sequence may contain branches in form of jumps to other places in the program. 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 by Friedberg in 1958 [488, 489] used a fixed-length instruction sequence genome.

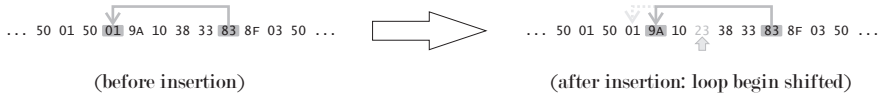
The area of Genetic Programming that works with such instruction string genomes is called *linear Genetic Programming* (LGP) [677, 678, 679, 680, 681, 682] in contrast to the traditional *tree-based Genetic Programming* (TGP). It can furthermore be distinguished from approaches like grammatical evolution (see Section 4.5.5 on page 165) by the fact that strings there are just genotypic, intermediate representations that encode the program trees. Here however, they are the center of the whole evolution and contain the program code directly.

Simple reusing the genetic operators for variable-length string genomes, introduced in Section 3.4.2 on page 126 that randomly insert, delete, or toggle bits, is however not really feasible [682].

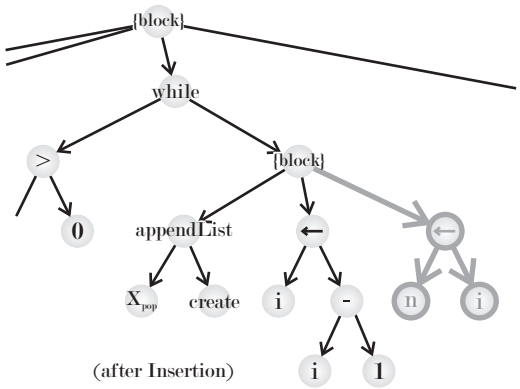
We must visualize the alternatives and loops that we know from high-level programming languages are mapped to conditional and unconditional jump instructions in machine code. These jumps target to either absolute or relative addresses inside the program. Let us for example take the insertion of a single, new command into the instruction string, maybe as result of a mutation or crossover operation. If we do not perform any further corrections after this insertion, it is well possible that the resulting address-shift invalidates the control flow and renders the whole program useless as illustrated in 4.19a.

Tree-based genomes on the other hand are less vulnerable to such insertions – the loop in stays intact, although one useless instruction richer.

The advantage of linear Genetic Programming lies in the easy evaluation of the evolved algorithms. Its structure allows for limiting the runtime in individual evaluation and even to simulate parallelism.



(a) Inserting into an instruction string.



(b) Inserting in a tree representation.

Fig. 4.19: The impact of insertion operations in Genetic Programming.

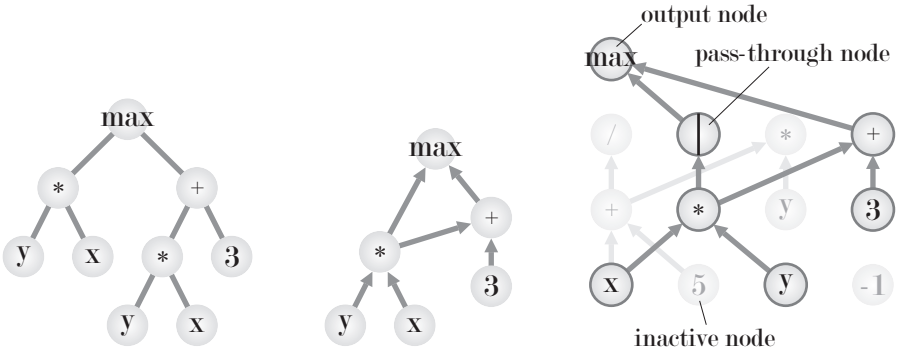
4.7 Graph-based Approaches

In this section we will discuss some Genetic Programming approaches that are based on graphs rather than on trees or linear sequences of instructions.

4.7.1 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 in the mid 1990s [683, 684, 685, 686]. 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.

In order to illustrate this structure, we use the formula term $\max\{x*y, x*y+3\}$ as example. We already have elaborately discussed how we can express mathematical terms as trees. Figure 4.20a illustrates a such a function tree. Using a directed graph, as outlined in Figure 4.20b, 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, we first have to define a grid structure. In Figure 4.20c, we settle 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 Figure 4.20b into this grid. This



(a) tree structure (b) graph structure (c) PDGP structure

Fig. 4.20: The term $\max\{x * y, x * y + 3\}$

leaves some nodes unoccupied. If we assume that Figure 4.20c represents a solution grown by this Genetic Programming approach, these nodes would be labeled with some unused expressions and would somehow be connect 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 Figure 4.20c. The nodes that have influence on the program's result,i. e. those which are connected to a output node directly or indirectly, are named *active* nodes.

We may impose restrictions on the connectivity of PDGP graphs. For instance, we can define that each row must only be connected 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 Figure 4.20c. 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 [687]. However, we can as well do without any form of restrictions for the connective and may allow backward connections in the programs, depending on the application.

An interesting part of PDGP is how we execute the programs. Principally, it allows for a great proportion of parallelism. Coming back to the example outlined Figure 4.20c, 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 representation, new 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 145, 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

SAAN Crossover The basic crossover operation in PDGP is *Sub-graph Active-Active Node (SAAN) crossover*. 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.

The idea of SAAN crossover is that active sub-graphs represent functional units and this way, we explore new combinations of them.

SSAAN Crossover The *Sub-Sub-Graph Active-Active Node (SSAAN) Crossover* method works essentially the same like SAAN, 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 there would be no arguments which 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

We can extend the mutation operation from Standard Genetic Programming easily to PDGP. Then, a new, random graph is created and inserted at a random point 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, 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[33] 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 [688, 689].

4.7.2 Cartesian Genetic Programming

Cartesian Genetic Programming (CGP) was developed by Julian Miller and Peter Thomson [690, 691, 692] in order to achieve a higher degree of effectiveness in learning Boolean functions. In his 1999 paper [691], Miller 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 4.21, the $i = 6$ input variables $X_1 \ldots X_3$ and $Y_1 \ldots Y_3$, placed to the left, are numbered from 0 to 5. As function set, we could use $\{+ = 0, - = 1, * = 2, / = 3, \vee = 4, \wedge = 5, \oplus = 6, \neg = 7\}$. Like in PDGP, before the evolution begins, we define a grid of cells. 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 4.21.

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 Climber. The genotype of Cartesian Genetic Programming is a fixed-length integer string. It consists 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 4.21) denoting the function it will carry

[33] More information on ADFs can be found in Section 4.3.9 on page 151.

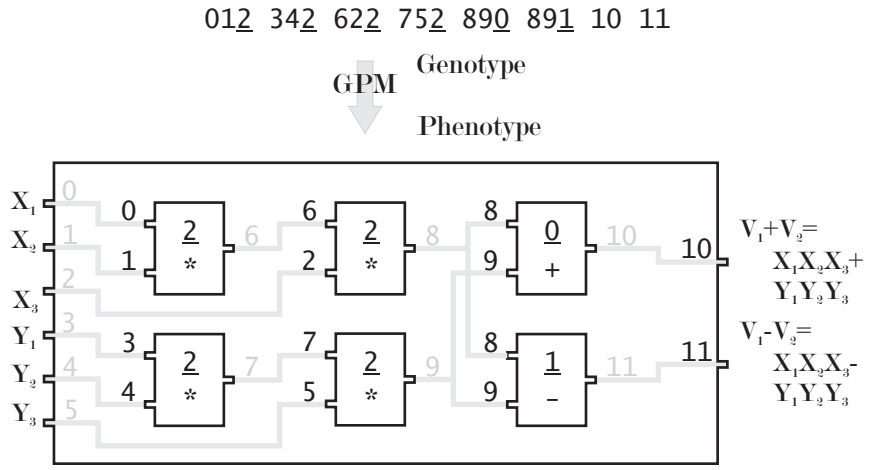


Fig. 4.21: An example for the GPM in Cartesian Genetic programming.

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 cells are necessarily are connected with the output-producing cells. Furthermore, not all functions need to incorporate all i' inputs into their results. $\neg$ for instance, which is also part of the example function set, 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 also being connected with `0-5`. Of course, the reproduction operations have to respect the levels-back value set.

Although CGP labeled itself a Genetic Programming technique from the beginning. However, crossover was often absent in many of the work contributed about it. So one could regarded it also as an Evolutionary Programming[34] method. Lately, researchers begin also to focus on efficient crossover techniques for CGP [693].

Neutrality in CGP

Cartesian Genetic Programming explicitly utilizes different forms of neutrality[35] in order to foster the evolutionary progress. Normally, neutrality can

[34] See Chapter 6 on page 209 for more details on Evolutionary Programming.

[35] See Section 1.4.4 on page 26 and Section 3.7.3 on page 135 for more information on neutrality and redundancy.

have positive as well as negative effects on the evolvability of a system. In [83, 694], Yu and Miller outline different forms of neutrality in Cartesian Genetic Programming which also apply to other forms of GP or GAs:

- Inactive genes configure 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*.
- 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 [82].

Embedded Cartesian Genetic Programming

In 2005, Wagner and Miller developed *Embedded* Cartesian Genetic Programming (ECGP), a new type of CGP with a module acquisition [695] method in form of automatic module creation [696, 697, 698]. Therefore, three new operations are introduced:

- **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".
- **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.
- 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
circuit design and layout	[699, 82, 690, 700, 701, 697]
learning Boolean functions	[691, 692, 83]
symbolic regression	[692]
evolving robot controllers	[702, 703]
prime number prediction	[704]

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 grammar-guided Genetic Programming methods. 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.

Whenever we use such methods to evolve real algorithms, we encounter the problem of epistasis. In an algorithm, each instruction depends on the instructions that have been executed before. The result of one instruction will influence the behavior of those executed afterwards. Besides this general dependency, we can observe a set of other epistatic effects which we will introduce in this section.

4.8.1 Problems of String-to-Tree GPMs

In this chapter, we have learned about multiple grammar-guided Genetic Programming methods that employ a genotype-phenotype mapping between an (integer) string genome and trees that represent sentences in a given grammar, like grammatical evolution, Christiansen grammar evolution, Gads, and binary Genetic Programming (BGP).

According to [631], the idea of mapping string genotypes can very well be compared to one of the natural antetypes of artificial embryogeny[36]: the translation of the DNA into proteins. This process depends very much on the proteins already produced and now present around the cellular facilities. If a certain piece of DNA has created a certain protein X and is transcripted again, it may result into a molecule of protein type Y because of the presence of X.

Although this is a nice analogy, it also bears an important weakness. These genomes usually violate the causality[37] [505]. In Grammatical Evolution for example, a change in any gene in G will also change the meaning of all subsequent alleles. This means that mutation and crossover will probably have very destructive impacts on the individuals [466]. Additionally, 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 4.22 outlines how the change of a single bit 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.

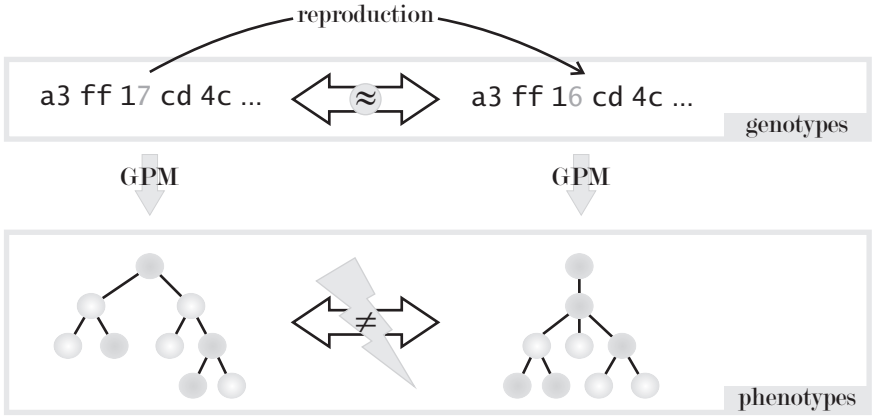


Fig. 4.22: Epistasis in Grammatical Evolution.

The lack of causality is rooted in a strong epistasis[38] in the string-tree-GPM approaches: many loci of the genotypes have some mutual influence. The efficiency of the reproduction operations of the mentioned approaches

[36] Find out more about artificial embryogeny in Section 3.5.1 on page 128.
[37] The principles of causality and locality are discussed in Section 3.7.1 on page 134.
[38] Epistasis is elaborated on in Section 3.7.2 on page 135.

will probably decrease with a growing set of non-terminal symbols and corresponding productions.

One way to decrease these effects is (implicitly) inherent in the Gene Expression Programming GPM approach (see Section 4.4.3 on page 156). An a-priori division of genotypes into different sections, each independently encoding different parts of the phenotype, can reduce the harmful influences of the reproduction operations.

4.8.2 Positional Epistasis

In order to clarify the role of positional epistasis in the context of Genetic Programming, we begin with some basic assumptions. Let us consider a program P as a form 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 $P_1(i) = P_2(i) \ \forall i \in I$.[39]

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 n statements, there are $n!$ possible permutations. We argue that the fraction $\theta(P) = \frac{v}{n!}$ of permutations v that leads to programs equivalent to P is a measure of robustness for a given phenotypic representation in Genetic Programming. More precisely, a low value of θ indicates a high degree of epistasis, which means that the loci (the positions) of many different genes in a genome have influence on their functionality [74]. This reduces for example the efficiency of reproduction operations like crossover, since they often change the number and order of instructions in a program. A general rule in evolutionary algorithms is thus to reduce epistasis [469] (see rule 9 in Section 3.7 on page 133).

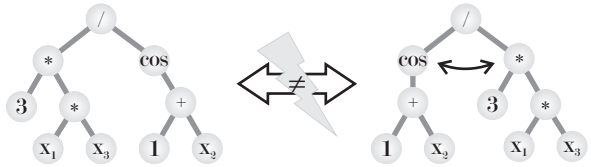
Many of the phenotypic and most genotypic representations in Genetic Programming mentioned so far seem to be rather fragile in terms of insertion and crossover points. One of the causes is that their genomes have high positional epistasis (low θ-measures), as sketched in Figure 4.23.

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 or even better solutions. We point out here just some classes of problems that, if successfully solved, could even increase their utility.

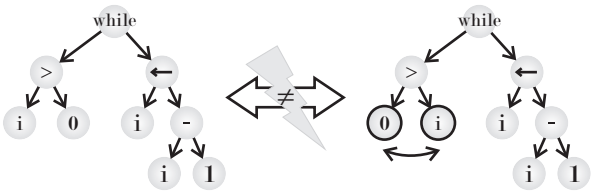
4.9 Rule-based Genetic Programming

There exists one class of evolutionary algorithms that elegantly circumvents the positional epistasis discussed in Section 4.8.2: the learning classifier systems (LCS) [705, 706] which you can find discussed in Chapter 7 on page 211.

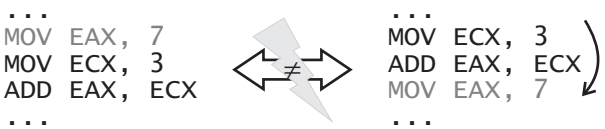
[39] In order to cover stateful programs, the input set may also comprise sequences of input data.



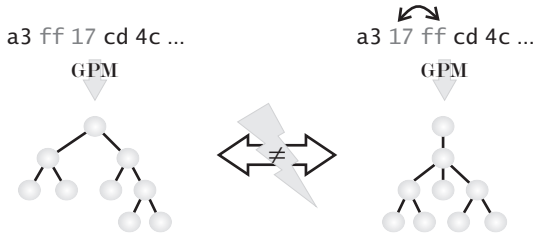
(a) In Standard Genetic Programming and Symbolic Regression



(b) In Standard Genetic Programming.



(c) In Linear Genetic Programming.



(d) With Genotype-Phenotype Mapping, as in Grammatical-Evolution like approaches.

Fig. 4.23: Positional epistasis in Genetic Programming.

Here we focus on the Pittsburgh approach associated with Smith and De Jong [707, 708], where a genetic algorithm evolves a population of rule sets. 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 for rules concerning the same output bits, i. e. $\theta(C) \approx 1$.

The basic idea behind rule-based Genetic Programming approach is to use this knowledge to create a new individual representation that retains these high θ values in order to become more robust in terms of reproduction operations. This will probably lead to a smoother evolution with a higher probability

of finding good solutions. On the other hand, it also extends LCS by introducing some of the concepts from Genetic Programming like mathematical operations.

This approach to Genetic Programming is illustrated by the example shown in Figure 4.24. Like in Pitt-style learning classifier systems, the programs consist of arbitrary many rules. A rule evaluates the values of some symbols in its condition part (left of ⇒) and, in its action part, assigns a new value to one symbol or performs any other procedure specified in its setup.

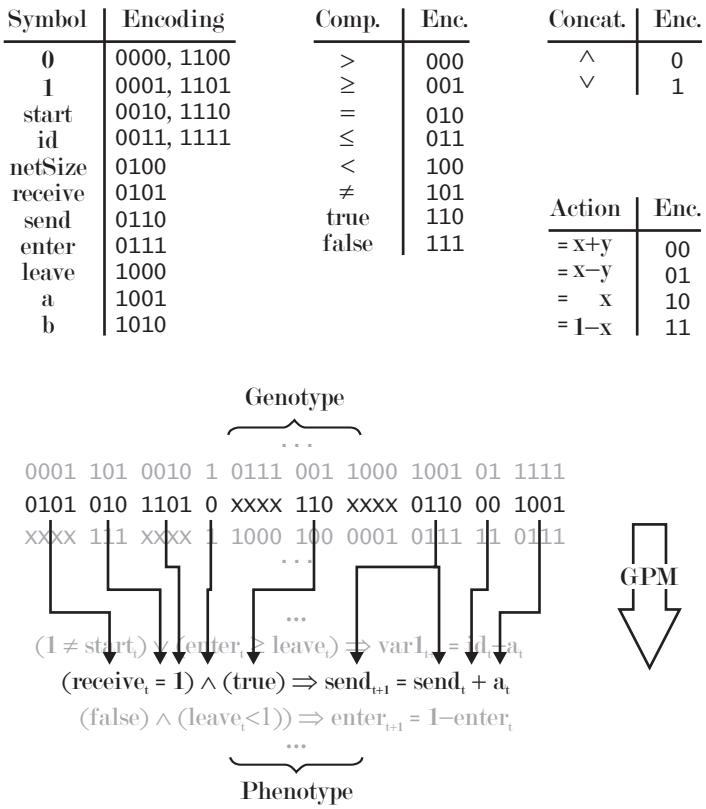


Fig. 4.24: Genotype-Phenotype mapping in Rule-based Genetic Programming.

4.9.1 Genotype and Phenotype

Before the evolution begins, the number of symbol 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. Generally we also define some constants like `0` and `1` and an input symbol `start` which will only be `1` during the first execution of the program and then becomes `0` (but may be written to the program). Additionally, a program can be provided with some general-purpose variables (`a` and `b` in the example). We can introduce further symbols with special meanings. If we, for instance, want to evolve distributed algorithms, we could add an input symbol `receive` where incoming messages will occur and a variable `send` from which outgoing messages could be transmitted. An action set containing mathematical operations like addition, subtraction, value assignment, and some sort of logical negation (`1-x`) is sufficient in most cases but may arbitrarily be extended. Alternatively to the `send` and `receive` symbol, actions could be defined with the same semantics.

From these specifications, the system can determine how many bits are needed to encode a single rule. The genotypes are bit strings with a length which is a multiple of this bit count.

With this simple genotype, we can encode any possible nesting depth of condition statements and all possible logical operations. Furthermore, we could even construct a tree-like program structure from the rules, since each of them corresponds to a single `if` statement in a normal programming language.

There are similarities between our RBGP and some special types of LCS, like Browne's abstracted LCS [709] and S-expression-based LCS [710]. 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 LCS. They 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 and performs an asynchronous and interactive computation with its environment. Furthermore, the syntax of RBGP is very extensible. The nature of the symbols and actions is not bound to specific data types, our approach can for example easily be adapted to floating point computation.

4.9.2 Program Execution and Dimensions of Independence

The simplest method to execute a rule-based program would be to loop over it in a cycle. Although this approach is sufficient for simulation purposes, it would result in a large waste of CPU power on a real system. This consumption of computational power can very much be decreased if the conditional parts of the rules are only evaluated when one of the symbols that they access changes.

Positional Independence

Such changes can either be caused by data incoming from the outside, like messages that are received by a distributed algorithm (and are stored in the `receive` symbol 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 temporary variables. After all actions have been processed, these values are written back to the real memory. Therefore, 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 $t + 1$.

This approach allows for a greater amount of disarray in the rules, since the only possible positional dependencies left are those of rules that write to the same variables. 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 "learning" features like the bucket brigade algorithm[40] from the evolution, we additionally gain some form of 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) the target symbol. Assuming that an additional objective function which puts pressure into the direction of smaller programs is always imposed, superfluous appearances of 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 1.4.4 on page 26). The positional and cardinality independence are a clear example of phenotypic redundancy and neutrality in RBGP. They allow a useful rule to be replicated arbitrary often in the same program without decreasing its functional fitness. This is likely to happen during crossover, without changing any functionality. By doing so, the conditional parts of the rule will (obviously) be copied too. Subsequent mutation operations may now modify the action part of the rule and lead to improved behavior.

4.9.3 Complex Statements

From the above descriptions, it would seem that rule-based programs are strictly sequential, without branching or loops. This is generally not the case.

[40] The bucket brigade algorithm is discussed in Section 7.3.7 on page 219.

Complex Conditions

Assume that we have five variables `a...e` and want to express something like

```
1 if( (a<b) ∧ (c>d) ∧ (a<d) ) {
2   a += c;
3   c--;  }
```

Listing 4.8: A complex conditional statement.

we can do this in RBGP with 4 rules:

```
1 true ∧ true ⇒ e_{t+1} = 0
2 (a_t<b_t) ∧ (c_t>d_t) ⇒ e_{t+1} = 1
3 (a_t<d_t) ∧ (e_t=1) ⇒ a_{t+1} = a_t + c_t
4 (a_t<d_t) ∧ (e_t=1) ⇒ c_{t+1} = c_t - 1
```

Listing 4.9: The RBGP version of listing 4.8.

Although this does not look very elegant, it fulfils the task by storing the first part 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 of them write to temporary variables, the then-part of the condition (in lines 3 and 4) will be reached in the next round (if `a<d` holds too). Notice that the only positional dependency in listing 4.9 are that rule 2 must always occur after 1 – all rule permutations that obey this statement are equivalent and so is listing 4.10:

```
1 (a_t<d_t) ∧ (e_t=1) ⇒ a_{t+1} = a_t + c_t
2 true ∧ true ⇒ e_{t+1} = 0
3 (a_t<b_t) ∧ (c_t>d_t) ⇒ e_{t+1} = 1
4 (a_t<d_t) ∧ (e_t=1) ⇒ c_{t+1} = c_t - 1
5 true ∧ true ⇒ e_{t+1} = 0
6 (a_t<d_t) ∧ (e_t=1) ⇒ a_{t+1} = a_t + c_t
7 (a_t<b_t) ∧ (c_t>d_t) ⇒ e_{t+1} = 1
8 (a_t<d_t) ∧ (e_t=1) ⇒ c_{t+1} = c_t - 1
```

Listing 4.10: An equivalent alternative version of listing 4.9.

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 4.11: A loop.

The loop defined in listing 4.11 can be expressed in RBGP like outlined in listing 4.12, where we use the `start`-symbol (line 1) to initialize `a`. As its name suggests, the `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 (a_t>0) ∧ true ⇒ a_{t+1}=a_t-1
3 false ∨ (a_t>0) ⇒ b_{t+1} = b_t * a_t
```

Listing 4.12: The RBGP-version of listing 4.11.

Here no positional or cardinality restrictions occur at all, so listing 4.13 is equivalent to listing 4.12.

```
1 false ∨ (a_t>0) ⇒ b_{t+1} = b_t * a_t
2 (start_t>0) ∨ false ⇒ a_{t+1}=c_t
3 false ∨ (a_t>0) ⇒ b_{t+1} = b_t * a_t
4 (a_t>0) ∧ true ⇒ a_{t+1}=a_t-1
5 (start_t>0) ∨ false ⇒ a_{t+1}=c_t
```

Listing 4.13: An equivalent alternative version of listing 4.12.

4.10 Artificial Life and Artificial Chemistry Approaches

Definition 65 (Artificial Life). Artificial Life[41], also abbreviated with *AL-ife* or *AL*, is a field of research that studies the general properties of life by synthesizing and analyzing life-like behavior [711].

4.10.1 Push, PushGP, and Pushpop

Push is a stack-based programming language introduced by Spector especially suitable for evolutionary computation [712, 713, 714]. 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* [715, 716].

A Push program is either an instruction, a literal, or a sequence of zero or more Push programs inside parentheses.

```
1 program ::= instruction | literal | ( {program} )
```

Each 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 available, 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 [714, 715] is

[41] `http://en.wikipedia.org/wiki/Artificial_life` [accessed 2007-12-13]

```
1  ( 5 1.23 INTEGER.+ ( 4 ) INTEGER.- 5.67 FLOAT.* )
2  Which will leave the stacks in the following states:
3  FLOAT STACK   : (6.9741)
4  CODE STACK    : ( ( 5 1.23 INTEGER.+ ( 4 ) INTEGER.- 5.67
5                     FLOAT.* ) )
6  INTEGER STACK: (1)
```

Listing 4.14: 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 [712, 715]:

```
1  (CODE.QUOTE (2 3 INTEGER.+) CODE.DO)
```

Listing 4.15: 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 4.15 is just a complicated way to add $2 + 3 = 5$.

```
1  (CODE.QUOTE
2    (CODE.QUOTE (INTEGER.POP 1)
3     CODE.QUOTE (CODE.DUP INTEGER.DUP 1 INTEGER.- CODE.DO
         INTEGER.*)
4     INTEGER.DUP 2 INTEGER.< CODE.IF)
5  CODE.DO)
```

Listing 4.16: Another example for the usage of the CODE stack.

Listing 4.16 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 4.16 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 4.17: An example for the creation of procedures.

In listing 4.17, 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 4.18: .]An example for the creation of procedures similar to listing 4.17].

The EXEC stack is very similar to the CODE stack, except that its elements are pushed in the inverse order. The program in listing 4.18 is similar to listing 4.17 [715].

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 [712, 713, 562, 717, 718]. 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* [712, 713, 719]. 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

- `CODE.RAND` pushes newly created random code onto the `CODE` stack.
- `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.
- `ELDER` performs a tournament between n individuals of the previous generation and returns the winner.
- `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.

4.11 Evolving Algorithms

As already discussed, with Genetic Programming we breed algorithms and programs suitable for a given class of problems. In order to guide such an evolutionary process, we have to evaluate the utility of these grown programs. Algorithms are valuated 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 for example with its size and memory consumption. We specify (multiple) objective values to map these attributes to the space of the real numbers $\mathbb{R}$.

4.11.1 Restricting Problems

While some of the non-functional objective values can easily be computed (for example the algorithms size), at least determining its functional utility requires testing. The functional utility of the algorithm specifies how close its

results produced from given inputs come to the output values wanted. This problem cannot be solved by any computable function since it is an instance of the Entscheidungsproblem[42] [720] as well as of the halting problem[43] [721].

Entscheidungsproblem

The *Entscheidungsproblem* formulated by David Hilbert asks for an algorithm that, if provided with a description of a formal language and statement in that language, can decide whether the statement is `true`or `false`. In the case of Genetic Programming, the formal language is the language in which the algorithms are evolved and the statements are the algorithms itself. Church [722, 723] and Turing [724, 725] both proved that such an algorithm cannot exist.

Halting problem

The *Halting problem* asks for an algorithm that decides if another algorithm will finish at some time or runs forever if provided with a certain, finite input. Again, Turing [724, 725] proved that an algorithm solving the Halting Problem cannot exist. His proof is based on a counter-example. If we assume that a correct algorithm *doesHalt* exists (as presumed in Algorithm 4.2) which takes an algorithm as input and determines if the algorithm will halt or not. We could now specify the algorithm *trouble* which in turn uses *doesHalt* to determine if it itself will halt at some point of time. If *doesHalt* returns `true`, it loops forever. Otherwise it halts. *doesHalt* cannot return the correct result for *trouble* and hence it cannot be universally applied and thus it is not possible to solve the halting problem algorithmically.

Algorithm 4.2: Halting problem: reductio ad absurdum

```
1  doesHalt(algo) ∈ {true, false}
2  begin
3  |  ...
4  end
5  trouble()
6  begin
7  |  if doesHalt(trouble) then
8  |  |  while true do
9  |  |  |  ...
10 end
```

[42] `http://en.wikipedia.org/wiki/Entscheidungsproblem` [accessed 2007-07-03]
[43] `http://en.wikipedia.org/wiki/Halting_problem` [accessed 2007-07-03]

4.11.2 Why No Exhaustive Testing?

Since we can neither determine the evolved programs will terminate, let alone if they will provide correct results, we must use testing in order to determine whether they are suitable for a given problem.

Software testing is a very important field in software engineering [726, 727, 728, 729]. Its core problem is the size of the input space. Let us assume that we have, for example, a program that takes two integer numbers (32 bit) as input and computes another one as output. If we wanted to test this program for all possible inputs, we have to perform $2^{32} * 2^{32} = 2^{64} = 18'446'744'073'709'551'616$ single tests. Even if each test run would only take $1\mu s$, exhaustive testing would take approximately $584'542$ years. In most cases however the input space is much larger and in general, we can consider it to be countable infinite large according to Definition 92 on page 504.

Thus, we can only pick a very small fraction of the possible test scenarios and hope that they will provide significant results. The probability that this will happen depends very much on the method with which we select the test cases.

Here we can draw another interesting analogy to nature: One can never be sure whether evolved behavioral patterns (like algorithms) are perfect and free from error in all possible situations or not. Nature indeed has the same problem as the noble scientist trying to apply Genetic Programming.

What spontaneously comes to my mind here is a 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; it has served the monkeys well for generations. But with the occurrence of the hairless apes called human beings, the situation changed. African hunters use this behavior against the monkey by creating a situation that was never relevant during its evolutionary *testing* period. The slice a coconut in half and put a hole in one side just big enough for a monkey 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 a free meal, find the coconut with the hole, stick its hand inside, and grab the fruit. However, it cannot pull out the hand anymore with the orange in its fist. The hunter now can very easily catch the monkey, to whom it never occurs that it could let go the fruit and save its life.

What I here want to say is that 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.

From the impossibility of deciding the Halting problem, another important fact follows. If we compile the algorithms evolved to real machine code and execute them as a process, it is possible that this process will never terminate. We therefore either need a measure to restrict the runtime of the executed

programs which might be more or less difficult, depending on the operating system we use, or we could use simulations. The latter approach is more suitable. It is very easy to limit the runtime of a program that is executed on a simulator – even to an exact number of CPU ticks, which, in general, would not be possible with real processes. Furthermore, the simulation allows us a much deeper insight into the behavior of the algorithm and we may also emulate resources required like network connections, sensors, or physical phenomena like interference.

4.11.3 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.

Code Size

In section Section 37.1.1 on page 585 we give 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 66 (Code Size). The code size $c(A)$ of an algorithm or program A 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 $c(A) \in \mathbb{N}_0$. Since algorithms are statically finite per definition (see Definition 199 on page 588), the code size is always finite.

Code Bloat

Definition 67 (Bloat). In Genetic Programming, bloat is the uncontrolled growth in size of the individuals during the course of the evolution [730, 731, 539, 11, 732].

Code bloat is often used in conjunction with code *introns*, which are regions inside programs that do not contribute to the fitness because they can never be reached (see Definition 54 on page 123). 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. Programs become unnecessarily big – solutions for a problem should always be as small as possible.
2. Mutation and crossover operators always have to select the point in an individual where they will change it. If there exist many points in an individual that do not contribute to its functionality, the probability of

selection such a point for modification is high. The modified 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 and the breeding process of new solution candidates.
4. It leads to increased memory consumption.

There are many theories about how code bloat emerges [730], some of them are:

- Unnecessary code hitchhikes with good individuals. Since it is part of a fit solution candidate that creates many offspring, it is likely to be part of many new individuals. [733]
- 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 good solution candidate has good objective values, unnecessary code can be one measure for defense against crossover and mutation. If the genetic operators are neutralized, the offspring will have the same good fitness as its parent [731, 597, 734, 735, 736, 539, 737]. The reduction of the destructive effect of crossover on the fitness can also have positive effects [738, 739] since it may lead to a smoother evolution.
- Similar to the last theory, the idea of removal bias 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 an individual is finite, there also exists an upper limit of the amount of code that can be removed without altering the functionality of the individual. On the other hand, for the size of new sub-trees that could be inserted due to mutation or crossover, no such limit exists. Therefore, programs tend to grow [740, 737].
- According to the diffusion theory, the number of large programs in the solution space that are functional is higher than the number of small functional programs. Thus, code bloat would be the movement of the population into this direction [737].
- Another theory states that for each unreachable or dysfunctional code, there exists and invalidator. In the formula $4 + 0 * (4 - x)$ for example, 0 invalidates the whole part $0 * (4 - x)$. Luke [730] argues that if the proportion of such invalidators remains constant, the chance of their occurrence close to the root of a tree would be higher. Hence, their influence on large trees would be bigger than on small trees and with the growth of the programs the amount of unnecessary instructions would increase.
- Luke [730] furthermore defines a more general theory for the tree growth. As already stated, crossover is likely to destroy the functionality of an individual. On the other hand, the deeper the crossover occurs in the tree, the lesser is its influence since fewer instructions are removed. On the other hand, if only a few instructions are replaced from a functional program,

they are likely to be exchanged by a larger sub-tree. The new offspring, retaining its parent's functionality, therefore tends to be larger.
- Instead of being real solutions, programs that grow uncontrolled also tend to be some sort of decision tables. This phenomenon is called *overfitting* and is discussed in detail in Section 1.4.6 on page 27 and Section 19.1.3 on page 332. The problem is that overfit programs tend to have marvelous good fitness for the test cases/sample data, but are normally useless for any other input.

Runtime

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.

Memory Consumption

Like the complexity in time, the complexity in space of the bred solutions often should to be minimized. The count of variables and memory cells needed by program in order to perform its work should be as small as possible.

Errors

An example for an application where non-functional errors can occur should be minimized is symbolic regression. Here, the division operator *div* is often redefined in order to prevent division-by-zero errors. Therefore, such a division is either rendered to a *nop* (i. e. does nothing) or yields 1 as result. However, one could count the number of such errors and make the subject to minimization too.

Transmission Count

If we evolve distributed algorithms, we want the number of messages required to solve a problem to 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.

- One of the simplest means of doing so is to define additional objective functions and perform a multi-objective optimization. Successful and promising experiments showed that this is a viable countermeasure for code bloat [741], for instance.
- 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 [742].
- Poli [743] suggests that the fitness of a certain portion of the population with above-average code size is simple zeroed. These artificial *fitness holes* will repel the individuals from becoming too large and hence reduce the code bloat.

5

Evolution Strategy

5.1 Introduction

An evolution Strategy[1] (abbreviated with ES) is a heuristic optimization technique based in the ideas of adaptation and evolution, a special form of evolutionary algorithms [744, 470, 745, 746, 747, 199, 748, 1]. Evolution strategies have the following features:

- They usually use vectors of real numbers as individuals, i. e. a string genome[2] of fixed-size floating point number sequences.
- Mutation and selection are the primary operators and the recombination is less common.
- Mutation most often changes the elements x_i of the individual vector x to a number drawn from a normal distribution $N(x_i, \sigma_i^2)$. (see also Section 3.4.1 on page 124).
- Then, the values σ_i are governed by self-adaptation [749, 750] or by covariance matrix adaptation [751, 752, 753, 754].

In evolution strategy, these operations, as in any normal evolutionary algorithms, are performed iteratively in a cycle along with evaluation and reproduction processes where the iterations are called *generations*.

5.2 General Information

5.2.1 Areas Of Application

Some example areas of application of evolution strategy are:

Application	References

[1] `http://en.wikipedia.org/wiki/Evolution_strategy` [accessed 2007-07-03], `http://www.scholarpedia.org/article/Evolution_Strategies` [accessed 2007-07-03]

[2] see Section 3.4 on page 124

data mining and data analysis	[231]
scheduling problems	[755]
chemistry	[756, 757, 758]
to find emission sources from atmospheric observations	[759]
combinatorial optimization	[760]
geometry, surface reconstruction, CAD/CAM	[761, 762]
solving Travelling Salesman Problem (TSP)-like tasks	[763, 764]
optics and image processing	[765, 766]

For more information see also [397].

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 61

5.2.3 Books

Some books about (or including significant information about) evolution strategy are (ordered alphabetically):

Schwefel: *Evolution and Optimum Seeking: The Sixth Generation* (see [199])
Rechenberg: *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution* (see [470])
Rechenberg: *Evolutionsstrategie '94* (see [745])
Beyer: *The theory of evolution strategies* (see [748])
Quagliarella, Periaux, Poloni, and Winter: *Genetic Algorithms and Evolution Strategy in Engineering and Computer Science: Recent Advances and Industrial Applications* (see [397])

5.3 Populations in Evolutionary Strategy

The following classification has been partly borrowed from German Wikipedia site for evolution strategy[3].

[3] `http://de.wikipedia.org/wiki/Evolutionsstrategie` [accessed 2007-07-03]

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 8 on page 223.

5.3.2 $(\mu + 1)$-ES

Here, the population contains μ individuals from which one is drawn randomly. This individual is reproduced then reproduced and 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 individuals are created. From the joint set of offspring and parents, only the μ fittest ones are kept [767].

5.3.4 (μ, λ)-ES

Again, from μ parents $\lambda > \mu$ children are created. The parents are subsequently deleted and from the λ offspring individuals, only the μ fittest are retained [768].

5.3.5 $(\mu/\rho, \lambda)$-ES

Evolution strategies named $(\mu/\rho, \lambda)$ are (μ, λ) strategies. Here 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 as used in other evolutionary algorithms is applied, $\rho = 2$ holds. A special case of $(\mu/\rho, \lambda)$ algorithms is the $(\mu/\mu, \lambda)$ evolution strategy [769, 768].

5.3.6 $(\mu/\rho + \lambda)$-ES

Analogously to $(\mu/\rho, \lambda)$-ESs, the $(\mu/\rho + \lambda)$-ESs are (μ, λ) strategies where ρ denotes the number of parents of an offspring individual.

5.3.7 $(\mu', \lambda'(\mu, \lambda)^\gamma)$-ES

From a population of the size μ', λ' offspring is created and isolated for γ generations. 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 of the γ isolated populations is selected and the cycle starts again with λ' new child individuals. This nested evolutionary strategy can be more efficient than the approaches when applied to complex multimodal fitness environments [745, 770].

5.4 One-Fifth Rule

The $\frac{1}{5}$ success rule defined by Rechenberg denotes 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[4] (DE,DES) is a method for mathematical optimization of multidimensional functions that belongs to the group of evolution strategies [771, 772, 773, 774, 775, 101, 776]. First published in [777], the DE technique has been invented by Price and Storn in order to solve the Chebyshev polynomial fitting problem. It has proven to be a very reliable optimization strategy for many different tasks with parameters that can be encoded in real vectors like digital filter design, multiprocessor synthesis, neural network learning, optimization of alkylation reactions, and the design of water pumping systems or gas transmission networks.

The crucial idea behind differential evolution is the way the recombination operator is defined for creating trial parameter vectors. The difference $\mathbf{x}_1 - \mathbf{x}_2$ of two vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ in $\tilde{X}$ is weighted with a weight $w \in \mathbb{R}$ and added to a third vector $\mathbf{x}_3$ in the population.

$$\mathbf{x}_n = deRecombination(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) \Leftrightarrow \mathbf{x}_n = \mathbf{x}_3 + w(\mathbf{x}_1 - \mathbf{x}_2) \tag{5.1}$$

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.

5.5.2 General Information

Areas Of Application

Some example areas of application of differential evolution are:

Application	References
mechanical design optimization	[778, 779]

[4] `http://en.wikipedia.org/wiki/Differential_evolution` [accessed 2007-07-03], `http://www.icsi.berkeley.edu/~storn/code.html` [accessed 2007-07-03]

chemistry	[780, 781, 782, 783]
scheduling	[784]
function optimization	[785]
filter design	[786, 787]

Journals

Some journals that deal (at least partially) with differential evolution are (ordered alphabetically):

Journal of Heuristics (see Section 1.8.3 on page 43)

Books

Some books about (or including significant information about) differential evolution are (ordered alphabetically):

Price, Storn, Lampinen: *Differential Evolution – A Practical Approach to Global Optimization* (see [771])
Feoktistov: *Differential Evolution – In Search of Solutions* (see [772])
Corne, Dorigo, Glover: *New Ideas in Optimisation* (see [178])

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). 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. Hence, mutation and selection are the only operators used in EP and recombination is not applied. The selection scheme utilized in evolutionary programming is quite similar to the $(\mu + \lambda)$ method in evolution strategies.

Evolutionary programming was first invented by Lawrence Fogel [788, 19] in order to evolve finite state machines as predictors for data streams. His son David Fogel [789, 790, 791] and he together [792, 793] are also the major forces developing it further.

Generally, it is hard to distinguish between EP and genetic programming, genetic algorithms, and evolution strategy. Although there are semantic differences (as already mentioned), the author thinks that the evolutionary programming approach generally has melted together with these other research areas.

6.2 General Information

6.2.1 Areas Of Application

Some example areas of application of evolutionary programming are:

[1] `http://en.wikipedia.org/wiki/Evolutionary_programming` [accessed 2007-07-03]

Application	References
machine learning	[789]
evolution of finite state machines	[788]
evolving game players	[790, 791]
training of artificial neural networks	[793]
chemistry and biochemistry	[794, 795, 796]
electronic and controller design	[797, 798]
fuzzy clustering	[799]
general constraint optimization	[800]
robotic motion control	[801]

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 60
History: 1998: San Diego, California, USA, see [802]
1997: Indianapolis, Indiana, USA, see [803]
1996: San Diego, California, USA, see [804]
1995: San Diego, California, USA, see [805]
1994: see [806]
1993: see [807]
1992: see [808]

EUROGEN: Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems
see Section 2.2.2 on page 61

6.2.3 Books

Some books about (or including significant information about) evolutionary programming are (ordered alphabetically):

Fogel, Owens, and Walsh: *Artificial Intelligence through Simulated Evolution* (see [788])

Fogel, Owens, and Walsh: *System Identification through Simulated Evolution: A Machine Learning Approach to Modeling* (see [789])

Fogel: *Blondie24: playing at the edge of AI* (see [791])

7

Learning Classifier Systems

7.1 Introduction

In the late 1970s, Holland invented and patented a new class of cognitive systems, called classifier systems (CS) [705, 706]. These systems are a special case of production systems [809, 810] and consist of four major parts:

1. a set of interacting productions, called *classifiers*,
2. a performance algorithm that directs the action of the system in the environment,
3. a simple learning algorithm that keeps track on each classifier's success in bringing about rewards, and
4. a more complex algorithm, called the *genetic algorithm*, that modifies the set of classifiers so that variants of good classifiers persist and new, potentially better ones are created in a provably efficient manner [811].

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) [1, 812]. Today, learning classifiers are sometimes subsumed under a new machine learning paradigm called evolutionary reinforcement learning (ERL) [1, 813].

7.2 General Information

7.2.1 Areas Of Application

Some example areas of application of learning classifier systems are:

Application	References

[1] `http://en.wikipedia.org/wiki/Learning_classifier_system` [accessed 2007-07-03]

data mining	[814, 815, 816]
inferring the grammar of natural language	[817, 818, 819]
decease risk-prediction in medicine	[820]
image processing	[821, 822]

7.2.2 Conferences, Workshops, etc.

Some conferences, workshops and such and such on learning classifier systems are:

IWLCS: International Workshop on Learning Classifier Systems
Nowadays often co-located with GECCO (see Section 2.2.2 on page 62).
History: 2007: London, England, see [823]
2006: Seattle, WA, USA, see [824]
2005: Washington DC, USA, see [825, 826]
2004: Seattle, Washington, USA, see [827, 826]
2003: Chicago, IL, USA, see [828, 826]
2002: Granada, Spain, see [829]
2001: San Francisco, CA, USA, see [830]
2000: Paris, France, see [831]
1999: Orlando, Florida, USA, see [832]
1992: Houston, Texas, USA, see [833]

7.3 The Basic Idea of Learning Classifier Systems

Figure 7.1 illustrates Holland's original idea of 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 binary messages that are written into a message list *(d)*. On this list, the classifiers, simple `if-then` rules *(e)*, 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 [834]. 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 [835].

Such a classifier system is computational complete and is hence as powerful as any other Turing-equivalent programming language [836, 837]. One can imagine it as something like a computer program where the rules play the role of the instructions and the messages are the memory.

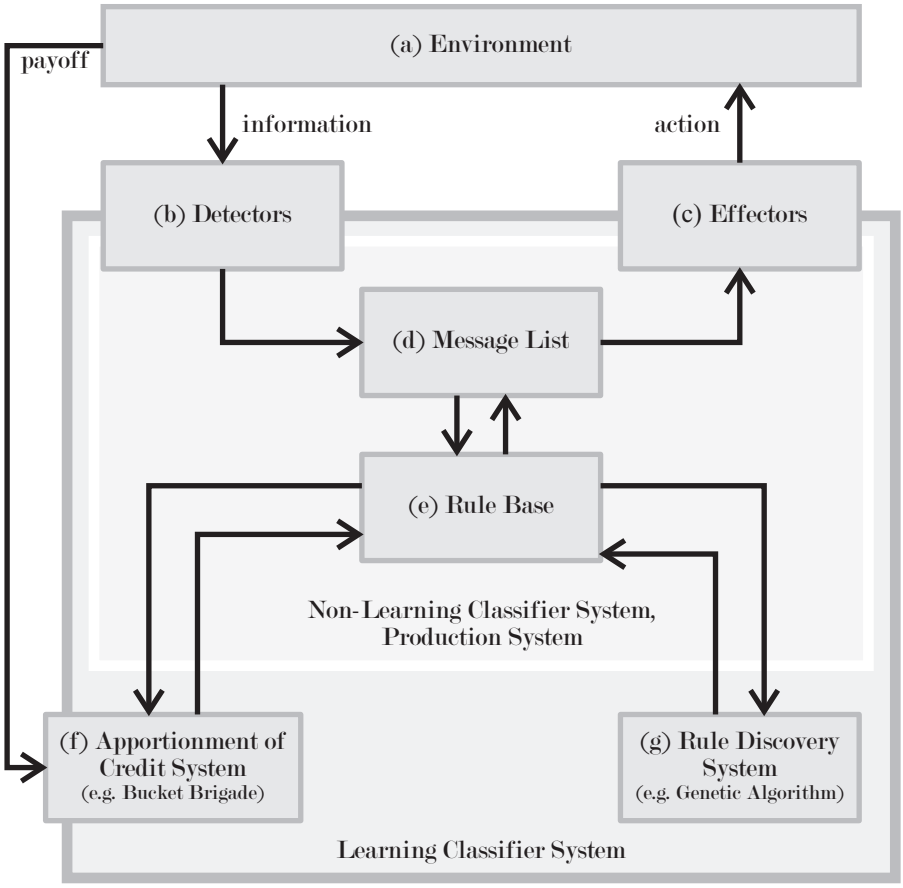


Fig. 7.1: The structure of a Michigan style learning classifier system.

7.3.1 Messages

In order to describe how rules and messages are structured in a basic classifier systems, we borrow a simple example from [1] and modify it so it fits to our needs.

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, and therefore it can sense a small, flying object. These objects can be eaten if they are right in front of it. They can be distinguish if they have stripes or not, because small, flying objects with stripes probably are bees or wasps which should preferably not be eaten. The frog furthermore also senses large, looming objects far above, birds in human-lingo, which should be avoided by jumping away quickly. The frog furthermore has a sense of direction – it detects if the

objects are in front, to the left, or to the right of it and it can then turn into this direction. Now we can compile this behavior into the form of `if-then` rules which are listed in Table 7.1.

Table 7.1: `if-then` rules for frogs

No.	`if`-part	`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

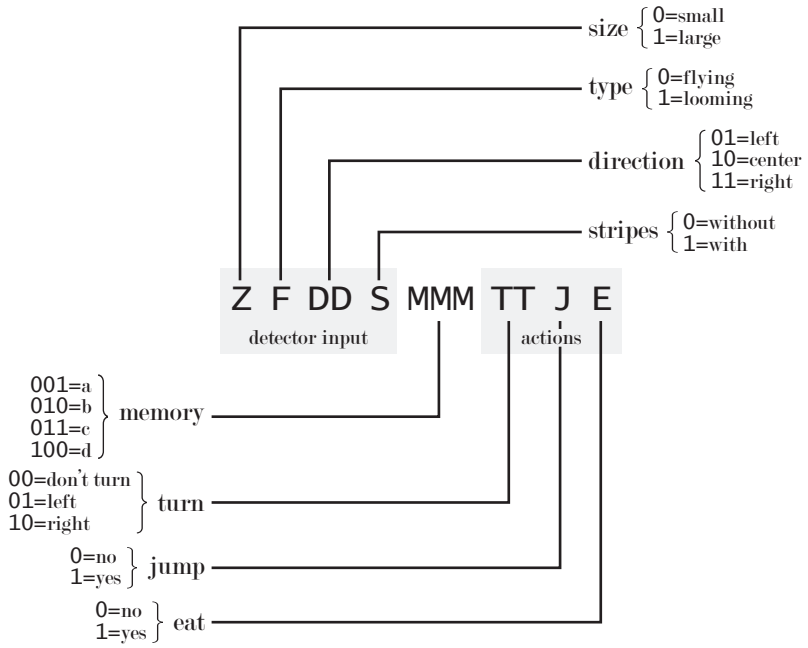


Fig. 7.2: One possible encoding of messages for a frog classifier system

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 are encoded in one message type. Also, three bits are assigned for encoding the internal messages a to d. Two bits would not suffice, since `00` would also occur in all input messages. The idea is that at the beginning of a classification process, the input messages are written to the message list. They contain useful values only at the positions reserved for detections and zeros everywhere else. They will be transformed 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. $|m| = 12 \; \forall$ *message* m.

7.3.2 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 `*`. In a condition,

- a `0` means that the corresponding bit in the message must be `0`,
- a `1` means that the corresponding bit in the message must be `1`, and
- a `*` means *do not care*, i. e. the corresponding bit in the message may be `0` as well as `1`

for the condition to match.

Definition 68 (*matchesCondition*). A message m matches to a condition c if $matchesCondition(m, c)$ evaluates to `true`.

$$matchesCondition(m, c) = \; \forall\, 0 \leq i < |m| \Rightarrow m[i] = c[i] \vee c[i] = * \qquad (7.1)$$

The condition part of a rule may consist of more than one condition which are then implicitly concatenated with and ($\wedge$). A classifier is satisfied if all its conditions are satisfied by *some* message in the current message list. This means that each of the conditions of a classifier may match to a different message. Furthermore, we can precede each single condition c with an additional ternary digit which defines if it should be negated or not: `*` denotes $\neg c$ and `0` as well as `1` denotes c[2]. A negated condition evaluates to `true` if no message exists that matches it. By combining $\wedge$ and $\neg$ we get `nand`s with which we can build all other logic operations and hence, whole computers [838]. Algorithm 7.1 illustrates how the condition part C is matched against the message list M.

Definition 69 (Condition Specificity). We define the condition specificity $conditionSpecificity(x)$ of a classifier x to be the number of non-`*` symbols in its condition part $C(x)$.

$$conditionSpecificity(x) = |\{\forall; i : C(x)[i] \neq *\}| \qquad (7.2)$$

[2] Here we deviate from one of our sources (namely [835]) because the definition of the *conditionSpecificity* (see Definition 69) makes more sense this way.

Algorithm 7.1: $\{\texttt{true}, \texttt{false}\} = matchesConditions(C, M)$

Input: C the condition part of a rule
Input: M the message list
Input: Implicit: k the length of the messages $m \in M$ and the single conditions $c \in C$
Input: Implicit: $hasNegPart$ `true` if and only if the single conditions have a negation prefix, `false` otherwise
Data: $x_{new} \in \tilde{X}$ the new element created
Data: X_{tabu} the tabu list
Output: `true` if messages exist that match the condition part C, `false` otherwise

```
1  begin
2      i ⟵ 0
3      match ⟵ true
4      while (i < |C|) ∧ match do
5          if hasNegPart then
6              neg ⟵ C[i] = *
7              i ⟵ i + 1
8          else neg ⟵ false
9          c ⟵ subList(C, i, k)
10         i ⟵ i + k
11         match ⟵ neg xor (∃ m ∈ M : matchesCondition(m, c))
12     return match
13 end
```

A 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 $conditionSpecificity(x_1) > conditionSpecificity(x_2)$ is more general than the rule x_1. We can use this information for example if two rules match to one message, and we want only one to post a message. By always selecting the more specific rule in such situations, we built a *default hierarchy* [839] which allows specialized classes to be contained in more general classes.

7.3.3 Actions

The action part of a rule most often has exactly the length of a message. It can either be binary or ternary encoded. 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 one 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 70 (*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 implicit access to the first 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 ($m = createList(|a|, 0)$). 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.

$$\begin{aligned} n = mergeAction(a) \Leftrightarrow\ & (|n| = |a| \wedge) \\ & (n[i] = a[i]\ \forall\ i : a[i] \neq *) \wedge \\ & (n[i] = m[i]\ \forall\ i : a[i] = *) \end{aligned} \tag{7.3}$$

7.3.4 Classifiers

So we now know that a rule x consist of a condition part C and an action part a. C is 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 go back to our frog example. Based on the encoding scheme defined in Figure 7.2, we can now translate Table 7.1 into a set of classifiers. In our example we use two conditions c_1 and c_2 with negation symbols n_1 and n_2, i. e. $r = 2$.

Table 7.2: `if-then` rules for frogs in encoded form

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 contains the result of this encoding. We now can apply this classifier to a situation in the life of our frog: 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. $m_1 =$ (`0 0 01 0 000 00 0 0`) for the fly,
2. $m_2 =$ (`0 0 11 1 000 00 0 0`) for the bee, and
3. $m_3 =$ (`1 1 01 0 000 00 0 0`) 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. $m_4 =$ (`0 0 00 0 001 00 0 0`) from rule 1 and
2. $m_5 =$ (`0 0 00 0 100 00 0 0`) 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 $m_6 =$ (`0 0 00 0 000 00 1 0`) 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.

7.3.5 Non-Learning Classifier Systems

So far we have described a non-learning classifier system. Algorithm 7.2 describes 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. Then we can use genetic algorithms to evolve them, for example.

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.

7.3.6 Learning Classifier Systems

In order to convert this non-learning classifier system to learning classifier system as proposed by Holland [840] and defined in Algorithm 7.3, we have to add the aforementioned missing components. [1] suggests 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 $\mathfrak{s}(x)$, the 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* [841, 842].

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: $I \in \mathbb{N}$ the maximum number of iterations for the internal loop, avoids endless loops
Input: Implicit: $a(x)$, $C(x)$ functions that extract the action and the condition part from a rule x
Data: $i \in \mathbb{N}$ a counter the internal loop
Data: M_i the message list of the i^{th} iteration
Data: S the set of satisfied classifiers x

```
1  begin
2      while true do
3          M_1 ⟵ readDetectors()
4          i ⟵ 1
5          repeat
6              S ⟵ {x ∈ P : matchesConditions(C(x), M_i)}
7              M_{i+1} ⟵ ∅
8              foreach x ∈ S do
9                  M_{i+1} ⟵ M_{i+1} ∪ {mergeAction(a(x))}
10             i ⟵ i + 1
11         until (S = ∅) ∨ (i > I)
12         sendEffectors(M_i)
13 end
```

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 that 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, turning the LCS into an ordinary classifier system.

7.3.7 The Bucket Brigade Algorithm

The bucket brigade algorithm [843, 844, 706] is one method of solving the credit assignment problem in learning classifier systems.

It selects the classifiers from the match set S that are allowed to post a message (i. e. becoming member in the activated set S') by an auction. Therefore, each matching classifier $x \in S$ places a bid $B(x)$ which is the product of a linear function ϑ of the condition specificity of x, a constant

Algorithm 7.3: $learningClassifierSystem(B)$

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 S that are allowed to trigger their actions
Input: Implicit: $I \in \mathbb{N}$ the maximum number of iterations for the internal loop, avoids endless loops
Input: Implicit: $a(x)$, $C(x)$ functions that extract the action and the condition part from a rule x
Input: Implicit: $generationCriterion$ a criterion that becomes `true`if new classifiers should be created
Input: Implicit: $createNewRules$ a function that finds new rules
Data: P the population of rules x_i that determine the behavior of the classifier system
Data: $i \in \mathbb{N}$ a counter the internal loop
Data: M_i the message list of the i^{th} iteration
Data: S the set of satisfied classifiers x
Data: S' the set of classifiers x selected from S
Data: R a variable to receive the credit/reward

```
1  begin
2      P ⟵ generateClassifiers()
3      s ⟵ 1 ∀ x ∈ P
4      while true do
5          M_1 ⟵ readDetectors()
6          i ⟵ 1
7          repeat
8              S ⟵ {x ∈ P : matchesConditions(C(x), M_i)}
9              S' ⟵ selectMatchingClassifiers(S)
               // ...update the strengths ...
10             M_{i+1} ⟵ ∅
11             foreach x ∈ S' do
12                 M_{i+1} ⟵ M_{i+1} ∪ {mergeAction(a(x))}
13             i ⟵ i + 1
14         until (S = ∅) ∨ (i > I)
15         sendEffectors(M_i)
           // ...receive and distribute the payoffs ...
16         if generationCriterion() then P ⟵ createNewRules(P)
17 end
```

$0 < \beta \leq 1$ that determines the fraction of the strength of x should be used and the strength $\mathfrak{s}(x)$ of x itself. In practical applications, values like $\frac{1}{8}$ or $\frac{1}{16}$ are often chosen for β.

$$B(x) = \vartheta(x) * \beta * \mathfrak{s}(x) + random_n(0, \sigma^2) \tag{7.4}$$

Sometimes a normal distributed random number is added to each bid in order to make the decisions of the system less deterministic.

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 [834].

$$\vartheta(x) = \frac{conditionSpecificity(x)}{|C(x)|} + \alpha \tag{7.5}$$

The bucket brigade version of the $selectMatchingClassifiers$-method 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 that 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.

$$P(x) = \beta * \mathfrak{s}(x) \tag{7.6}$$

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 [834].

$$T(x) = \tau * \mathfrak{s}(x) \tag{7.7}$$

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 $\mathfrak{s}(x)_{t+1}$ of a classifier x is composed of its old strength, its payment $P(x)$, the life-tax $T(x)$, and the reward $R(x)$.

$$\mathfrak{s}(x)_{t+1} = \mathfrak{s}(x)_t - P(x)_t - T(x)_t + R(x)_t \tag{7.8}$$

7.3.8 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 117) replaces old rules by new ones. In learning classifiers we apply steady-state genetic algorithms which are discussed in Section 2.1.3 on page 54. They will retain most of the classifier population and only replace the weakest rules. Therefore, the strength $\mathfrak{s}(x)$ of a rule x is used as its fitness $\mathfrak{f}(x)$.

$$\mathfrak{f}(x) = \mathfrak{s}(x) \tag{7.9}$$

For mutation and crossover, the well known reproduction operations for fixed-length string chromosomes discussed in Section 3.4.1 on page 124 are employed.

7.4 Families of Learning Classifier Systems

The exact definition of learning classifier systems still seems contentious [845, 846, 847, 848]. There exist for example versions without message list where the action part of the rules does not encode messages but direct output signals.

Also the importance of the role of genetic algorithms in conjunction with the reinforcement learning component is not quite clear. There are scientists who emphasize more the role of the learning components [849] and others who grant the genetic algorithms a higher weight [850, 705].

Depending on how the genetic algorithm acts, we can divide learning classifier systems into two types. The Pitt approach is originated at the University of Pittsburgh and mainly associated with Smith and De Jong [707, 708, 851, 852]. Pittsburgh-style learning classifier systems work on a population of separate rule sets, single classifiers, which are combined and reproduced by the genetic algorithm [815, 853, 854]. Closer to the original idea of Holland are Michigan-style LCSs where the whole population itself is considered as classifier. They focus on selecting the best rules in this rule set [855, 834, 856]. Wilson developed two subtypes of Michigan-style LCS:

1. ZCS systems use fitness sharing [857, 858, 859, 860] for reinforcement learning.
2. They have later been somewhat superseded by XCS systems which are accuracy-based [861, 862, 863, 864, 865].

8

Hill Climbing

8.1 Introduction

Hill climbing[1] (HC) [866] is a very simple search/optimization algorithm. In principle, it is a loop in which the currently known best solution $x^\star$ candidate is used to produce one offspring x_{new}. If this new individual is better than its parent, it replaces it. Then the cycle starts all over again, as specified in Algorithm 8.1.

The major problem of hill climbing is that it very easily gets stuck on a local optimum. It *always* uses the best known individual $x^\star$ to produce new solution candidates and the therefore used *mutate* operation will return an element $x_{new} \in \tilde{X}$ neighboring $x^\star$. Section 8.4 on the next page discusses how this problem can be overcome.

It should also be noted that hill climbing can be implemented in a deterministic manner if the neighbor sets are always finite and can be iterated over.

Algorithm 8.1: $x^\star = hillClimbing(f)$

Input: f the objective function to be minimized
Data: $x_{new} \in \tilde{X}$ the new element created
Output: $x^\star \in \tilde{X}$ the best element found

```
1 begin
2 |   x* ⟵ create()
3 |   while ¬terminationCriterion() do
4 |   |   x_new ⟵ mutate(x*)
5 |   |   if f(x_new) < f(x*) then x* ⟵ x_new
6 |   return x*
7 end
```

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

8.2 General Information

8.2.1 Areas Of Application

Some example areas of application of hill climbing are:

Application	References
application server configuration	[867]
development of robotic behavior	[868]
feature selection	[869]
solving games like matermind	[870]
to solve combinatorial problems like Knapsack	[871]
finding planted bisections	[872]

8.3 Multi-Objective Hill Climbing

As illustrated in Algorithm 8.2 on the facing page, we can easily extend the hill-climbing algorithm with a support for multi-objective optimization. Additionally, this new hill climber returns set of best known solutions instead of a single individual as done in Algorithm 8.1 on the previous page. This algorithm needs a selection scheme in order to determine which of the possible multiple currently best solution candidates should be used as parent for the next offspring. The selection algorithm applied must not rely on prevalence comparison directly, since no element in $X^\star$ prevails any other per definition – otherwise, the set of optimal elements $X^\star$ would contain non-optimal solution candidates. Selection should thus either be randomized (seeSection 2.4.2 on page 80) or a density/diversity/crowding-based fitness assignment process.

8.4 Problems in Hill Climbing

Both versions of the algorithm are 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 should be regarded as 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:

- A tabu-list can be used which stores elements recently evaluated. By denying visiting them again, a better exploration of the problem space can be enforced. This technique is used in tabu search, see Chapter 11 on page 237.

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

Algorithm 8.2: $X^\star = hillClimbing(c_F)$

Input: c_F the comparator function which allows us to compare the fitness of two solution candidates, used by *updateOptimalSet*
Data: $x_{new} \in \tilde{X}$ the new element created
Output: $X^\star \subseteq \tilde{X}$ the set of the best elements found

```
1  begin
2    X* ⟵ ∅
3    x_new ⟵ create()
4    while ¬terminationCriterion() do
5      X* ⟵ updateOptimalSet(X*, x_new)
6      X* ⟵ pruneOptimalSet(X*)
7      x_new ⟵ select(X*, 1)[0]
8      x_new ⟵ mutate(x_new)
9    return X*
10 end
```

- Not always use the better solution candidate to continue. Simulated annealing introduces a heuristic based on the physical model the cooling period of molten metal to decide whether a superior offspring should replace its parent or not. It is discussed Chapter 10 on page 231.
- The dynamic hill climbing approach [873] uses the last two visited points to compute unit vectors. With this technique, the directions are adjusted according to the structure of the space and a new coordinate frame is created that points more likely into the right direction.
- Randomly restart the search after so-and-so many steps, see Section 8.5.
- Use a reproduction scheme that not necessarily generates solution candidates directly neighboring $x^\star$, as done in random optimization introduced in Chapter 9 on page 227.

8.5 Hill Climbing with Random Restarts

Another approach is to simple restart the algorithm after some time at a random state. Of course, the best solutions found are remembered in a set $X^\star$ but are no longer incorporated in the search. An independent best-of-run set, $X^\star cur$ is used to create new solution candidates. Algorithm 8.3 on the following page illustrates this idea.

This method is also called *Stochastic Hill Climbing* (SH) or *Stochastic gradient descent*[3] [874, 875].

For the sake of generality, we define a function *shouldRestart*() that is evaluated in every iteration and determines whether or not the algorithm

[3] `http://en.wikipedia.org/wiki/Stochastic_gradient_descent` [accessed 2007-07-03]

should be restarted. *shouldRestart* therefore could for example count the iterations performed or check if any improvement was produced in the last ten iterations.

Algorithm 8.3: $X^\star = hillClimbing(c_F)$ (random restarts)

Input: c_F the comparator function which allows us to compare the fitness of two solution candidates, used by *updateOptimalSet*
Data: $X^\star_{cur} \subseteq \tilde{X}$ the set of the best elements found in the current run
Data: $x_{new} \in \tilde{X}$ the new element created
Output: $X^\star \subseteq \tilde{X}$ the set of the best elements found

```
1  begin
2      X* ⟵ {create()}
3      X*_cur ⟵ X*
4      while ¬terminationCriterion() do
5          if shouldRestart() then X*_cur ⟵ {create()}
6          else
7              x_new ⟵ select(X*_cur, 1)[0]
8              x_new ⟵ mutate(x_new)
9              X*_cur ⟵ updateOptimalSet(X*_cur, x_new)
10             X*_cur ⟵ pruneOptimalSet(X*_cur)
11             X* ⟵ updateOptimalSet(X*, x_new)
12             X* ⟵ pruneOptimalSet(X*)
13     return X*
14 end
```

9

Random Optimization

9.1 Introduction

Random optimization[1] [876] is a global optimization method *algorithmically* very close to hill climbing (see Chapter 8 on page 223). There are, however, three important differences between the two approaches:

1. In hill climbing, the new solution candidates that are created from a good individual $x^\star$ are always very close neighbors of it. In random search, that is not *necessarily* but only *probably*.
2. Random optimization is a purely numerical approach, i. e. most often $\tilde{X} \subseteq \mathbb{R}^n$.
3. In random optimization, we distinguish between objective values and constraints.

Like for simulated annealing, it has been proven that with random optimization the global optimum can be found with probability one. Furthermore, this premise even holds if the objective function is multimodal and even if its differentiability is not ensured [877].

The reason for this property and the first two differences to hill climbing lies in the special $mutate_{ro}$ procedure applied:

$$mutate_{ro}(\mathbf{x}^\star) = \mathbf{x}_{new} = \mathbf{x}^\star + \mathbf{r}, \quad \mathbf{x}, \mathbf{r} \in \mathbb{R}^n \tag{9.1}$$

$$\mathbf{r} = \begin{pmatrix} random_n(\mu_1, \sigma_1^2) & (\sim N(\mu_1, \sigma_1^2) \\ random_n(\mu_2, \sigma_2^2) & (\sim N(\mu_2, \sigma_2^2) \\ \dots \\ random_n(\mu_n, \sigma_n^2) & (\sim N(\mu_n, \sigma_n^2) \end{pmatrix} \tag{9.2}$$

To the (currently best solution candidate) vector $\mathbf{x}^\star$ we add a vector of normally distributed random numbers $\mathbf{r}$. The μ_i are the expected values and the σ_i the standard deviations of the normal distributions, as introduced

[1] `http://en.wikipedia.org/wiki/Random_optimization` [accessed 2007-07-03]

in Section 35.4.2 on page 537. The μ_i define a general direction for the search, i. e. if $\mu_i > 0$, $random_n(\mu_i, \sigma_i)$ will likely also be greater zero and for $\mu_i < 0$ it will probably also be smaller than zero (given that $|\sigma_i| \ll |mu_i|$). 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 chose 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.

The normal distribution is generally unbounded, which means that even for $\mu_i = 0$ and $\sigma_i \approx 0$, it is generally possible that the random elements in $\mathbf{r}$ can become very large. Therefore, local optima can be left again even with bad settings of μ and σ.

In order to respect the idea of constraint satisfaction in random optimization, we define the additional function $valid(x)$ that returns `true`if and only if a solution candidate x does not violate any of the constraints and requirements and `false`otherwise.

Algorithm 9.1 illustrates how random optimization works, clearly showing connatural traits in comparison with the hill climbing Algorithm 8.1 on page 223.

Algorithm 9.1: $x^\star = randomOptimization(f)$

Input: f the objective function to be minimized
Data: $x_{new} \in \tilde{X}$ the new element created
Output: $x^\star \in \tilde{X}$ the best element found

```
1  begin
2      repeat
3          x* ⟵ create()
4      until valid(x_new)
5      while ¬terminationCriterion() do
6          repeat
7              x_new ⟵ mutate_ro(x*)
8          until valid(x_new)
9          if f(x_new) < f(x*) then x* ⟵ x_new
10     return x*
11 end
```

Setting the values of μ and σ adaptively can lead to large improvements in convergence speed. The heuristic random optimization (HRO) algorithm [878] and its successor [879] random optimizer II for example update them by utilizing gradient information or reinforcement learning.

9.2 General Information

9.2.1 Areas Of Application

Some example areas of application of (heuristic) random optimization are:

Application	References
medicine	[880, 881]
biotechnology and bioengineering	[882]
training of artificial neural networks	[883]
global optimization of mathematical functions	[878]

10

Simulated Annealing

10.1 Introduction

Simulated Annealing[1] (SA), [884] is a global optimization algorithm inspired by the manner in which metals crystallize in the process of annealing or in which liquids freeze. Annealing steel is cooled down slowly in order to keep the system of the melt in a thermodynamic equilibrium which will increase the size of its crystals and reduce their defects. As cooling proceeds, the atoms of the steel become more ordered. If the cooling was prolonged beyond normal, the system would approach a "frozen" ground state at $T = 0K$ – the lowest energy state possible. 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 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 measuren in Kelvin, and k_B is the Boltzmann's constant[2] $k_B = 1.380650524 * 10^{-23} \frac{J}{K}$.

Simulated annealing was first introduced by Metropolis et al. [885]. The original method 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 in each iteration. The energy at the new geometry is computed and the energetic difference between the current and the new geometry is given as ΔE. The probability that this new geometry is accepted, $P(\Delta E)$ is defined as:

$$\Delta E = E(pos_{i+1}) - E(pos_i) \tag{10.1}$$

[1] `http://en.wikipedia.org/wiki/Simulated_annealing` [accessed 2007-07-03]
[2] `http://en.wikipedia.org/wiki/Boltzmann%27s_constant` [accessed 2007-07-03]

$$P(\Delta E) = \begin{cases} e^{-\frac{\Delta E}{k_B T}} & : \text{if } \Delta E > 0 \\ 1 & : \text{if } \Delta E \leq 0 \end{cases} \tag{10.2}$$

Thus, if the new nearby geometry has a lower energy level, the transition is accepted. Otherwise, the step will only be accepted if a uniformly distributed random number $r = random_u() \in [0, 1)$ generated is less or equal the Boltzmann probability factor $r \leq P(\Delta E)$. At high temperatures T, this factor is very near to 1, leading to the acceptance of many uphill steps. As the temperature falls, the proportion of steps accepted which increase the energy level diminishes. Now the system cannot escape local regions any more and (hopefully) comes to a rest in the global minimum at temperature $T = 0K$. The use of this stochastic technique turns Simulated Annealing into a Monte Carlo algorithm and is the major difference to Hill Climbing (see Chapter 8).

The abstraction of this method in order to allow arbitrary problem spaces is straightforward - the energy computation $E(pos_i)$ is replaced by a more general objective function $f(x \in \tilde{X})$. Algorithm 10.1 represents the basic simulated annealing process.

Algorithm 10.1: $x^\star = simulatedAnnealing(f)$

Input: f the objective function to be minimized
Data: $x_{new} \in \tilde{X}$ the new element created
Data: $x_{cur} \in \tilde{X}$ the element currently investigated
Data: $T \in \mathbb{R}^+$ the temperature of the system which is decreased over time
Data: $t \in \mathfrak{N}_0$ the current time index
Data: $\Delta E \in \mathbb{R}$ the enery difference of the x_{new} and x_{cur}.
Output: $x^\star \in \tilde{X}$ the best element found

```
1  begin
2      x_new ⟵ create()
3      x_cur ⟵ x_new
4      x* ⟵ x_new
5      t ⟵ 0
6      while ¬terminationCriterion() do
7          ΔE ⟵ f(x_new) − f(x_cur)
8          if ΔE ≤ 0 then
9              x_cur ⟵ x_new
10             if f(x_cur) < f(x*) then x* ⟵ x_cur
11         else
12             T ⟵ getTemperature(t)
13             if random_u() < e^(−ΔE/(k_B T)) then x_cur ⟵ x_new
14         x_new ⟵ mutate(x_cur)
15         t ⟵ t + 1
16     return x*
17 end
```

Simulated algorithms have an associated proof of asymptotic convergence which means that they will actually find the global optimum. Such algorithms are usually very slow [886]. Therefore, the process is often speeded up so a good solution is found faster while voiding the guaranteed convergence on the other hand. Such algorithms are called *Simulated Quenching* (SQ).

10.2 General Information

10.2.1 Areas Of Application

Some example areas of application of simulated annealing are:

Application	References
traveling salesman problem (TSP)	[887, 884, 888]
global optimization of mathematical functions	[889]
chemistry	[890, 891]
positron emission tomography/image reconstruction	[892, 893, 894, 888]
finance and trading	[895, 896]
circuit design	[897, 888]
path planning	[898, 899, 888]
planar mechanism synthesis	[900, 901, 888]
paper cutting waste optimization	[902]
seismic waveform inversion	[903]

For more information see also [904, 905].

10.3 Temperature Scheduling

Definition 71 (Temperature Schedule). The temperature schedule represents the sinking temperature in simulated annealing and it is accessed by the function $getTemperature(t)$ which returns the temperature to be used for the current iteration step t. The start temperature is T_{start} which will reduced by all subsequent iterations until it reaches $0K$ in the final step.

$$T = getTemperature(t) \in \mathbb{R}^+ \tag{10.3}$$

$$T_{start} = getTemperature(0) \tag{10.4}$$

$$0K = getTemperature(\infty) \tag{10.5}$$

Although there exists a wide range of methods to determine this temperature schedule – Miki et al. for example used genetic algorithms for this [906] – we will introduce only the three simple variants here given in [907].

- Reduce T to $(1-\epsilon)T$ after every m iterations, where ϵ and m are determined by experiment.
- Grant a total of K iterations, and reduce T after every m steps to a value $T = T_{start}(1-\frac{t}{K})\alpha$. t is the already introduced iteration index, 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.
- 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. Since ΔE_c may be 0, we allow a maximum temperature change of $T * \gamma, \gamma \in [0,1]$.

Another technique that may be applied are the random restarts already known from hill climbing (see page 225).

10.4 Multi-Objective Simulated Annealing

Again we want to incorporate this ability together with multi-objectivity and also enable the resulting algorithm to return a set of optimal solutions. Like for multi-objective hill climbing, this algorithm needs a selection scheme. The selection algorithm applied here should not rely on prevalence comparison directly, since no element in $X^\star$ prevails any other. Selection should thus either be randomized (Section 2.4.2 on page 80), employ a density/diversity/crowding-based fitness assignment process be strictly elitist like PSEA-II (Section 2.4.12 on page 94). If a fitness assignment process (Section 2.3 on page 65) is applied, one could also use the scalar fitness instead of the prevalence comparator in line 18, replacing it with $\Delta E \longleftarrow \mathfrak{f}(x_{new}) - \mathfrak{f}(x_{cur})$.

Algorithm 10.2: $X^\star = simulatedAnnealing(c_F)$

Input: c_F the comparator function which allows us to compare the fitness of two solution candidates, used by *updateOptimalSet*
Data: $X^\star_{cur} \subseteq \tilde{X}$ the optimal set of the current run
Data: $x_{cur} \in \tilde{X}$ the element currently investigated
Data: $x_{new} \in \tilde{X}$ the new element created
Data: $X^\star_{new} \subseteq \tilde{X}$ the optimal set of the current run after updating it with x_new
Data: $T \in \mathbb{R}^+$ the temperature of the system which is decreased over time
Data: $t \in \mathfrak{N}_0$ the current time index
Data: $\Delta E \in \mathbb{R}$ the enery difference of the x_{new} and x_{cur}.
Output: $X^\star \subseteq \tilde{X}$ the best element found

```
1  begin
2      X*_cur ⟵ {create()}
3      X* ⟵ X_cur
4      t ⟵ 0
5      while ¬terminationCriterion() do
6          if shouldRestart() then
7              X*_cur ⟵ {create()}
8              t ⟵ 0
9          else
10             x_cur ⟵ select(X*_cur, 1)[0]
11             x_new ⟵ mutate(x_cur)
12             X*_new ⟵ updateOptimalSet(X*_cur, x_new)
13             if X*_new ≠ X*_cur then
14                 X*_cur ⟵ pruneOptimalSet(X*_new)
15                 X* ⟵ updateOptimalSet(X*, x_new)
16                 X* ⟵ pruneOptimalSet(X*)
17             else
18                 ΔE ⟵ c_F(x_new, x_cur)
19                 T ⟵ getTemperature(t)
20                 if random_u() < e^(−ΔE/(k_B T)) then
21                     X*_cur ⟵ X*_cur \ {x_cur}
22                     X*_cur ⟵ X*_cur ∪ {x_new}
23             t ⟵ t + 1
24     return X*
25 end
```

11

Tabu Search

11.1 Introduction

Tabu search[1] (TS) [908, 909, 910, 911, 912] is an optimization method basing on local search. It improves the search performance and decreases the probability of getting stuck at local optima by using internal memory.

Tabu search works on single solution candidate. It creates a neighboring individual of this candidate and uses its memory to check if the new element is allowed. If so, it can transcend to it and again start to explore its neighbors. The simplest version of this method will store n individuals already visited in a tabu list and prohibit visiting them again. More complex variants will store only specific properties of the individuals which are tabu. This will also lead to more complicated algorithms since it also may prohibit new solutions which are actually very good. Therefore, aspiration criteria can be defined which override the tabu list and allow certain individuals. The length of the tabu list is limited to n and therefore the list must be truncated surpassing that maximum length. Therefore, techniques like clustering could be used. In Algorithm 11.1 we just remove the oldest list elements.

[1] `http://en.wikipedia.org/wiki/Tabu_search` [accessed 2007-07-03]

Algorithm 11.1: $x^\star = tabuSearch(f)$

Input: f the objective function to be minimized
Input: Implicit: n the maximum length of the tabu list X_{tabu}
Data: $x_{new} \in \tilde{X}$ the new element created
Data: X_{tabu} the tabu list
Output: $x^\star \in \tilde{X}$ the best element found

```
1  begin
2      x_new ⟵ create()
3      x* ⟵ x_new
4      X_tabu ⟵ ()
5      while ¬terminationCriterion() do
6          if search_u(x_new, X_tabu) < 0 then
7              if f(x_new) < f(x*) then x* ⟵ x_new
8              if |X_tabu| ≥ n then X_tabu ⟵ deleteListItem(X_tabu, 0)
9              X_tabu ⟵ addListItem(X_tabu, x_new)
10         x_new ⟵ mutate(x*)
11     return x*
12 end
```

11.2 General Information

11.2.1 Areas Of Application

Some example areas of application of tabu search are:

Application	References
general manufacturing problems	[913]
traveling salesman problem (TSP)	[914, 915]
quadratic assignment problem	[916]
general combinatorial problems	[908, 909]
optimization of artificial neural networks	[917]
resonance assignment in NMR (Nuclear Magnetic Resonance) spectroscopy	[918]
test design in education	[919]
vehicle routing	[920]

11.3 Multi-Objective Tabu Search

The simple tabu search is very similar to hill climbing and simulated annealing (see Chapter 8 and Chapter 10). Like for those two algorithms, we can define a multi-objective variant for tabu search too (see Algorithm 11.2 for more details).

Algorithm 11.2: $X^\star = tabuSearch(c_F)$

Input: c_F the comparator function which allows us to compare the fitness of two solution candidates, used by *updateOptimalSet*
Input: Implicit: n the maximum length of the tabu list X_{tabu}
Data: $x_{new} \in \tilde{X}$ the new element created
Data: X_{tabu} the tabu list
Output: $X^\star \subseteq \tilde{X}$ the set of the best elements found

```
1  begin
2      X* ⟵ ∅
3      X_tabu ⟵ ()
4      x_new ⟵ create()
5      while ¬terminationCriterion() do
6          if search_u(x_new, X_tabu) < 0 then
7              X* ⟵ updateOptimalSet(X*, x_new)
8              X* ⟵ pruneOptimalSet(X*)
9              if |X_tabu| ≥ n then X_tabu ⟵ deleteListItem(X_tabu, 0)
10             X_tabu ⟵ addListItem(X_tabu, x_new)
11         x_new ⟵ select(X*, 1)[0]
12         x_new ⟵ mutate(x_new)
13     return X*
14 end
```

12

Ant Colony Optimization

12.1 Introduction

Ant colony optimization[1] (ACO) [921, 922, 923, 924, 925, 926] is a bio-inspired optimization method for problems that can be reduced to paths in graphs.

It is based on the metaphor of ants that seek for food. An ant will leave the anthill and begin to wander randomly around. If it finds some food, it will return to the anthill while laying out a track of special pheromones. After deposing the food, it will follow the path to the food item by tracking the pheromones. Other ants will do the same, so many of them will arrive at the food piece. Every ant marks the path again by going back to the anthill, so the pheromone gets stronger and stronger. The ability to follow a pheromone trail is dependent on the amount placed there: the ants movements are always randomized, but the probability of flitting into the direction of pheromones if higher. Pheromones also vanish by time if they are not refreshed. If all of the food is collected cart away, ants will stop putting pheromones onto the track since they cannot find new food at this location anymore. Therefore, the pheromones vaporize and the ants will head to new, random locations.

The process of distributing and tracking pheromones is called stigmery[2]. Stigmery sums up all ways of communication by modifying the environment [927].

These mechanisms are copied by ACO: the 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. They will not follow the marked paths fully, maybe taking other routes at junctions, since their walk is still randomized, just with modified probabilities.

[1] `http://en.wikipedia.org/wiki/Ant_colony_optimization` [accessed 2007-07-03]
[2] `http://en.wikipedia.org/wiki/Stigmergy` [accessed 2007-07-03]

It is interesting to note that even real vector optimizations can be viewed as graph problems, as introduced by Korošec and Šilc in [928].

12.2 General Information

12.2.1 Areas Of Application

Some example areas of application of ant colony optimization are:

Application	References
route planning/TSP	[922, 929, 104, 930]
scheduling	[931]
load balancing	[932]
network routing	[933]
other combinatorial problems	[934]

12.2.2 Conferences, Workshops, etc.

Some conferences, workshops and such and such on ant colony optimization are:

BIOMA: International Conference on Bioinspired Optimization Methods and their Applications
see Section 2.2.2 on page 60

CEC: Congress on Evolutionary Computation
see Section 2.2.2 on page 60

12.2.3 Journals

Some journals that deal (at least partially) with ant colony optimization are (ordered alphabetically):

Adaptive Behavior, ISSN: Online ISSN: 1741-2633, Print ISSN: 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 64)
The Journal of the Operational Research Society (see Section 1.8.3 on page 43)

12.2.4 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.

13

Particle Swarm Optimization

13.1 Introduction

Particle Swarm Optimization[1] (PSO) developed by Kennedy and Eberhard in 1995 [935, 936] is a form of swarm intelligence in which the behavior of a biological social system, for example a flock of birds or a school of fish, is simulated. If the swarm looks for food and one individual part of the swarm found it, the rest will follow. On the other hand, each individual has a degree of freedom, or randomness, in its movement which enables it to find new food accumulations [937, 938, 939, 940].

A swarm of particles in a n-dimensional search space is simulated where each particle p has a position $x(p) \in \tilde{X} \subseteq \mathbb{R}^n$ and a velocity $v(p) \in \mathbb{R}^n$. The position $x(p)$ of a particle p denotes a possible solution for the problem whereas its velocity $v(p)$ determines in which direction the search will continue and if it has an explorative (high velocity) or an exploitive (low velocity) character. Both, the position and the velocity are real vectors. Because of this individual representation, PSO is especially suitable for numerical optimization.

At startup, the positions and velocities of all individuals are randomly initialized. In each step, first the velocity of an particle is updated and then its position. Therefore, each particle p has a memory holding its best position $best(p) \in \tilde{X}$. Furthermore, it also knows a set of topological neighbors $N(p)$, i. e. neighboring particles in a specific perimeter. We can define this neighboring particles for example as set of all individuals which are no further away from $x(p)$ than a given distance δ according to a given distance measure[2] $dist$ (normally, the Euclidian distance):

$$\forall\, p, q \in X_{pop} : q \in N(p) \Leftrightarrow dist(x(p), x(q)) \leq \delta \tag{13.1}$$

[1] `http://en.wikipedia.org/wiki/Particle_swarm_optimization` [accessed 2007-07-03]

[2] See Section 36.1 on page 574 for more information on distance measures.

Assuming that each particle can communicate with its neighbors, the best position obtained so far by any element in $N(p)$ is $best(N(p))$. The optimal position ever obtained by any individual in the population (which the optimization algorithm always keeps track of) is $best(X_{pop})$.

The PSO algorithm may use either $best(N(p))$ or $best(X_{pop})$ in order to adjust the adjust the velocity of the particle p. If it relies on the global best position, the algorithm will converge 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. Equation 13.2 and Equation 13.3 show this update process for the i^{th} components of the two vectors $x(p)$ and $v(p)$ for global and local PSO.

$$v(p)_i = v(p)_i + (c_i * random_u() * (best(p)_i - x(p)_i)) + (d_i * random_u() * (best(X_{pop})_i - x(p)_i)) \tag{13.2}$$

$$v(p)_i = v(p)_i + (c_i * random_u() * (best(p)_i - x(p)_i)) + (d_i * random_u() * (best(N(p))_i - x(p)_i)) \tag{13.3}$$

$$x(p)_i = x(p)_i + v(p)_i \tag{13.4}$$

The learning rate vectors c and d have further influence of the convergence speed. Furthermore, the values of all dimensions of $x(p)$ are normally confined to a certain maximum and minimum of the search space. For the absolute values of the velocity, normally maximum thresholds also exist.

Algorithm 13.1 illustrates the native form of the particle swarm optimization algorithm implementing the update procedure according to Equation 13.3. Like we have done it with hill climbing, this algorithm can easily be generalized for multi-objective optimization and for returning sets of optimal solutions (compare with Section 8.3 on page 224).

Algorithm 13.1: $x^\star = particleSwarmOptimize(f)$

Input: f the function to optimize
Input: Implicit: s the population size
Input: Implicit: n the dimension of the vectors
Data: $X_{pop} \subseteq \tilde{X} \subseteq \mathbb{R}^n$ the particle population
Data: $p \in X_{pop}$ a particle
Data: $i \in 1..n$ a counter variable
Output: $x^\star \in \tilde{X}$ the best value found

```
1  begin
2      X_pop ⟵ createPop(s)
3      while ¬terminationCriterion() do
4          foreach p ∈ X_pop do
5              i ⟵ 1
6              while i ≤ n do
                   // update according to Equation 13.3
                   v(p)_i = v(p)_i + (c_i * random_u() * (best(p)_i − x(p)_i)) +
7                                    (d_i * random_u() * (best(N(p))_i − x(p)_i))
8                  x(p)_i = x(p)_i + v(p)_i
9                  i ⟵ i + 1
10             if f(x(p)) < f(best(N(p))) then best(N(p)) ⟵ x(p)
11             if f(x(p)) < f(best(X_pop)) then best(X_pop) ⟵ x(p)
12     return best(X_pop)
13 end
```

13.2 General Information

13.2.1 Areas Of Application

Some example areas of application of particle swarm optimization are:

Application	References
training of artificial neural networks	[936, 941]
training of hidden Markov models	[942]
global optimization of mathematical functions	[936, 943]
in antenna or filter design	[944]
water resource and quality management	[945]
quantitative structure-activity relationship (QSAR) modeling in chemistry	[946, 947]

13.2.2 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

13.2.3 Conferences, Workshops, etc.

Some conferences, workshops and such and such on particle swarm optimization are:

SIS: IEEE Swarm Intelligence Symposium
`http://www.computelligence.org/sis/` [accessed 2007-08-26]
History: 2007: Honolulu, Hawaii, USA, see [948]
2006: Indianapolis, IN, USA, see [949]
2005: Pasadena, CA, USA, see [950]
2003: Indianapolis, IN, USA, see [951]

14

Memetic Algorithms

Memetic Algorithms[1] [952, 953, 954, 955, 956] (MA) try to simulate cultural evolution rather than the biological one, as evolutionary algorithms do. They are a combination of population-based global optimization and heuristic local search. First, individuals are initialized randomly. Starting with each single individual, one local search is performed. After that, the individuals start to interact either competitively (in the form of selection) or cooperatively (in the form of recombination). These two steps are repeated until the stopping criterion is met.

In the context of this work, we will regard memetic algorithms as two-level optimization algorithm, where the top-level algorithm is an evolutionary or otherwise population based algorithm and at the bottom-level, a single individual optimizer like hill climbing or simulated annealing (see Chapter 8 and Chapter 10) can be found.

In order to create such algorithms, one would replace the *createPop* and *reproducePop* algorithms (introduced in Section 2.5 on page 99) by their locally optimizing counterparts $createPopMA$ and $reproducePopMA$ (see Algorithm 14.1 and Algorithm 14.2). The $localOptimize(c_F, x)$-calls on line 5 of each of the two algorithms would then replaced to calls to a hill climber or a simulated annealing algorithm which would start with the supplied element x instead of a randomly created on. Now $createPopMA$ and $reproducePopMA$ can for example be implanted into any given evolutionary algorithm dealt with so far (see Chapter 2).

[1] `http://en.wikipedia.org/wiki/Memetic_algorithm` [accessed 2007-07-03]

Algorithm 14.1: $X_{pop} = createPopMA(n)$

Input: n the size of the population to be created
Input: Implicit: c_F the optimization criterion
Data: i a counter variable
Output: X_{pop} the new, random but locally optimized population

```
1 begin
2     X_pop ⟵ ∅
3     i ⟵ n
4     while i > 0 do
5         X_pop ⟵ X_pop ∪ {localOptimize(c_F, create())}
6         i ⟵ i − 1
7     return X_pop
8 end
```

Algorithm 14.2: $X_{new} = reproducePopMA(X_{mp}, k)$

Input: k the count of offspring to create
Input: Implicit: c_F the optimization criterion
Data: x the new element subject to optimization
Output: X_{new} the population containing the locally optimized offspring of X_{mp})

```
1 begin
2     X_tmp ⟵ reproducePop(X_mp, k)
3     X_new ⟵ ∅
4     foreach x ∈ X_tmp do
5         X_new ⟵ X_new ∪ {localOptimize(c_F, x)}
6     return X_new
7 end
```

15

State Space Search

15.1 Introduction

State space search strategies[1] are not directly counted as optimization algorithms. Instead, they are means to browse and find valid solutions the search space $\tilde{X}$. If we however assume that there we can define a threshold for each objective function $f \in F$ below which an individual is a valid solution, such a strategy can be applied. Another condition is that $\tilde{X}$ must be enumerable [866, 957, 958].

One feature of the state space 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.

Generally, two operations must be defined in such search algorithms: one that tells us if we have found what we are looking for (*isGoal*) and one that helps enumerating the search space (*expand*).

Definition 72 (isGoal). The function $isGoal(x) \in \{\texttt{true}, \texttt{false}\}$ is the target predicate of state space search algorithms that tells us whether a given state $x \in \tilde{X}$ is a valid solution (by returning `true`), i. e. the goal state, or not (by returning `false`).

If we consider *isGoal* again from the perspective of optimization, $isGoal(x)$ becomes `true`if and only if all objective functions $f_i(x)$ have a value lower than a corresponding threshold $\check{y}_i$. In other words, $x \in \tilde{X}$ is a valid solution if it satisfies all constraints.

$$isGoal(x) \Leftrightarrow f_i(x) \leq \check{y}_i \; \forall \; x \in \tilde{X}, i = 1 \ldots |F| \tag{15.1}$$

Definition 73 (expand). The operator $expand(x)$ computes a set of solution candidates (states) X_{new} from a given state x. It is the exploration

[1] `http://en.wikipedia.org/wiki/State_space_search` [accessed 2007-08-06]

operation of state space search algorithms. Different from the mutation operator of evolutionary algorithms (see Definition 44 on page 100), it is strictly deterministic and returns a set instead of single individual. Applying it to the same x values will always yield the same set X_{new}. We can consider $expand(x)$ to return the set of all possible result that we could obtain with $mutate(x)$.

$$expand(x) = X_{new},\ x \in \tilde{X}, X_{new} \subseteq \tilde{X} \tag{15.2}$$

$$expand(x) \equiv \{\forall\ s : s = mutate(x)\} \tag{15.3}$$

The realization of *expand* may have severe impact on the performance of search algorithms. An efficient implementation for example, should try to prevent that states that have already been visited are returned. It may be possible to reach the same solution candidate x by mutating different parents $x'_1, x'_2, \ldots$. Hence, x would be included in $expand(x'_1)$, in $expand(x'_2)$, and so on, if no further measures are applied. This will lead to the same solution candidate (and all of its children) to be evaluated multiple times, which is obviously useless. Another problem is that one can reach x from its parent $x \in expand(x')$ but possible also the parent from x ($x' \in expand(x)$). This is even more serious since, if not prevented, will lead to infinite loops in the search. Therefore, some form of caching or tabu lists should be used, as done in the previously discussed tabu search (see Chapter 11 on page 237).

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)?
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 37.1.3 on page 589.
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 74 (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 [866].

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, which, in turn, can be regarded as a special case of global optimization algorithms.

15.2 Uninformed Search

The optimization algorithms that we have considered up to now always require some sort of measure of the utility of possible solutions. These measures, the objective functions, are normally real-valued and allow us to make fine distinctions between different individuals. Under some circumstances, maybe only the 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 15.2.5 on page 256).

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, for example in form of heuristic functions, the search may take very long and quickly becomes infeasible [866, 957, 958].

15.2.1 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, and 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, if the number of expansion steps needed from the origin to a state is a measure for the costs, BFS is also optimal.

Algorithm 15.1 illustrates how breath-first search works. The algorithm starts with a root state $r \in \tilde{X}$ which marks the starting point of the search. We create a list initially only containing this state. In a loop we remove the first element s of that list and check whether the predicate $isGoal(s)$ evaluates to `true` or not. If s is a goal state, we can return a set $X^\star$ containing it as the solution. Otherwise, we expand s and append the newly found states to the end of queue S. If no solution can be found, this process will continue until the whole search space has been enumerated and S becomes empty. Then, an

[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]

Algorithm 15.1: $X^\star = bfs(r)$

Input: $r \in \tilde{X}$ the root node to start the expansion at
Input: Implicit: *expand* the expansion operator
Input: Implicit: *isGoal* an operator that checks whether a state is a goal state or not
Data: $s \in \tilde{X}$ the state currently processed
Data: $S \in \tilde{X}$ the queue of states to explore
Output: $X^\star \subseteq \tilde{X}$ the solution states found, or $\emptyset$

```
1 begin
2     S ⟵ (r)
3     while S ≠ ∅ do
4         s ⟵ deleteListItem(S, 0)
5         if isGoal(s) then return {s}
6         S ⟵ appendList(S, expand(s))
7     return ∅
8 end
```

empty set is returned in place of $X^\star$, because there is no element $x \in \tilde{X}$ for which $isGoal(x)$ becomes `true`.

In order to examine the space and time complexity of BFS, we assume a hypothetical state space $\tilde{X}_h$ where the expansion of each state $x \in \tilde{X}_h$ will return a set of $|expand(x)| = b$ 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

$$1 + b + b^2 + \ldots + b^d = \frac{b^{d+1} + 1}{b - 1} \in \mathcal{O}(b^d) \tag{15.4}$$

We have both, a space and time complexity from $\mathcal{O}(b^d)$. In the worst case, all nodes in depth d need to be stored, in the best case only those of depth $d - 1$.

15.2.2 Depth-First Search

Depth-first search[4] (DFS) is very similar to BFS. From the algorithmical 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 15.2 with line 4 in Algorithm 15.1). 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 is the reason for its name). It advances in depth until the current state cannot further be expanded, i. e. $expand(s) = \emptyset$. Then the search steps again up one level. If the whole search space has been browsed and no solution is found, $\emptyset$ is returned.

[4] `http://en.wikipedia.org/wiki/Depth-first_search` [accessed 2007-08-06]

Algorithm 15.2: $X^\star = dfs(r)$

Input: $r \in \tilde{X}$ the root node to start the expansion at
Input: Implicit: *expand* the expansion operator
Input: Implicit: *isGoal* an operator that checks whether a state is a goal state or not
Data: $s \in \tilde{X}$ the state currently processed
Data: $S \in \tilde{X}$ the stack of states to explore
Output: $X^\star \subseteq \tilde{X}$ the solution states found, or $\emptyset$

```
1 begin
2     S ⟵ (r)
3     while S ≠ ∅ do
4         s ⟵ deleteListItem(S, |S| − 1)
5         if isGoal(s) then return {s}
6         S ⟵ appendList(S, expand(s))
7     return ∅
8 end
```

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, then the time complexity is b^m in the 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.

15.2.3 Depth-limited Search

The depth-limited search[5] [866] 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 15.3. Analogously to the plain depth first search, the time complexity now becomes b^d and the memory complexity is in $\mathcal{O}(b*d)$. 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.

15.2.4 Iterative deepening depth-first search

The iterative deepening depth-first search[6] (IDDFS, [866]), defined in Algorithm 15.4, iteratively runs a depth-limited DFS with stepwise increasing maximum depths d. In each iteration, it visits the states in the state space according to the depth-first search. Since the maximum depth is always

[5] `http://en.wikipedia.org/wiki/Depth-limited_search` [accessed 2007-08-07]
[6] `http://en.wikipedia.org/wiki/IDDFS` [accessed 2007-08-08]

Algorithm 15.3: $X^\star = dl\_dfs(r, d)$

Input: $r \in \tilde{X}$ the node to be explored
Input: $d \in \mathbb{N}$ the maximum depth
Input: Implicit: *expand* the expansion operator
Input: Implicit: *isGoal* an operator that checks whether a state is a goal state or not
Data: $s \in \tilde{X}$ the state currently processed
Data: $S \in \tilde{X}$ set of "expanded" states
Output: $X^\star \subseteq \tilde{X}$ the solution states found, or $\emptyset$

```
1 begin
2     if isGoal(r) then return {r}
3     if d > 0 then
4         foreach s ∈ expand(r) do
5             S ⟵ dl_dfs(s, d − 1)
6             if S ≠ ∅ then return S
7     return ∅
8 end
```

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 a 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 $\mathcal{O}(d * b)$. The time consumption, of course, is still in $\mathcal{O}(b^d)$. IDDFS is the best uninformed search strategy and can be applied to large search spaces with unknown depth of the solution.

15.2.5 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. [959, 960]

From the viewpoint of evolutionary algorithms, we can also define a random like an optimization method working on small populations (usually size 1). In each step, one offspring is created using some predefined reproduction method *next* (which could be mutation, for instance) and always replaces the current population regardless if it is better or worse. It is very often assumed that the result of *next* is uniformly distributed amongst all possibilities (where *expand* defines the set of possible offspring), i. e.

$$\forall x, y \in X : P(y = next(x)) = \begin{cases} \frac{1}{|expand(x)|} & \text{if } y \in expand(x) \\ 0 & \text{otherwise} \end{cases} \tag{15.5}$$

[7] `http://en.wikipedia.org/wiki/Random_walk` [accessed 2007-11-27]

Algorithm 15.4: $X^\star = iddfs(r)$

Input: $r \in \tilde{X}$ the node to be explored
Input: Implicit: *expand* the expansion operator
Input: Implicit: *isGoal* an operator that checks whether a state is a goal state or not
Data: $d \in \mathbb{N}$ the (current) maximum depth
Output: $X^\star \subseteq \tilde{X}$ the solution states found, or $\emptyset$

```
1 begin
2   d ⟵ 0
    /* This algorithm is for infinitely large search spaces. In
       real systems, there is a maximum d after which the whole
       space would be explored and the algorithm should return ∅ if
       no solution was found.                                    */
3   repeat
4     X* ⟵ dl_dfs(r, d)
5     d ⟵ d + 1
6   until X* ≠ ∅
7   return X*
8 end
```

Under some circumstances, random walks can be the search algorithms of choice. This for instance the case in

- If we encounter a state explosion because there are too many states to which we can possible transcend to and methods like breadth-first or depth-first search cannot be applied because they would consume too much memory.
- 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 about reasoning agents [961].

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, the average difference between two consecutive populations, and such and such. However, also from practically running random walks some information about the search space can be extracted. Skubch for instance uses the number of encounters of certain state transition during the walk in order to successively build a heuristic function [961].

15.3 Informed Search

In an informed search[8], a heuristic function helps to decide which nodes are to be expanded next. If the heuristic is good, informed search algorithms may dramatically outperform uninformed strategies [8, 9, 10].

As specified in Definition 2 on page 6, heuristic functions are problem domain dependent. In the context of an informed search, a heuristic function $h : \tilde{X} \mapsto \mathbb{R}^+$ maps the states in the state space $\tilde{X}$ to the positive real numbers $\mathbb{R}^+$. The value $h(s)$ should be some form of *estimate* on how likely expanding or testing the state s will lead to a correct solution or how many expansion steps a correct solution is away. Here we focus on the latter notation which makes heuristics subject to minimization. This also means that all heuristics become zero if s already is a valid solution.

$$\forall\, s \in \tilde{X} : isGoal(s) \Rightarrow h(s) = 0 \;\forall\, heuristics\; h : \tilde{X} \mapsto \mathbb{R}^+ \tag{15.6}$$

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 $h(s)$ for a state s is the higher, the more *expand*-steps s is *probably* (or *approximately*) away from 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 optimal value o for an objective function f, or at least, a value from where on all solutions are feasible. If this is the case, we could for example set $h(s) = \max\{0, f(s) - o\}$, 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 15.6 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 solution space are probably also close to each other in objective space (the opposite does not necessarily hold).

A best-first search[9] is a search algorithm that incorporates such an estimation function v in a way that promising solution candidates s with low estimation values $v(s)$ are evaluated before other states t that receive a higher values $v(t) > v(s)$. For estimation functions, the same constraints are valid as for heuristic functions. Matter of fact, an estimation may be a heuristic function itself (as in greedy search) or be based on a heuristic function (as in A* search).

[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]

15.3.1 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. Here, the estimation function is the heuristic function itself.

The greedy algorithm internal sorts the list of currently known states in *descending* order according to a comparator function $c_h(x_1, x_2) \in \mathbb{R}$. As a comparator function defined in compliance with Section 1.3.5 on page 20, c_h will be below zero if x_1 should be preferred instead of x_2 ($h(x_1) < h(x_2)$) and higher then zero for all $h(x_1) > h(x_2)$, which indicate that x_2 is more a more prospective solution candidate. Thus, the elements with the best heuristic value will be at the end of the list, which then can be used as a stack.

$$c_h(x_1, x_2) = h(x_1) - h(x_2) \tag{15.7}$$

The greedy search as specified in Algorithm 15.5 now works like a depth-first search on this stack. It 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 15.5: $X^\star = greadySearch(r)$

Input: $r \in \tilde{X}$ the root node to start the expansion at
Input: Implicit: $h : \tilde{X} \mapsto \mathbb{R}^+$ the heuristic function
Input: Implicit: *expand* the expansion operator
Input: Implicit: *isGoal* an operator that checks whether a state is a goal state or not
Data: $s \in \tilde{X}$ the state currently processed
Data: $S \in \tilde{X}$ the sorted list of states to explore
Output: $X^\star \subseteq \tilde{X}$ the solution states found, or $\emptyset$

```
1 begin
2     S ⟵ (r)
3     while S ≠ ∅ do
4         S ⟵ sort_d(S, c_h)
5         s ⟵ deleteListItem(S, |S| − 1)
6         if isGoal(s) then return {s}
7         S ⟵ appendList(S, expand(s))
8     return ∅
9 end
```

Notice that we can replace c_h with any other valid comparator function. In principle, we could even apply objective functions and Pareto-based comparisons here.

[10] http://en.wikipedia.org/wiki/Greedy_search [accessed 2007-08-08]

15.3.2 A* Search

In A* search[11] is a best-first search that uses a estimation function $h^* : \tilde{X} \mapsto \mathbb{R}^+$ which is the sum of a heuristic function $h(s)$ that estimates the costs needed to get from s to a valid solution and a function $g(s)$ that computes the costs that we had to find s.

$$h^*(s) = g(s) + h(s) \tag{15.8}$$

A* search proceeds exactly like the greedy search outlined in Algorithm 15.5, if h^* is used instead of plain h. A* search will definitely find a solution if there exists one, i. e. it is complete.

Definition 75 (Admissible Heuristic Function). A heuristic function $h : \tilde{X} \mapsto \mathbb{R}^+$ is admissible if it never overestimates the minimal costs of reaching a goal.

Definition 76 (Monotonic Heuristic Function). A heuristic function $h : \tilde{X} \mapsto \mathbb{R}^+$ is monotonic[12] if it never overestimates the costs from getting from one state to its successor.

$$h(s) \leq g(s') - g(s) + h(s') \; \forall \; s' \in expand(s) \tag{15.9}$$

An A* search is optimal if the heuristic function h used is admissible. If we implement *expand* in a way the prevents that a state is visited more than once, h also needs to be monotone in order for the search to be optimal.

15.3.3 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:

- *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.
- *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.

[11] `http://en.wikipedia.org/wiki/A%2A_search` [accessed 2007-08-09]
[12] see Definition 115 on page 510

- *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 climbers 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 [962, 479]. The dynamics of such populations in near-equilibrium states with low mutation rates can be approximated with one-mutant adaptive walks [963, 962, 479].

16

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 [964, 965]. When painting a fence, 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 [966, 967]. 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. Off the shelf PCs usually have no more than two CPUs, limiting the benefit of local parallelization. We can however connect arbitrarily many of such computers in a network for distributed processing.

16.1 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 in a very general manner for evolutionary algorithms as example for population-based optimizers.[4].

[1] Section 37.2 on page 592 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 2.1.1 on page 51 you can find the basic evolutionary algorithm.

There are two parts in evolutionary algorithms whose performance potentially can be increased by parallelization: the evaluation and the reproduction stages. As sketched in Figure 16.1, 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. 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 again 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 *create* operation, $n = 1$ resembles mutation, and $n = 2$ means crossover. Each creation of a new solution candidate is an independent task.

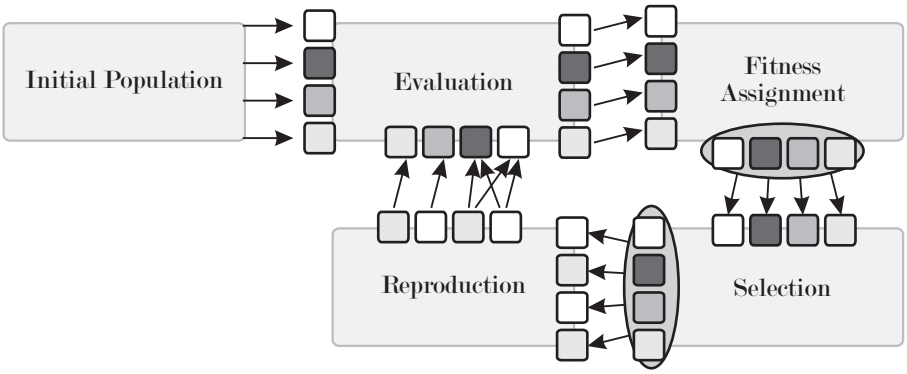


Fig. 16.1: Parallelization potential in evolutionary algorithms.

Despite running an evolutionary algorithm in single a thread[5] of execution (see Figure 16.2), our "analysis" has shown that makes sense to have at least the evaluation and reproduction phase executed in parallel as illustrated in Figure 16.3. Usually, the population is larger than the number of

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

available CPUs[6], so one thread is 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 come with hyper-threading[7] technology [968, 969] or even dual-core[8] CPUs [970, 971].

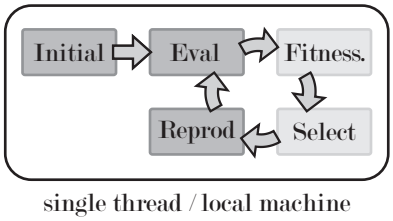


Fig. 16.2: A sequentially proceeding evolutionary algorithm.

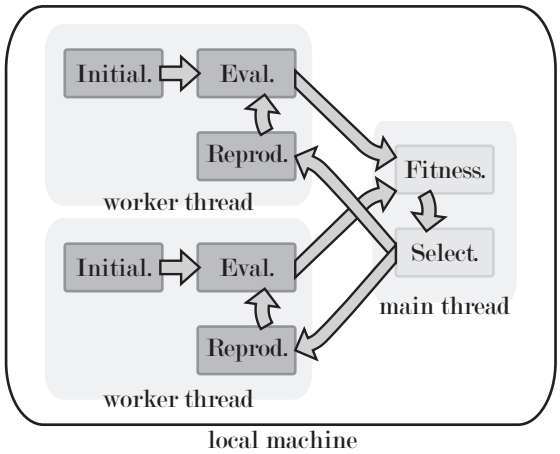


Fig. 16.3: A parallel evolutionary algorithm with two worker threads.

Parallel evolutionary algorithms can be divided into two classes [972]:

- In *globally* parallelized EAs, each individual in the population can (possible) always mate with any other.

[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]

- 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 the following, we are going to discuss some of the different parallelization methods from the viewpoint of distribution because of its greater generality.

16.2 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 [552].

16.2.1 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[973, 974] (also called master-slave).[9]

Figure 16.4 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 [975, 976, 977]. 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 can than be integrated into the population where fitness assignment and selection is performed.

In a practical realization, we can use a queue where all the selected individuals are pushed into as mating pool. Each server in the network could be represented by a thread on the client side. These threads successively pull individuals from the queue, send them to their according server, wait for the result to be returned, place the individuals they receive into the new population, and then starts over again. The servers may posses multiple processors, which can be taken into account by representing each one by an appropriate number of threads.

[9] A general discussing concerning the client-server architecture can be found in Section 37.2.2 on page 596

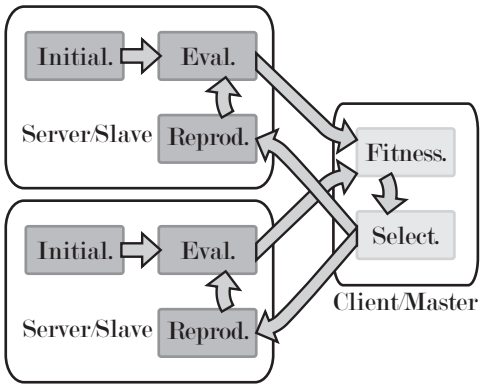


Fig. 16.4: An EA distributed according to the client-server approach.

16.2.2 Island Model

Under some circumstances however, the client-server approach may not be optimal, especially if

- 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 actual work would take if performed locally, the client/server method would actually slow down the system.
- 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 however there may occur obstacles like mountain ranges or oceans, separating parts of the population and creating isolated sub-populations. Another example for such a scenario is an archipelago like the Galapagos islands where Darwin, the father of the evolution theory performed his studies [5]. 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, they can compete and survive in the new habitat. Otherwise, they will be extruded by its native residents. This way, all islands manage an approximately equal level of fitness of their individuals while still preserving a large amount of diversity.

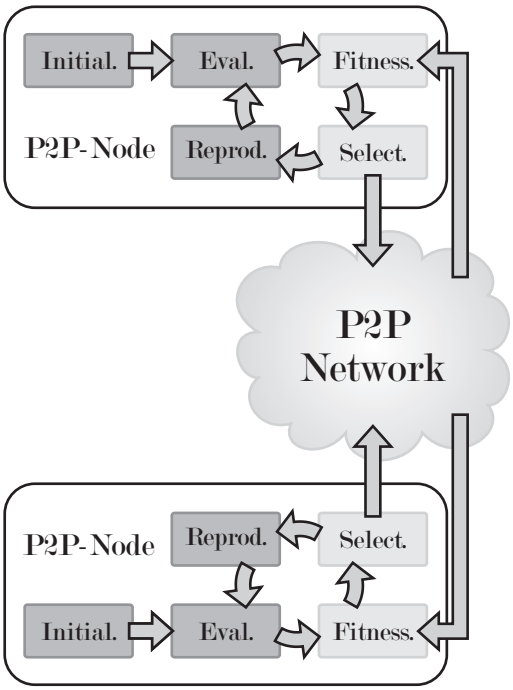


Fig. 16.5: An evolutionary algorithm distributed in a P2P network.

Using separate sub-populations, we can easily copy this natural role model for evolutionary algorithms [978, 979, 980, 981, 982, 983, 984] and use it for coarse grained parallelization. By distributing these sub-populations on n different nodes in a network of computers, each representing one island, both disadvantages of the original master/slave approach are circumvented.

Communication between nodes is only needed when individuals *migrate* between the nodes. This communication can be performed asynchronously to the n independently running evolutionary algorithms and does not slow down their performance. It is determined by the migration rule which can 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 second disadvantage mentioned, the memory consumption problem.

This island model can be realized by peer-to-peer networks[10] as illustrated in Figure 16.5, where each node runs an independent evolution. 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

[10] P2P networks are discussed in Section 37.2.2 on page 597.

problems, solution candidates migrating over the network can either enter the fitness assignment process on the receiving machine directly are may take part in the evaluation process first. If the latter is the case, different objective functions can applied on the nodes.

Driving this thought further, one will recognize that the peer-to-peer approach inherently allows mixing of different optimization technologies [552]. On one node, for instance, the SPEA2-algorithm (see Section 2.6.14 on page 111) may run whereas another node could optimize according to plain hill climbing as described in Chapter 8 on page 223. Such a system, illustrated in Figure 16.6, has one striking advantage: for different problems, different optimization algorithms will perform best. If the problem is for example unimodal, i. e. has exactly one global optimum and no local optima, the hill climber will outperform any other technique since it will directly converge to this optimum. If the fitness landscape is rugged one the other hand, methods like SPEA2 that have a very balanced exploration/exploitation proportion are able to yield better results while hill climbers 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, the hill climber will be fast in developing this solution candidate further until its best form is found, i. e. the top of this local optimum is reached. In other phases, an exploration of the solution space may be required since all known local optima have been tracked; another 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 [406, 985, 986, 987, 988].

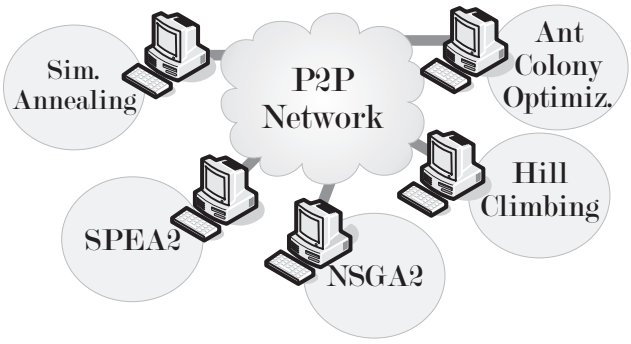


Fig. 16.6: An example for a heterogeneous search.

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. for example argue that the island model can be used to preserve the solution diversity [989]. By doing so it decreases the probability of premature convergence (see Section 1.4.1 on page 21).

16.2.3 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 outlined in Figure 16.7. Such a system will be especially powerful if we need large populations of individuals that take long to evaluate. 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.

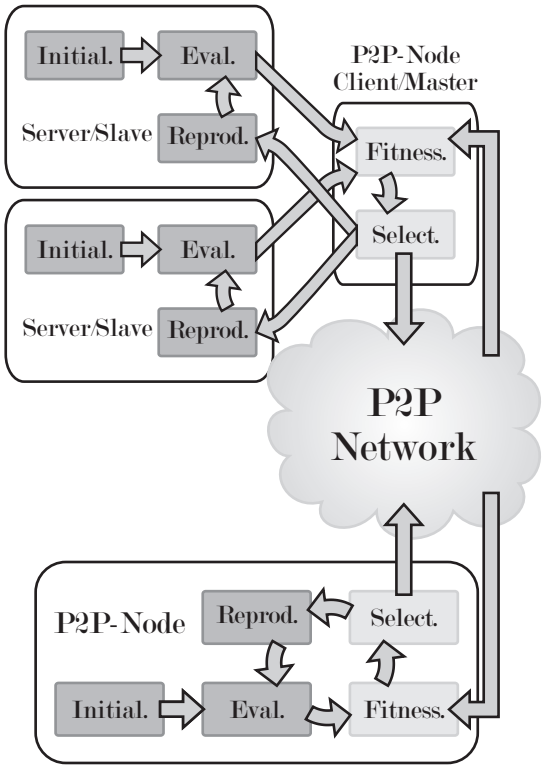


Fig. 16.7: A mixed distributed evolutionary algorithms.

16.3 Cellular GA

The cellular genetic algorithm is a special parallelization model for genetic algorithms. A good understanding of this model can be reached by starting with the basic architecture of the cellular system [360].

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_{ij} can thus communicate with $m_{(i+1) \bmod N\ j}$, $m_{(i-1) \bmod N\ j}$, $m_{i\ (j+1) \bmod N}$, and $m_{i\ (j-1) \bmod N}$.[11]

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 such fixed mating restrictions like in the island model, regions that are about 20 or so moves away will virtually not influence each other. We can consider groups of cells that distant as separate subpopulations. This form of separation is called *isolation by distance* – again, a term that originally stems from biology [12, 360, 990, 991, 983]. For observing such effects, it is said that a certain minimum of cells is required (at least about 1000) [360].

[11] It is also possible to extend this neighborhood to all cells in a given Manhattan distance larger than one, but this here is the easiest case.

Part II

Applications

17

Benchmarks and Toy Problems

In this chapter, we discuss some benchmarks and toy problems that are used to demonstrate the utility of global optimization algorithms. They usually do not have a direct real-world application but are well understood, widely researched, and can be used

- to measure the speed and the convergence-ability of evolutionary algorithms and newly developed techniques (as done for example in [992, 732]),
- as basis to verify theories (used for instance in [993]),
- as playground to test new ideas, research, and developments,
- as easy-to-understand examples to discuss features/problems of optimization problems (as done here in Section 1.3 on page 13),
- for demonstration purposes since they normally are interesting, funny, and can be visualized in a nice manner.

17.1 Benchmark Functions

Benchmark functions are especially interesting for testing and comparing techniques like plain evolution strategy (see Chapter 5 on page 203), differential evolution (see Section 5.5 on page 206), and particle swarm optimization (see Chapter 13 on page 245). The optimum of these functions is already known and we are interested in the number of solution candidates that need to be processed to find it. 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.

17.1.1 Single-Objective Optimization

In this section, we list some of the most important benchmark functions for scenarios involving only a single objective. 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 [994] (or De Jong's F_1 [209]) is a very simple measure of efficiency of optimization methods. They have, for instance, been used by Rechenberg for testing his Evolution Strategy-approach [470].

function	$f_{sphere}(x) = \sum_{i=1}^{n} x_i^2$	(17.1)
domain	$\tilde{X} \in \mathbb{R}^n,\ \tilde{X}_i \in [-10, 10]$	(17.2)
optimum	$x^\star = (0, 0, \ldots, 0)^T$	(17.3)
separable	yes	
multimodal	no	

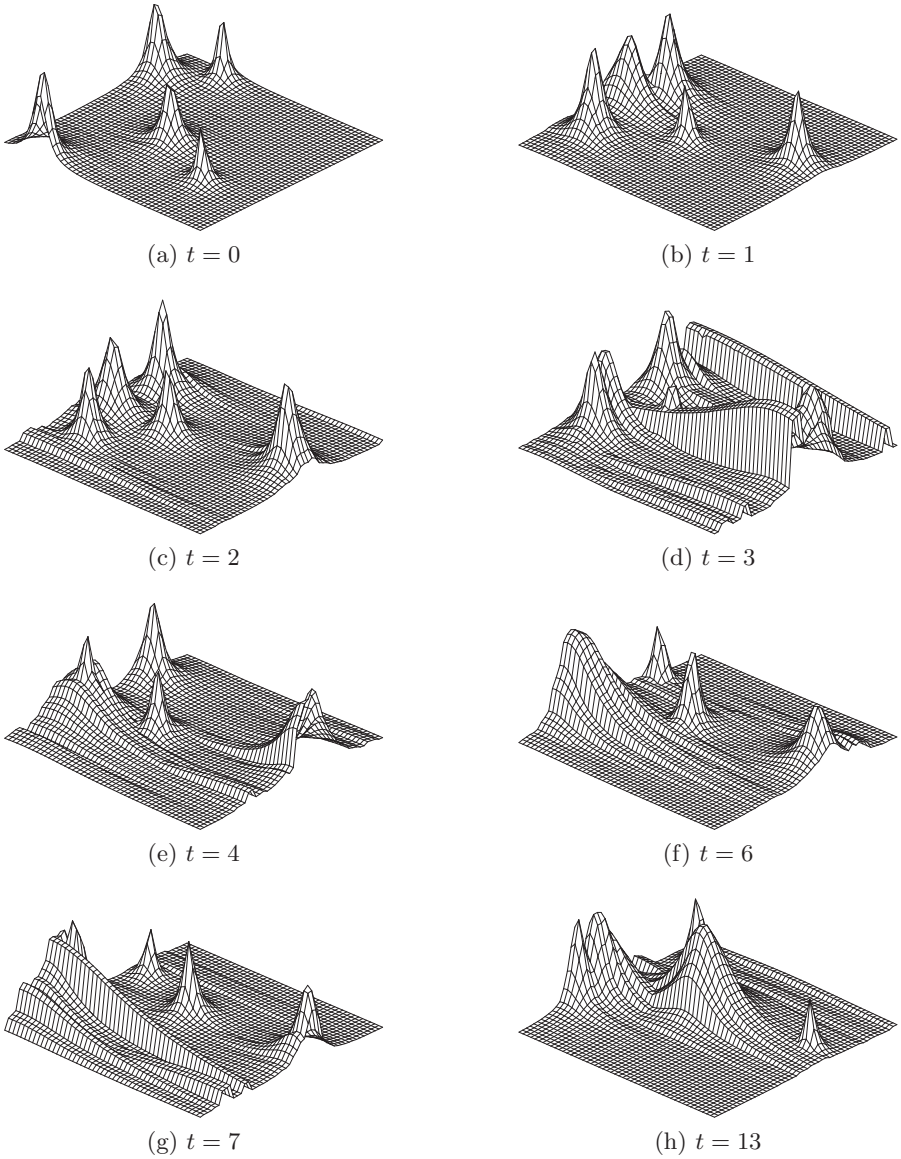
todo

17.1.2 Multi-Objective Optimization

In this section, we list some of the most important benchmark functions for scenarios involving multiple-objectives (see Section 1.3 on page 12). todo

17.1.3 Dynamic Fitness Landscapes

The moving peaks benchmarks independently developed by Branke [87] and Morrison and De Jong [85] illustrate the behavior of dynamic environments as discussed in Section 1.4.5 on page 27. Figure 17.1 shows an example of this benchmark for a two-dimensional real parameter setting (the third dimension is the fitness).



(a) $t = 0$ (b) $t = 1$

(c) $t = 2$ (d) $t = 3$

(e) $t = 4$ (f) $t = 6$

(g) $t = 7$ (h) $t = 13$

Fig. 17.1: An example for the moving peaks benchmark [87].

17.2 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 theory. It is thus not surprising that biologists also contributed very much to the research of their interaction. In the late 1980s, Kauffman defined the *NK fitness landscape* [995, 996, 997], a family of fitness functions with tunable epistasis, in an effort to investigate the links between epistasis and ruggedness.

The genome of this problem are bit strings of the length N ($\tilde{X} = \mathbb{G} = \{0,1\}^N$), where each bit is treated as a single gene. In terms of the NK landscape, only one single objective function is used and referred to as fitness function $F_{N,K} : \{0,1\}^N \mapsto \mathbb{R}^+$. Each gene x_i contributes one value f_i 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 $x_{i_1}, x_{i_2}, \ldots, x_{i_K}$ with $i_{1\ldots K} \in [0, N-1] \setminus \{i\} \subset \mathbb{N}_0$, called its *neighbors*.

$$F_{N,K}(x) = \frac{1}{N} \sum_{i=0}^{N-1} f_i(x_i, x_{i_1}, x_{i_2}, \ldots, x_{i_K}) \tag{17.4}$$

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. If $K = 0$, there is no such 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 one 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 can consider the f_i as single objective functions that are combined to a fitness value $F_{N,K}$ by averaging. 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 [13], Weinberger [998], and Fontana et al. [999] will be discussed here. We therefore borrow from the summaries provided in [479] and [1000].

[1] Fitness landscapes have been introduced in Section 1.2.3 on page 12.
[2] Epistasis is discussed in Section 3.7.2 on page 135.

17.2.1 $K = 0$

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.

- There is a single optimum $x^\star$ which is globally attractive, i. e. which can and will be found by any (reasonable) optimization process regardless of the initial configuration.
- For each individual $x \neq x^\star$ there exists a fitter neighbor.
- 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 $\frac{N}{2}$.
- The fitness of direct neighbors is highly correlated since it shares $N - 1$ components.

17.2.2 $K = N - 1$

For $K = N - 1$, the fitness function equals a random assignment of fitness to each point of the search space.

- The probability that a genotype is a local optimum is $\frac{1}{N-1}$.
- The expected total number of local optima is thus $\frac{2^N}{N+1}$.
- The average distance between local optima is approximately $2 \ln (N - 1)$.
- The expected length of adaptive walks is approximately $\ln (N - 1)$.
- The expected number of mutants to be tested in an adaptive walk before reaching a local optimum is $\sum_{i=0}^{\log_2 (N-1)-1} 2^i$.
- 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* [13].

17.2.3 Intermediate K

- For small K, the best local optima share many common alleles. As K increases, this correlation diminishes, for the random neighbors method faster than for the nearest neighbors method.
- For larger K, the fitness of the local optima approach a normal distribution with mean m and variance s approximately

$$m = \mu + \sigma\sqrt{2 \ln (K + 1)}K + 1 \quad s = \frac{(K+1)\sigma^2}{N(K+1+2(K+2)\ln (K+1))} \tag{17.5}$$

  where μ is the expected value of the f_i and σ^2 is their variance.

[3] See Section 15.3.3 on page 260 for a discussion of adaptive walks.

- The mean distance between local optima, roughly twice the length of an adaptive walk, is approximately $\frac{N \log_2 (K+1)}{2(K+1)}$.
- The autocorrelation function $\rho(k, F_{N,k})$ and the correlation length τ are:

$$\rho(k, F_{N,k}) = \left(1 - \frac{K+1}{N}\right)^k \quad , \quad \tau = \frac{-1}{\ln\left(1 - \frac{K+1}{N}\right)} \tag{17.6}$$

17.2.4 Computational Complexity

[479] nicely summarizes the four most important theorems about the computational complexity of optimization of NK fitness landscapes. These theorems have been proven by different algorithms introduced by Weinberger [1001] and Thompson and Wright [1002].

- The NK optimization problem with adjacent neighbors is solvable in $\mathcal{O}(2^K N)$ steps and thus in $\mathcal{P}$ [1001].
- The NK optimization problem with random neighbors is $\mathcal{NP}$ complete for $K \geq 2$ [1001, 1002].
- The NK optimization problem with random neighbors and $K = 1$ is solvable in polynomial time. [1002].

17.3 The Royal Road

The Royal Road functions [1003, 465, 1004] which have been presented at the Fifth International Conference on Genetic Algorithms in July 1993 are a set of special fitness landscapes for genetic algorithms with fixed-length bit string genomes. They are closely related to the Schema Theorem[4] and the Building Block Hypothesis[5] 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 fitness functions, subject to maximization, as

$$f(x) = \sum_{\forall s \in S} c(s)\sigma(s, x) \tag{17.7}$$

where $x \in \tilde{X} \equiv \mathbb{G}$ is a bit string, $c(s)$ is a value assigned to the schema s and $\sigma(s, x) \begin{cases} 1 : \textit{if } x \textit{ is an instance of } s \\ 0 : \textit{otherwise} \end{cases}$. In the original version, $c(s)$ is the order of the schema s and S is defined as follows.

```
1  s1  = 11111111************************************************************; c(s1) = 8
2  s2  = ********11111111********************************************************; c(s2) = 8
3  s3  = ****************11111111****************************************************; c(s3) = 8
4  s4  = ************************11111111************************************************; c(s4) = 8
```

[4] See Section 3.6 on page 129 for more details.
[5] The Building Block Hypothesis is elaborated on in Section 3.6.5 on page 132

```
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  = 111111111111111*************************************************; c(s9 ) = 16
10 s10 = ***************1111111111111111*******************************; c(s10) = 16
11 s11 = *******************************1111111111111111***************; c(s11) = 16
12 s12 = ************************************************111111111111111; c(s12) = 16
13 s13 = 11111111111111111111111111111111*******************************; c(s13) = 32
14 s14 = *******************************11111111111111111111111111111111; c(s14) = 32
15 s15 = 1111111111111111111111111111111111111111111111111111111111111111; c(s15) = 64
```

Listing 17.1: An example Royal Road function.

The Royal Road function provides 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, Forrest, and Holland performed several tests with their Royal Road functions. These tests revealed or confirmed that

- Crossover is a useful reproduction operation in this scenario. Genetic algorithms which apply this operation clearly outperform hill climbing approaches solely based on mutation.
- In the sprit 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 Mitchellet al. showed the exact opposite: leaving them away speeds up the evolution significantly [465]. 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.

17.3.1 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. in [1005].

The search space $\tilde{X}_\Sigma$ of the VLR (variable-length representation) Royal Road problem is based on an alphabet Σ with $N = |\Sigma|$ letters. The fitness of an individual $x \in \tilde{X}_\Sigma$ 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

$$B_b(x,l) = \begin{cases} 1 & \textit{if } \exists 0 \leq i < (\textit{length}(x) - b) : x[i+j] = l \; \forall 0 \leq j < (b-1) \\ 0 & \textit{otherwise} \end{cases} \tag{17.8}$$

Where

- $b \geq 1$ is the length of the building blocks,
- Σ is the alphabet with $N = |\Sigma|$ letters,
- l is a letter in Σ,
- $x \in \tilde{X}_\Sigma$ is a solution candidate, and
- $x[k]$ 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 $\textit{length}(x) < b$ this cannot be the case and $B_b(x,l)$ will be zero.

We can now define the functional objective function $f_{\Sigma b} : \tilde{X}_\Sigma \mapsto [0,1]$ which is subject to maximization as

$$f_{\Sigma b}(x) = \frac{1}{N} \sum_{i=1}^{N} B_b(x, \Sigma[i]) \tag{17.9}$$

An optimal individual $x^\star$ 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 of all such optima $X_b^\star$ with $f_{\Sigma b}(x^\star) = 1$.

$$X_b^\star \equiv \left\{ x^\star \in \tilde{X} : B_b(x^\star, l) = 1 \forall l \in \Sigma \right\} \tag{17.10}$$

Such an optimum $x^\star$ for $b = 3$ and $\Sigma = \{A, T, G, C\}$ is

$$x^\star = \mathbf{AAA}\,GT\mathbf{GGG}\,TAA\mathbf{TTT}\,T\mathbf{CCC}\,TCCC \tag{17.11}$$

The relevant building blocks of $x^\star$ 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 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.

17.3.2 Epistatic Road

In [1000], Defoin Platel et al. 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 $N = |\Sigma|$ 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:

$$B_b(x, \Sigma[0])B_b(x, \Sigma[1]) \dots B_b(x, \Sigma[N-1]) \tag{17.12}$$

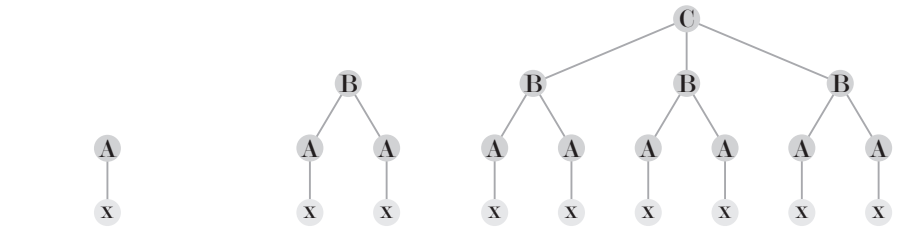
Now we can define a NK landscape for the Epistatic Road by substituting the $B_b(x, l)$ into Equation 17.4 on page 278:

$$F_{N,K,b}(x) = \frac{1}{N} \sum_{i=0}^{N-1} f_i(B_b(x, \Sigma[i]), B_b(x, \Sigma[i_1]), \dots, B_b(x, \Sigma[i_K])) \tag{17.13}$$

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 $0, 1^N$ and defining the f_i in a way that $B_b(x, l) = 1 \; \forall \, l \in \Sigma \Rightarrow F_{N,K,b} = 1$.

17.3.3 Royal Trees

An analogue of the Royal Road for Genetic Programming has been specified by Punch et al. in [1006]. 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.



(a) Perfect A-level (b) Perfect B-level (c) Perfect C-level

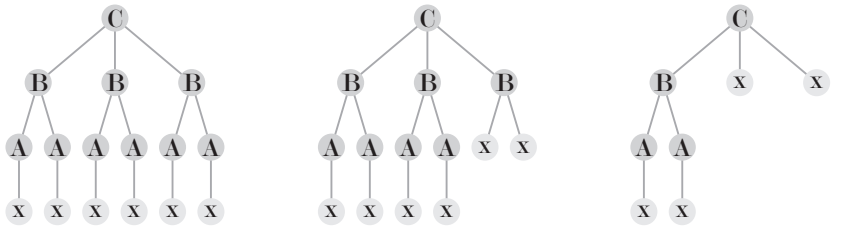
Fig. 17.2: The perfect Royal Trees.

For the first free levels, the perfect trees are shown Figure 17.2. 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), its fitness is multiplied with the constant $FullBonus$, which is normally 2. If the child is not a perfect tree, but has the correct root, the weight is $PartialBonus$ (usually 1). If it is otherwise incorrect, its fitness is multiplied with $Penalty$, which is $\frac{1}{3}$ per default. After evaluating the root of the tree, if it is a perfect tree, the raw fitness is finally multiplied with $CompleteBonus$ which normally is also 2. The value of a of a X leaf is 1.

From [1006] we can furthermore borrow three examples for this fitness assignment, outlined in Figure 17.3. 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 $CompleteBonus(FullBonus{*}32 + FullBonus * 32 + FullBonus * 32) = 2(2 * 32 + 2 * 32 + 2 * 32) = 384$.



(a) $2(2{*}32{+}2{*}32{+}2{*}32) = 384$ (b) $2(2{*}32{+}2{*}32{+}\frac{2}{3}{*}1) = 128\frac{2}{3}$ (c) $2(2{*}32{+}\frac{1}{3}{*}1{+}\frac{1}{3}{*}1) = 64\frac{2}{3}$

Fig. 17.3: Example fitness evaluation of Royal Trees.

17.4 Artificial Ant

We have discussed parts of the artificial ant [1007, 1008, 1009, 11, 1010, 1011, 1012] problem already in Section 1.3 on page 13 and Section 1.2.2 on page 10 – here we are going to investigate it more thoroughly.

The goal of the original problem defined by Jefferson and Collins [1007] is to find a program that controls an artificial ant in an environment. This environment has the following features:

- It is divided in a toroidal grid generating rectangular cells in the plane making the positions of coordinates of all objects discrete.

- There exists exactly one ant in the environment.
- The ant will always be inside one cell at one time.
- 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 the space, the time in the artificial ant problem is also discrete. Thus, the ant may carry out one of the following actions per time unit:

- 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.
- The ant may turn left or right by 90.
- The ant may do nothing in a time unit.

17.4.1 Santa Fe trail

One instance of the artificial ant problem is the "Santa Fe trail" (see Figure 17.4) designed by Christopher Langdon [11]. This 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. In trail of food pellets, there are gaps of five forms:

- one cells along a straight line
- two cells along a straight line
- one cell in a corner
- two cells at a corner (requiring something like a "horse jump" in chess)
- 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 of course 89) and to walk a distance as short as possible in order to do so (the optimal route is illustrated in Figure 17.4). Of course, there will be a time limit set for the ant to perform this task (normally 200 time units).

17.4.2 Solutions

Genetic Algorithm evolving Finite State Machine

Jefferson, Collins et al. [1009] used a conventional genetic algorithm that evolved finite state machines encoded in a fixed-length binary string genome.

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 can specify the finite state

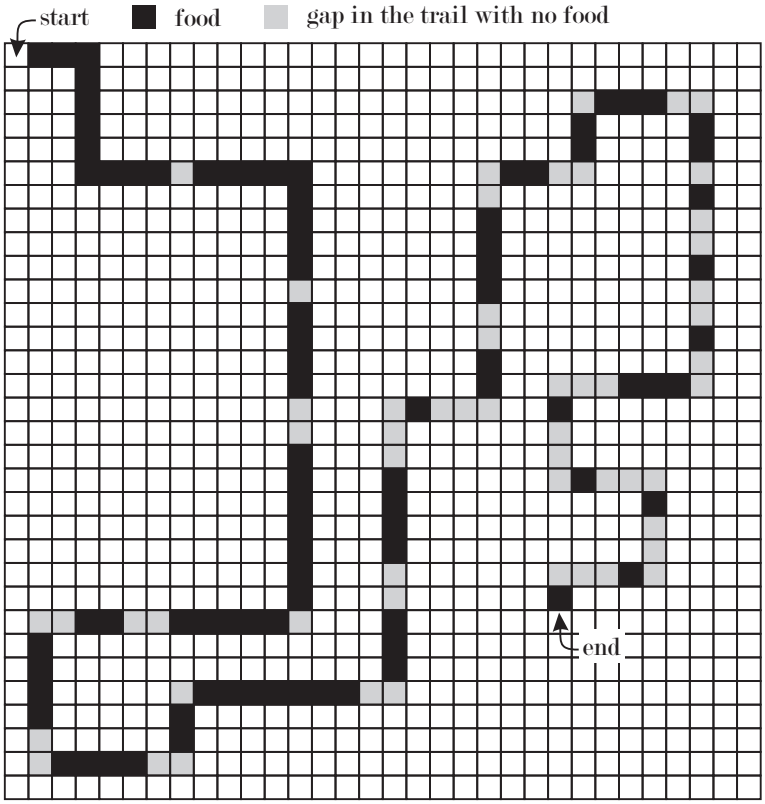


Fig. 17.4: The Santa Fee Trail in the Artificial Ant Problem.

machine as a table where $n + 1$ bits are used as row index. n of this index identify the current state and one bit for the sensor information (`1`=food ahead, `0`=no food ahead). In total, we have $2m$ rows. We do not 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 we also want to store the initial state in the chromosome, we need another n bits to do so. Every chromosome represents a valid finite state machine.

Jefferson, Collins et al. 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 Neural Network

Jefferson, Collins et al. also evolved a neural network (encoded in a 520 bit genome) with a genetic algorithm of the same population size to successfully solve the artificial ant problem.

Genetic Programming evolving Control Programs

Koza [1013] solved the artificial ant problem by evolving LISP[6]-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 two-parameter 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.

17.5 The Greatest Common Divisor

A problem suitable to test genetic programming approaches is to evolve an algorithm that computes the greatest common divisor[7], the GCD.

17.5.1 Problem Definition

Definition 77 (GCD). For two integer numbers $a, b \in \mathbb{N}_0$, the greatest common divisor (GCD) is the largest number c that divides both a ($c|a \equiv a \bmod c = 0$) and b ($c|b \equiv b \bmod c = 0$).

$$c = gcd(a,b) \Leftrightarrow c|a \wedge c|b \wedge (\nexists d : d|a \wedge d|b \wedge d > c) \tag{17.14}$$

$$\Leftrightarrow \max\{e : (a \bmod e = 0) \wedge (b \bmod e = 0)\} \tag{17.15}$$

The Euclidean Algorithm

The GCD can be computed by the Euclidean algorithm[8] which is specified in its original version in Algorithm 17.1 and in the improved, faster variant as Algorithm 17.2 [1014, 1015].

[6] `http://en.wikipedia.org/wiki/Lisp_%28programming_language%29` [accessed 2007-07-03]

[7] `http://en.wikipedia.org/wiki/Greatest_common_divisor` [accessed 2007-10-05]

[8] `http://en.wikipedia.org/wiki/Euclidean_algorithm` [accessed 2007-10-05]

Algorithm 17.1: $gcd(a, b) = euclidGcdOrig(a, b)$

Input: $a, b \in \mathbb{N}_0$ two integers
Data: $t \in \mathbb{N}_0$ a temporary variable
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 17.2: $gcd(a, b) = euclidGcd(a, b)$

Input: $a, b \in \mathbb{N}_0$ two integers
Data: $t \in \mathbb{N}_0$ 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 have evolved a program $x \in \tilde{X}$ which takes the two values a and b as input parameters and returns a new value $c = x(a, b)$. Unlike in symbolic regression[9], 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 $(c - gcd(a, b))^2$. 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 $f_1 : \tilde{X} \mapsto [0, 5]$ for this problem is Algorithm 17.3. It takes a test case (a, b) as argument and first checks whether the evolved program $x \in \tilde{X}$ returns the correct result for it. If so, $f_1(x)$ 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.

[9] See Section 19.1 for an extensive discussion of symbolic regression.

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 1 additional point for each violation. This objective function is very rugged and can take on only integer values between 0 (the optimum) and 5 (the worst case).

Algorithm 17.3: $f_1^{a,b}(x) \equiv euclidObjective(x, a, b)$

Input: $a, b \in \mathbb{N}_0$ the test case
Input: $x \in \tilde{X}$ the evolved algorithm to be evaluated
Data: v a variable holding the result of x for the test case
Output: $f_1^{a,b}(x) = r$ the objective value of the functional objective function f_1 for the test 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 r ⟵ r + 2
9          else
10             if v ≥ a then r ⟵ r + 1
11             if v ≥ b then r ⟵ r + 1
12     return r
13 end
```

Additionally to f_1, two objective functions optimizing non-functional aspects should be present. $f_2(x)$ should minimize the number of expressions in x and $f_3(x)$ should minimize 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 comparator $c_{F,gcd}$ as defined in Definition 18 on page 20, 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.

$$c_{F,gcd}(x_1, x_2) = \begin{cases} -1 \text{ if } (f_1(x_1) < f_1(x_2) \\ 1 \text{ if } (f_1(x_1) > f_1(x_2)) \\ c_{F,pareto}(x_1, x_2) \quad \text{otherwise} \end{cases} \tag{17.16}$$

In principle, Equation 17.16 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 Test Cases

Of further importance is the structure of the test cases. If we simple use two random numbers a and b, their GCD is likely to 1 or 2. Hence, we construct a single test case by first drawing a random number $r \in [10, 100000]$ as lower limit for the GCD and then keep drawing random numbers $a > r, b > r$ until $gcd(a, b) \geq r$. Furthermore, if multiple test 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 test cases after each generation of the applied evolutionary algorithm, the same goes for two subsequent test case sets. Some typical test sets are noted in listing 17.2.

```
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 17.2: Some test cases for the GCD problem.

17.5.2 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 4.9 on page 187). In this section we will elaborate on the different parameters that we have tried out and what results could be observed for these settings.

Configurations

As outlined in Table 17.1, we've tried to solve the GCD problem with rule-based genetic programming with a lot of different settings (60 in total) which we will discuss here.

Table 17.1: Parameters of the RBGP Test Series for the GCD Problem

parameter	shortcut	possible values
EA/random walk	rw	0⇒ EA, 1⇒ random walk
population size	pop	512, 1024, 2048
steady state/generational	ss	0⇒ generational, 1⇒ steady state
convergence prevention	cp	0⇒ off, 1⇒ on
number of test cases	tc	1, 10
changing test cases	ct	0⇒ constant test cases, 1⇒ test cases change each generation

Population Size [pop]

Experiments were performed with three different population sizes *pop*: 512, 1024, and 2048.

Steady State [ss]

The evolutionary algorithms could either be steady state (*ss=1*), meaning that the offspring of a generation has to compete with the already existing individuals in the selection phase, or generational/extinctive (*ss=0*), meaning that only the offspring of a generation will take part in the selection and the parents are discarded. See Section 2.1.3 on page 54 for details on steady state vs. generational evolutionary algorithms.

Convergence Prevention [cp]

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 premature convergence. 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 2.3.5 on page 69), for example. Such methods influence individuals close in objective space and decrease their chance to reproduce.

We decided for a very simple measure that only decreases probability of reproduction of individuals with exactly equal objective functions. In our scenario, this approach is absolutely sufficient since we do not perform a mathematical approximation were individuals close in objective space are often also close in solution space, but an evolution of algorithms, where such relation not necessarily holds. Our method of convergence prevention (*cp*) is to place a filter right after the evaluation of the individuals and before the selection algorithm. Whenever a program enters this filter, we lookup the number n of other programs with equal objective values that entered the filter this generation. The incoming individual is allowed to pass with probability $(1-c)^n$ (and discarded with probability $1-(1-c)^n$), with $c \in [0,1]$ and $c = 0.3$ in our experiments.

This filter has either been applied *cp=1* or not applied *cp=0*.

Number of Test Cases [tc]

We have tested two different numbers of test cases *tc*: 1 and 10. This means that the evaluation of a program was done by computing its objective values either directly with a single test case (*tc=1*) or by computing it on ten different test cases (*tc=10*), setting the final value to be the average.

This was done in order to find out whether overfitting will take place when only a single test case is used and if there is an increase in quality of the solution when ten test cases are performed.

Changing Test Cases [ct]

For the same reason, learning about the influence of overfitting, the utility of a second measure was examined: We either changed the test cases in each generation (*ct=1*) or left them constant throughout all runs (*ct=0*).

If *ct=1*, each individual in the population is evaluated with the same test cases. In each generation however, different test cases are applied. This is a common method to prevent an evolutionary algorithm from "learning" the characteristics of a certain test case and creating programs that only solve this certain problem, used for example in [558]. Such an individual, receiving good objective values during one generation, would probably being penalized with very bad fitness in the next.

General Settings

In the evolutionary algorithms, we always applied a binary tournament selection (see Algorithm 2.16 on page 83) and a prevalence ranking fitness assignment as defined in Algorithm 2.4 on page 68. All runs of all configurations are limited to 501 generations, starting with generation 0 and ending after generation 500.

Comparison to Random Walks [rw]

Last but not least, we found it necessary to compare the genetic programming approach for solving this problem with random walks, so we can be sure whether or not it can provide any advantage in a rugged fitness landscape. Therefore we either used an evolutionary algorithm with the parameters discussed above (*rw=0*) or parallel random walks (*rw=1*). Random walks in this context are principally evolutionary algorithms where neither fitness assignment nor selection are preformed. Hence we can test parameters like *pop*, *tc*, and *ct*, but no convergence prevention (*cp=0*) and also no steady state *ss=0*. The result of these random walks are the best individuals encountered during their course.

Results

The results of the application of the RBGP to the GCD problem are listed in Table 17.2 on the following page. Each of the sixty rows of this table denotes one single test series. The first seven columns of this table specify the settings of the test series as discussed in defined Table 17.1 on page 291. The last two columns contain the evaluation results, which are formatted as follows:

The Correct Solution Ratio [cr]

The column with the headline *cr* contains the ratio of correct solutions $\frac{c}{r}$ obtained by applying the configuration. In other words, the number of runs that yielded a correct, non-overfitted algorithm for computing the GCD c divided by the number of runs performed in total r.

Table 17.2: Results of the RBGP test series on the GCD problem.

rw	cp	ss	ct	tc	pop	cr	c/s/r	sg	rw	cp	ss	ct	tc	pop	cr	c/s/r	sg
0	1	1	1	10	2048	0.98	53/53/54	61.1	0	0	0	1	10	1024	0.06	3/3/53	264.3
0	1	1	0	10	2048	0.98	48/48/49	70.0	0	1	0	0	1	512	0.06	3/22/51	162.1
0	1	0	1	10	2048	0.79	41/41/52	101.1	0	1	1	1	1	512	0.04	2/2/49	102.0
0	1	0	0	10	2048	0.79	39/39/49	111.1	0	0	1	1	10	1024	0.03	2/2/54	328.5
0	1	1	1	10	1024	0.78	42/42/54	125.5	1	0	0	0	1	2048	0.02	1/50/54	120.9
0	1	1	0	10	1024	0.78	41/41/53	129.1	0	0	1	0	1	2048	0.02	1/18/51	245.5
0	1	0	1	10	1024	0.67	37/37/54	199.4	1	0	0	1	1	2048	0.00	0/2/53	101.5
0	1	0	0	10	1024	0.52	27/27/53	196.3	0	0	0	0	1	2048	0.00	0/16/54	146.2
0	1	1	0	10	512	0.49	25/25/51	153.0	0	0	1	0	1	512	0.00	0/6/51	202.0
0	1	1	1	10	512	0.41	21/21/50	231.5	1	0	0	1	1	1024	0.00	0/2/53	209.0
0	1	1	1	1	2048	0.28	14/14/54	107.8	0	0	0	0	1	1024	0.00	0/9/54	257.1
0	1	1	0	1	1024	0.28	15/45/53	100.4	0	0	1	0	1	1024	0.00	0/16/54	277.3
0	1	0	0	1	2048	0.27	14/49/51	85.2	0	0	0	0	1	512	0.00	0/4/50	369.5
0	1	0	1	10	512	0.25	13/13/52	231.5	0	0	0	0	10	512	0.00	1/1/52	492.0
0	1	1	0	1	2048	0.25	13/51/51	36.4	0	0	0	1	1	1024	0.00	0/0/53	–
0	0	0	0	10	2048	0.24	12/12/49	280.6	0	0	0	1	1	2048	0.00	0/0/53	–
0	1	0	0	10	512	0.19	10/10/52	250.8	0	0	0	1	1	512	0.00	0/0/51	–
1	0	0	0	1	1024	0.17	9/41/54	170.0	0	0	0	1	10	512	0.00	0/0/51	–
0	0	1	0	10	2048	0.16	8/8/49	249.9	0	0	1	0	10	512	0.00	0/0/52	–
0	0	0	0	10	1024	0.15	8/8/55	263.0	0	0	1	1	1	1024	0.00	0/0/52	–
0	0	1	1	10	2048	0.13	7/7/52	250.9	0	0	1	1	1	2048	0.00	0/0/54	–
0	0	1	0	10	1024	0.13	7/7/53	272.3	0	0	1	1	1	512	0.00	0/0/49	–
0	1	1	0	1	512	0.12	6/35/51	98.5	0	1	0	1	1	512	0.00	0/0/52	–
0	1	0	1	1	1024	0.10	5/5/52	197.4	1	0	0	0	10	1024	0.00	0/0/55	–
0	0	0	1	10	2048	0.10	5/5/50	237.4	1	0	0	0	10	2048	0.00	0/0/49	–
1	0	0	0	1	512	0.10	5/27/51	259.1	1	0	0	0	10	512	0.00	0/0/52	–
0	1	0	1	1	2048	0.09	5/5/53	46.6	1	0	0	1	1	512	0.00	0/0/51	–
0	1	0	0	1	1024	0.09	5/44/54	138.2	1	0	0	1	10	1024	0.00	0/0/53	–
0	0	1	1	10	512	0.08	4/4/50	320.3	1	0	0	1	10	2048	0.00	0/0/51	–
0	1	1	1	1	1024	0.06	3/3/52	116.0	1	0	0	1	10	512	0.00	0/0/51	–

Correct/Solution/Runs [c/s/r]

The *c/s/r*-column gives a more detailed review of the results. Here you can find the number of runs with correct solutions c in relation with the number of runs with solutions with optimal functional objective values ($f_1 = 0$, whether due to overfitting or not) s and the total number of runs performed with a given configuration r. The relation $r \geq s$ always holds, because there can never be more successful runs than runs in total. Furthermore $s \geq c$ is also always true because s includes the runs which returned individuals with an optimal functional objective value but are overfitted, i. e. will not work with inputs a and b different from those used in their evaluation like the one illustrated in

```
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 17.3: 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_t) ⇒ 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}=1-1
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}=1-1
```

Listing 17.4: An overfitted RBGP solution to the GCP problem.

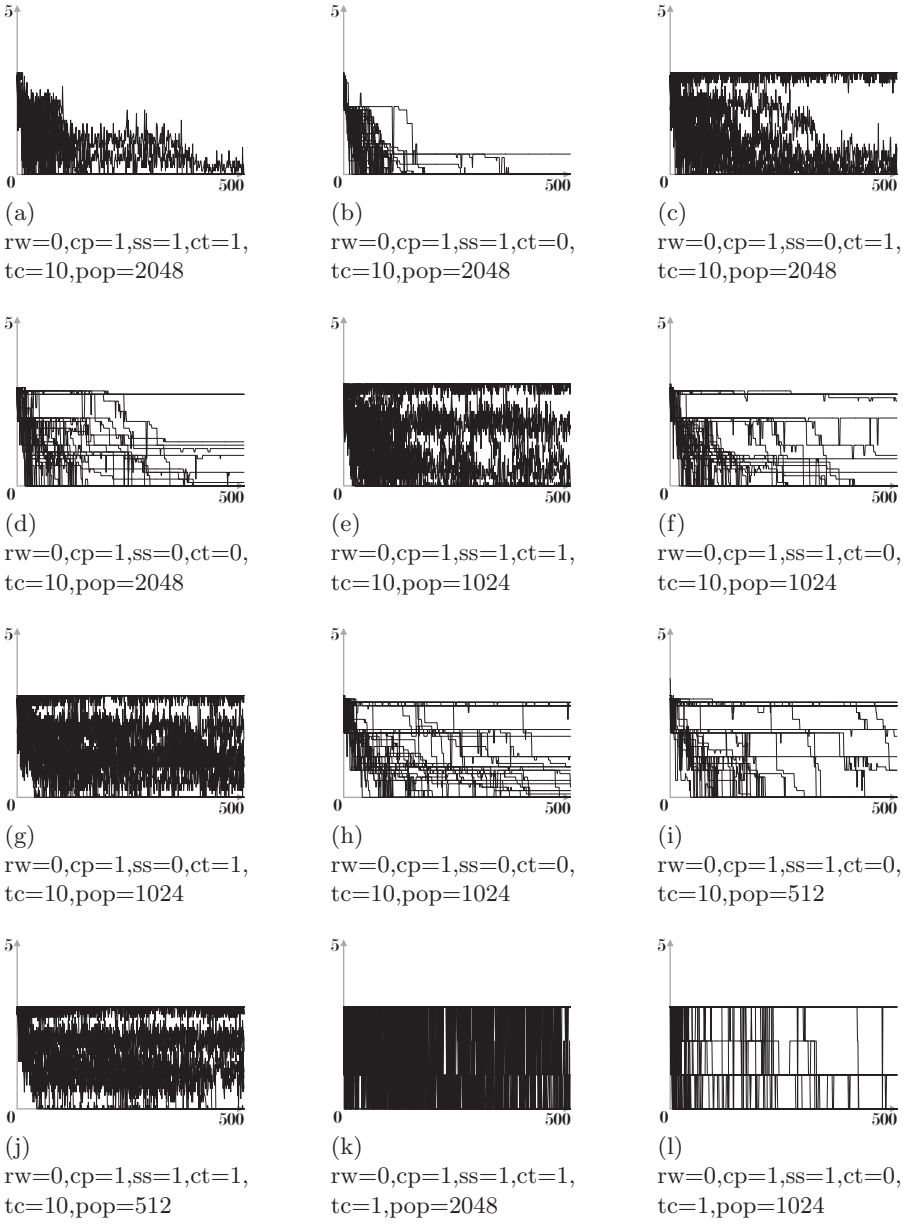
listing 17.4. Finally, c is the number of runs which lead to a correct solution like listing 17.3.

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.

The Average First Success Generation [sg]

The last column, *sg*, contains the average generation where the first solution $x \in \tilde{X}$ with $f_1(x) = 0$ was found. The lower the value in this column, the faster a solution was found. This average includes the runs with overfitted results.

Figure 17.5 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 *cr*-values). The curves are monotone for series with constant test sets (*ct=0*) and jagged for those where the test data changed each generation (*cr=1*).



(a) rw=0,cp=1,ss=1,ct=1, tc=10,pop=2048

(b) rw=0,cp=1,ss=1,ct=0, tc=10,pop=2048

(c) rw=0,cp=1,ss=0,ct=1, tc=10,pop=2048

(d) rw=0,cp=1,ss=0,ct=0, tc=10,pop=2048

(e) rw=0,cp=1,ss=1,ct=1, tc=10,pop=1024

(f) rw=0,cp=1,ss=1,ct=0, tc=10,pop=1024

(g) rw=0,cp=1,ss=0,ct=1, tc=10,pop=1024

(h) rw=0,cp=1,ss=0,ct=0, tc=10,pop=1024

(i) rw=0,cp=1,ss=1,ct=0, tc=10,pop=512

(j) rw=0,cp=1,ss=1,ct=1, tc=10,pop=512

(k) rw=0,cp=1,ss=1,ct=1, tc=1,pop=2048

(l) rw=0,cp=1,ss=1,ct=0, tc=1,pop=1024

Fig. 17.5: The f_1/generation-plots of the best configurations.

Discussion

We have sorted the runs according to their *cr*-values, i. e. the probability of yielding a correct solution, in Table 17.2 and Figure 17.5. In the further text we use this value as quality measure.

Population Size [pop]

The role of the population size is quite obvious, since 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 holds. However, the population size is bar far not the sole factor influencing the performance of genetic programming, because the second-best four series all have population size 1024.

Steady State [ss]

We can also make a very clear statement about the influence of the steady state parameter *ss* had in our experiments. If we again look at the four best runs, 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. The difference seems to be around $1 - \frac{79}{98} \approx 20\%$. Exactly the same relation can be observed with the second-best four configurations (with population size 1024), where those with steady state EAs each have a success rate of 78% and those without have 67% and 52%. Here, the ratio is with $1 - \frac{67}{78} \approx 14\%$ and $1 - \frac{52}{78} = \frac{1}{3} = 33\%$ again somewhat near to 20%.

Convergence Prevention [cp]

Even clearer is the influence of our primitive convergence prevention mechanism – the top 15 test series all have *cp=1*, and even generational tests with a population size of 512 beat steady-state runs a population of 2048 individuals if using convergence prevention. 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.

Number of Test Cases [tc]

The number of test cases has an almost as same as drastic effect: the top ten test series all are based on ten test cases (*tc=10*). We can think of a very simple reason for that which can be observed very well when comparing for example Figure 17.5l with Figure 17.5i. In the twelfth best series, based on only a single test (*tc=1*) and illustrated in Figure 17.5l, only six values (0..5) for the objective function f_1 could occur. The ninth best series depicted in Figure 17.5i on the other hand, had a much broader set of values of f_1 available. Since *tc=10* and the final objective value is the average multiple runs, it had a much smoother curve for f_1 with $51 = |0.0, 0.1, 0.2, \ldots, 4.8, 4.9, 5.0|$ levels.

By using multiple test sets for these runs, we have effectively reduced the ruggedness of the fitness landscape and made it easier for our EA to descend a gradient.

Changing Test Cases [ct]

In the top ten test series, it seems to have no direct relevance if the test cases are constant (*ct=0*) or change every generation (*ct=1*). This does also go the speed of the evolution – the average first success generation *sg* remains roughly constant, regardless if the test data changes or not. The best ten series all use ten test cases (*tc=10*), which seems to prevent overfitting sufficiently on its own. Hence, we furthermore consider only the series with *tc=1*.

At first glance, no benefit of changing test cases can be detected here either – the *cr* values are not really influenced by the *ct* settings. However, there is a difference when we compare their *c/s* ratios. In all runs that find a solution $x \in \tilde{X}$ with $f_1(x) = 0$, this solution is also correct if *ct=1*, i. e. *c=s*. In the test series where *ct=0*, usually only a fraction of the runs that found an individual with perfect functional fitness had indeed found a solution. Here, overfitting takes place and *s¿c* can be usually observed.

Because of the stronger influence of the other settings, we must admit that the parameter *ct* has no substantial influence on the chance of finding a solution to the GCD problem with rule-based genetic programming. This may be rooted in the sufficient prevention of overfitting by using enough constant test cases and it is quite well possible that there are problems, which can benefit from changing test cases. Nevertheless, the chance of having a correct solution when an EA finds a minimum in the fitness landscapes are higher with *ct=1*.

Comparison to Random Walks [rw]

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. Thus, we can safely declare that genetic programming is better than random walks even when solving a task with an extremely rugged fitness landscape as the GCD problem.

One of the reasons for the bad performance of the random walks was that the individuals tended to become unreasonable large. This also increased the amount of time needed for evaluation. The evolutionary algorithm runs usually took about one to ten minutes (depending on the population size) on normal off-the-shelf PCs with approximately 2 GHz processor power, whereas the random walks easily used up to forty minutes.

18

Contests

For most of the problems that can be solved with the aid of computers, multiple different approaches exist. They are often comparably good and their utility in the single cases depends on parameter settings and thus, the experience of the user. Contests provide a stage where students, scientists, and the industry can demonstrate their solutions to specific problems. They help us finding out which techniques are suitable for these 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 [1016, 1017, 1018]. In this chapter we discuss such competitions like the Data-Mining-Cup and how their tasks can be tackled by applying global optimization algorithms.

18.1 Data-Mining-Cup

18.1.1 Introduction

Data Mining

Definition 78 (Data Mining). Data mining[2] can be defined as *the nontrivial extraction of implicit, previously unknown, and potentially useful information from data* [1019] and *the science of extracting useful information from large data sets or databases* [1020].

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, web shops, by administrative

[1] `http://www.robocup.org/` [accessed 2007-07-03] and `http://en.wikipedia.org/wiki/Robocup` [accessed 2007-07-03]

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

and governmental bodies, and in science projects. These data sets are way to large to be incorporated directly into a decision 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 methods and techniques capable to perform this task. It is very closely related to estimation theory in stochastic (discussed in Section 35.6 on page 549) – the simplest digest of data sets is still the arithmetic mean. Data mining is also strongly related to artificial intelligence [866, 958], which includes learning algorithms that can generalize the given information. Some of the most wide spread and most common data mining techniques are:

- (artificial) neural networks (ANN) [1021, 1022],
- support vector machines (SVM) [1023, 1024, 1025, 1026],
- logistic regression [1027],
- decision trees [1028, 1029],
- learning classifier systems as introduced in Chapter 7 on page 211, and
- naïve Bayes Classifiers [1030, 1031].

The Data-Mining-Cup

The Data-Mining-Cup[3] (DMC) has been established in the year 2000 by the *prudsys AG*[4] and the *Technical University of Chemnitz*[5]. 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 18.1 shows the logos of the DMC from 2005 till 2007 obtained from `http://www.data-mining-cup.com/` [accessed 2007-07-03].

18.1.2 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

[3] 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]

[4] `http://www.prudsys.de/` [accessed 2007-07-03]

[5] `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) 2005 (b) 2006 (c) 2007

Fig. 18.1: Some logos of the Data-Mining-Cup.

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 22.1 on page 375. We have also summarized our work for this contest in a technical report [230].

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. Again, when selling tomatoes, 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 tasks of the DMC 2007 Contest, outlined in Section 18.1.2, are 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 techniques that cannot lead to good results in the initial phase instead of wasting working hours in trying them out to avail. 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 18.1.2 on the facing page.

The next step is to apply these 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 or overspecialized and can *only* process the data we are given. Normally however, we are only 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, it is sufficient to derive a solution for a subset of the available data samples, the training data. This solution is then tested on the test set, the remaining samples not used in its creations. 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 18.1.2 on page 306.

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 18.1.2 on page 310.

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:

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:

- Each coupon which is not redeemed costs 1 money unit.
- For each redeemed coupon of type **A**, the retailer gains 3 money units.
- 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.

Figure 18.2 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

$$P(C) = 3 * AA + 6 * BB - 1 * (NA + NB + BA + AB) \tag{18.1}$$

where

- AA is the number of correct assignments for coupon **A**.
- BB is the number of correct assignments for coupon **B**.
- NA is the number of wrong assignments to class **A** from the real class **N**.
- NB is the number of wrong assignments to class **B** from the real class **N**.
- BA is the number of wrong assignments to class **A** from the real class **B**.
- 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 training data sets 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 have some properties which make it especially hard for learning algorithms to find good solutions. Figure 18.3 for example shows three data samples with exactly the same features but different classes. In general,

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	[illegible]																				
83151	0	1	0	0	0	0	1	0	0	1	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	1	0	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	1	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	1	0	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	1	0	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

Fig. 18.2: A few samples from the DMC 2007 training data.

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

Fig. 18.3: DMC 2007 sample data – same features but different classes.

there is some degree of fuzzyness and noise, and clusters belonging to different classes overlap and contain each other. Since the 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 18.1. In general, such a small number of possible feature

Table 18.1: Feature-values in the 2007 DMC training sets.

value	number of ocurences
0	837′119
1	161′936
2	924
3	21

values makes it hard to apply methods 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 18.2, 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 211.

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 18.1 directly as objective function for a genetic algorithm.

Very much like in the Pitt-approach [708, 851, 707] in LCS, the genetic algorithm would base 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 18.2. The three different classes can be 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.

Table 18.2: Feature conditions in the rules.

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)

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 is 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 18.1.

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 18.4 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 is strangely almost always the case for classifier systems that sprung of the artificial evolution. 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

As already discussed in the previous section, we want to evolve the classifier systems directly. Therefore we apply two objective functions:

$$f_1(C) = -P(C) \tag{18.2}$$

$$f_2(C) = \max\{|C|, 3\} \tag{18.3}$$

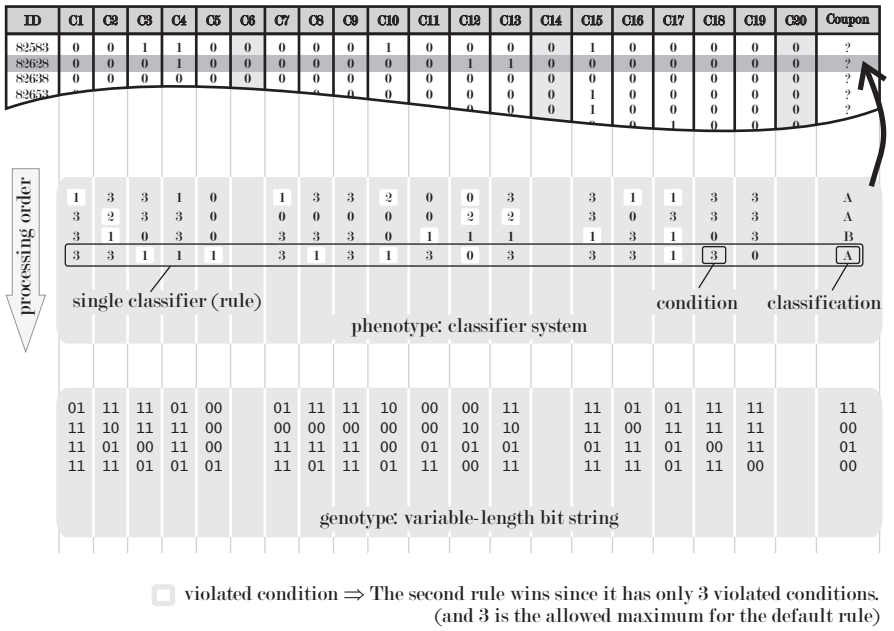


Fig. 18.4: An example classifier for the 2007 DMC.

Here f_1 represents the (negated) profit gained with a classifier system C and $f_2(C)$ is the number of rules in C (cut-off at a size of at 3). Both functions are subject to minimization. Figure 18.5 illustrates the course of the classifier system evolution. Here we have applied a simple elitist genetic algorithm with a population size of 10240 individuals. 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 50000 data samples has been reached. Experiments with 10000 datasets held back and an evolution on the remaining 40000 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 18.6 displays some classifier systems which are members of this set after generation 1000. $C1$ is the smallest non-dominated classifier 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).

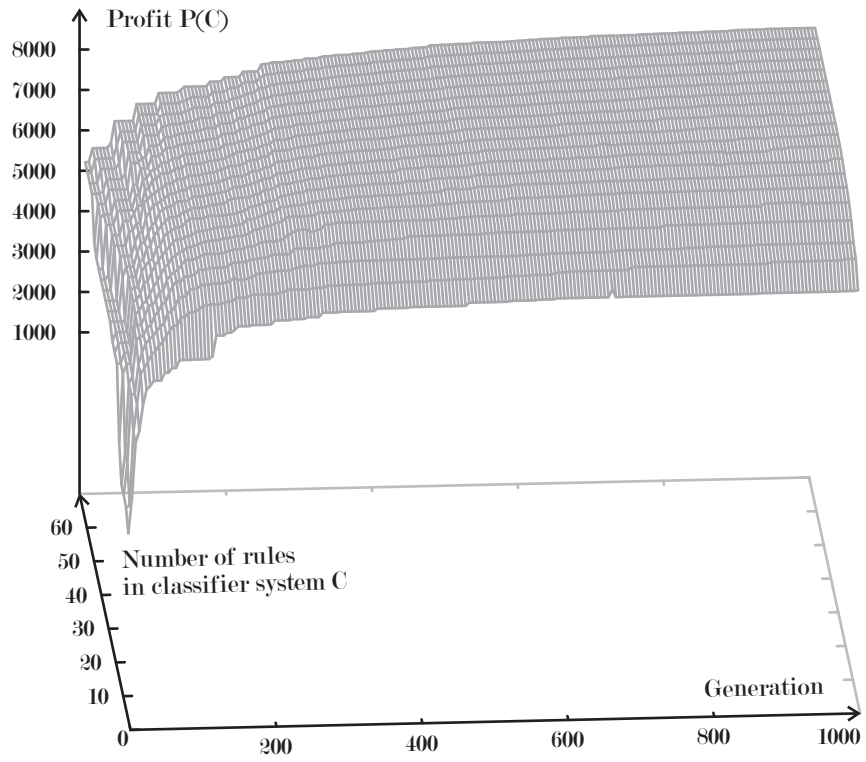


Fig. 18.5: The course of the classifier system evolution.

As shown in Table 18.1 on page 305, most feature values are `0` or `1`, 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 18.3 by changing the meaning of rule `2` from >1 to ≤ 1 (compare with Table 18.2 on page 306).

Table 18.3: Different feature conditions in the rules.

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)

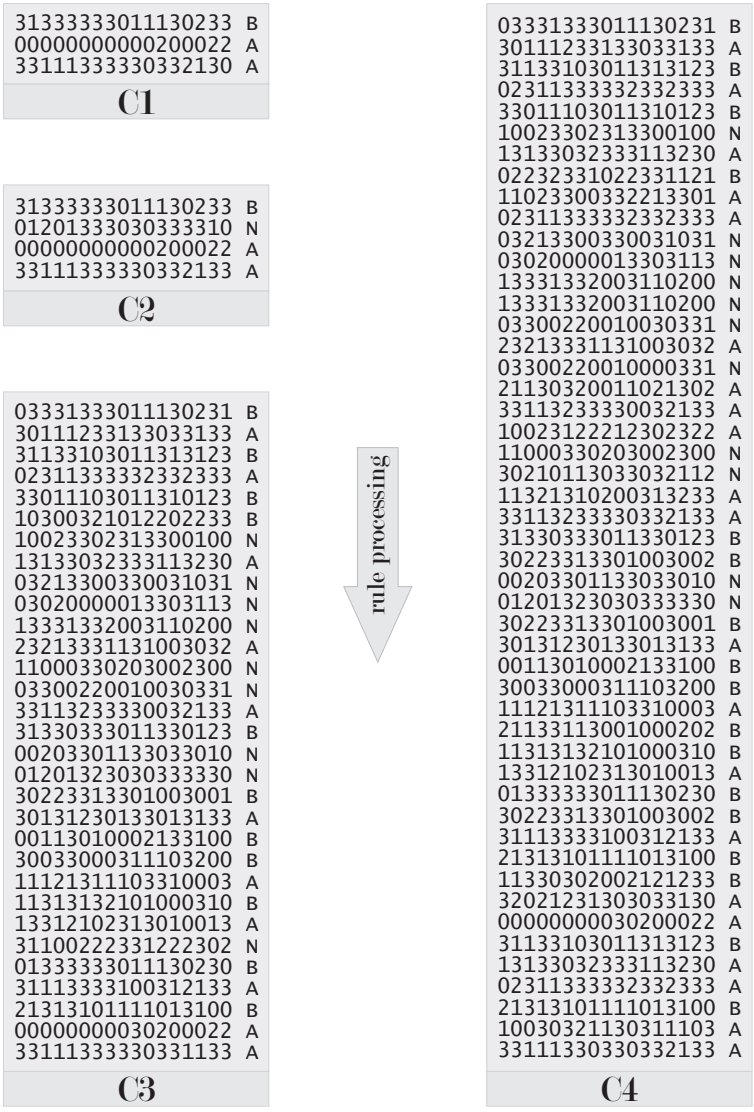


Fig. 18.6: Some Pareto-optimal individuals among the evolved classifier systems.

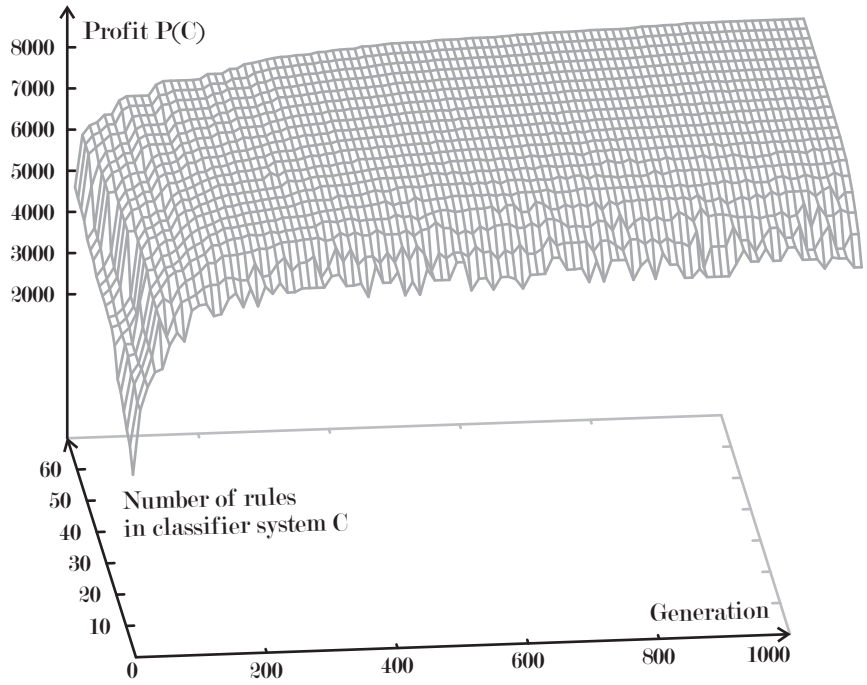


Fig. 18.7: The course of the modified classifier system evolution.

The progress of the evolution depicted in Figure 18.7 exhibits no significant difference to the first one illustrated in Figure 18.5. 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, 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 (only 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.

Remarkably 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 18.1 on page 303).

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 and overspecialization. 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[6]. There exist many sophisticated methods of combining different estimators in order

[6] See our discussion on estimation theory in Section 35.6 on page 549.

to achieve better results [1032]. The original version of such "boosting algorithms", developed by Schapire [1033], 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 [1034] 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 [1035, 1036], as already applied in a very primitive manner, would probably have been the best approach.

18.2 Web Service Challenge

18.2.1 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, regarding for instance accounting, billing, the workflows, and even in the office software. If external vendors, suppliers, or customers change, the interfaces to their IT systems must newly be created or modified too. Hence, the architecture of corporate software has to be built with the anticipation of changes and updates [1037, 1038, 1039].

A SOA is the ideal architecture for such systems [1040, 1041]. Service oriented architectures allow us to modularize business logic and implement in the form of services accessible in a network. Services are building blocks for service processes which represent the workflows of an enterprise. Services 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:

- An application may need various service functionalities.
- Different applications may need the same service functionality.
- 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 [1042] specify the order and types of service parameters and return values. Semantic interface description languages like OWL-S [1043] or WSMO [1044, 1045] annotate these parameters with a meaning. Where 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 an taxonomy we can now deduce that values which are annotated as either `ISBN-10` or `ISBN-13`[7] 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 [1046, 1047, 1048, 1049, 1044, 1045, 1050, 1051]. 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[8] (WS-Challenge, WSC) provides a platform for researchers in the area of web service composition to compare their systems and exchange experiences [1052, 1053, 1054]. 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 has to provide a 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.

18.2.2 The 2006/2007 Semantic Challenge

We have participated both in the Web Service 2006 and 2007 challenges [1055, 1056]. Here we present the system, algorithms and data structures for semantic web service composition that we applied in these challenges. A slightly more thorough discussion of this topic can be found in [1057].

[7] 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].

[8] see `http://www.ws-challenge.org/` [accessed 2007-09-02]

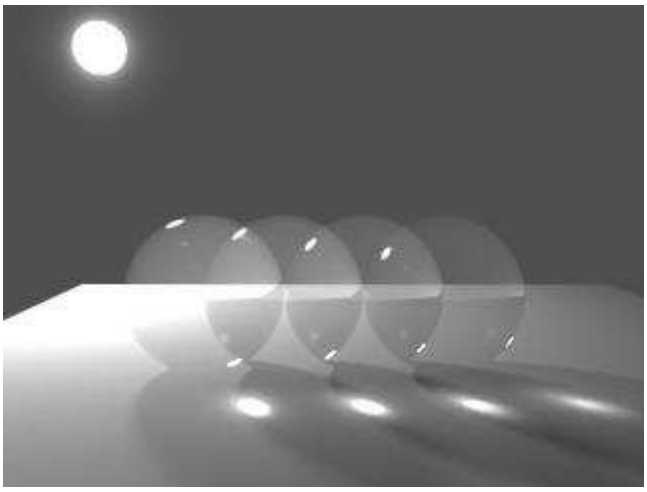


Fig. 18.8: The logo of the Web Service Challenge.

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 79 (subsumes). Two concepts $A, B \in \mathbb{M}$ can be related in one of four possible ways. We define the predicate $subsumes : (\mathbb{M}, \mathbb{M}) \mapsto \{\texttt{true}, \texttt{false}\}$ 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 $A = B$.

The *subsumes* relation is transitive, 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 hold.

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. After its completion, the service will return a set of output parameters as defined in its interface description.

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 discovers a (partially[9]) ordered set of n services $S = \{s_1, s_2, \ldots, s_n\} : s_1, \ldots, s_n \in \mathbb{S}$ that can successively be executed with the accumulated input parameters so that output parameters produced are treated as available input parameters in the next execution step. S can be executed with the available input parameters defined in $R.in$. If it is executed, it produces outputs that are either annotated with exactly the requested concepts $R.out$ or with more specific ones (*covariance*). This is the case if they can be brought into an order $(s_1, s_2, \ldots, s_n)$ in a way that

$$\begin{aligned} isGoal(S) \Leftrightarrow \forall\, A \in s_1.in\ \exists\, B \in R.in : subsumes(A, B) \wedge \\ \forall\, A \in s_i.in, i \in \{2..n\}\ \exists\, B \in R.in \cup s_{i-1}.out \cup \ldots \cup s_1.out : subsumes(A, B) \wedge \\ \forall\, A \in R.out\ \exists\, B \in s_1.out \cup \ldots \cup s_n.out : subsumes(A, B) \end{aligned} \tag{18.1}$$

assuming that $R.in \cap R.out = \emptyset$. With Equation 18.4 we have defined the goal predicate which we can use in any form of informed or uninformed state space search (see Chapter 15 on page 251).

[9] 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.

The Problem Definition

In the 2006 and 2007 WSC, 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 [1058].
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 [1042].
3. A query file containing multiple service composition requests $R_1, R_2, \ldots$ in a made-up XML [1059] format.

These formats are very common and allow the contestants to apply the solutions in real world applications later, or to customize their already existing applications so they can be used as contribution in the competition.

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 18.4. 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.

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 are the restrictions in the challenge tasks:

- There exists at least one solution for each query.
- 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.

These restrictions sure will be removed in future WSCs.

Before we elaborate on the solution itself, let us define the operation $getPromisingServices$ 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).

$$\forall s \in getPromisingServices(A)\ \exists\, B \in s.out : subsumes(A, B) \qquad (18.5)$$

The composition system that we have applied in the 2007 WSC consists of three types of composition algorithms. The search space 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 $\mathbb{S}$. $\tilde{X}$ is then the set of all possible permutations of the elements in such subsets, in other words $\tilde{X} \subseteq \{\forall perm(\xi) : \xi \in \mathcal{P}(\mathbb{S})\}$.

An (Uninformed) Algorithm Based on IDDFS

The first solution approach, Algorithm 18.1, is an iterative deepening depth-first search (IDDFS) algorithm, as discussed in Section 15.2.4 on page 255. It is maintained in our system since it was part of the award winning solution of the WSC'06. 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.

Algorithm 18.1 ($webServiceCompositionIDDFS$) 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 18.1 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 15.3 on page 258, 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

$$A \in wanted(S) \Leftrightarrow (\exists\, s \in S : A \in s.in \vee A \in R.out) \wedge \quad (18.6)$$
$$A \notin R.in \bigcup_{i=1}^{|S|} s_i \quad \ldots (s_i \in S)$$

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.

Algorithm 18.1: $S = webServiceCompositionIDDFS(R)$

Input: R the composition request
Data: $maxDepth$, $depth$ the maximum and the current search depth
Data: in, out current parameter sets
Data: $composition$,$comp$ the current compositions
Data: A,B,C,D,E some concepts
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  end
8  dl_dfs_wsc(in, out, composition, depth)
9  begin
10     foreach A ∈ out do
11         foreach s ∈ getPromisingServices(A) do
12             wanted ⟵ out
13             foreach B ∈ wanted do
14                 if ∃ C ∈ s.out : subsumes(B, C) then
                   wanted ⟵ wanted \ {B}
15             foreach D ∈ s.in do
16                 if ∄ E ∈ in : subsumes(D, E) then
                   wanted ⟵ wanted ∪ {D}
17             comp ⟵ s ⊕ composition
18             if wanted = ∅ then
19                 return comp
20             else
21                 if depth < maxDepth then
                   comp ⟵ dl_dfs_wsc(in, wanted, comp, depth + 1)
22                 if comp ≠ ∅ then return comp
23     return ∅
24 end
```

$$eliminated(S) = \left[R.out \cup \left(\bigcup_{i=1}^{|S|} s_i.in \right) \right] \setminus wanted(S) \tag{18.7}$$

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.

$$known(S) = R.in \cup \bigcup_{i=1}^{|S|} s_i.out \tag{18.8}$$

Instead of using these sets to build a heuristic, we can derive a comparator function c_{wsc} directly (see Section 15.3.1 on page 259). This comparator function has the advantage that we also can apply randomized optimization methods like evolutionary algorithms based on it.

Algorithm 18.2: $r = c_{wsc}(S_1, S_2)$

Input: $S_1, S_2 \in \tilde{X}$ two composition candidates
Data: i_1, i_2, e_1, e_2 some variables
Output: $r \in \mathbb{Z}$ indicating whether S_1 ($r < 0$) or S_2 ($r > 0$) should be expanded next

```
1  begin
2      i1 ⟵ |wanted(S1)|
3      i2 ⟵ |wanted(S2)|
4      if i1 ≤ 0 then
5          if i2 ≤ 0 then return |S1| − |S2|
6          else return −1
7      if i2 ≤ 0 then return 1
8      e1 ⟵ |eliminated(S1)|
9      e2 ⟵ |eliminated(S2)|
10     if e1 > e2 then return −1
11     else
12         if e1 < e2 then return 1
13     if i1 > i2 then return −1
14     else
15         if i1 < i2 then return 1
16     if |S1| ≠ |S2| then return |S1| − |S2|
17     return |known(S1)| − |known(S2)|
18 end
```

Algorithm 18.2 defines c_{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. c_{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.

The first thing it does is comparing 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. If the comparator has not returned a value yet, it means that both candidates still have wanted concepts. For us, it was

surprising that it is better to use the number of already satisfied concepts as next comparison criterion instead of the number of remaining unsatisfied concepts. However, if we do so, the search algorithms perform significantly faster. Only if both composition candidates have the same number of satisfied parameters, we again compare the wanted concepts. If their numbers are also equal, we prefer the shorter composition candidate. If the compositions are even of the same length, we finally base the decision of the total number of known concepts.

The form of this interesting 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.

The interesting thing that we experienced in our experiments is that it is not a good idea to decide on the utility of a solution candidate with

In order to apply pure greedy search, we still need to specify the *expand* operator computing the set of possible offspring that can be derived from a given individual. In Algorithm 18.1, we have realized it implicitly. Additionally, we can also define the *isGoal* predicate on basis of the *wanted* function:

$$expand(S) \equiv s \oplus S \; \forall \, s, A : \; s \in getPromisingServices(A) \wedge A \in wanted(S) \tag{18.9}$$

$$isGoal(S) \equiv wanted(S) = \emptyset \tag{18.10}$$

With these definitions, we can now employ plain greedy search as defined in Algorithm 15.5 on page 259.

A Genetic Approach

In order to use a genetic algorithm to breed web service compositions, we first need to define a proper genome 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 3.4.2 on page 126). Because of this well-known string form, we also could apply 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 $mutate_1$) or adds a promising service to S (as done in $mutate_2$). Using the adjustable variable σ as a threshold we can tell the search whether it should prefer growing or shrinking the solution candidates.

$$mutate_1(S) \equiv \begin{cases} \{s_2, s_3, \ldots, s_{|S|}\} & \text{if } |S| > 1 \\ S & \text{otherwise} \end{cases} \tag{18.11}$$

$$mutate_2(S) \equiv s \oplus S : \; s \in getPromisingServices(A) \wedge A \in wanted(S) \tag{18.12}$$

$$mutate(S) \equiv \begin{cases} mutate_1(S) & \text{if } random_u() > \sigma \\ mutate_2(S) & \text{otherwise} \end{cases} \tag{18.13}$$

A new *create* operation for building the initial random configurations can be defined as a sequence of $mutate_2$ invocations of random length. Initially, $mutate_2(\emptyset)$ will return a composition consisting of a single service that satisfies at least one parameter in $R.out$. We iteratively apply $mutate_2$ 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 c_{wsc} as specified as for the greedy search in Algorithm 18.2 on page 319. It combines multiple objectives, putting pressure towards the direction of

- compositions which are complete,
- small compositions,
- compositions that resolve many unknown parameters, and
- 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[10].

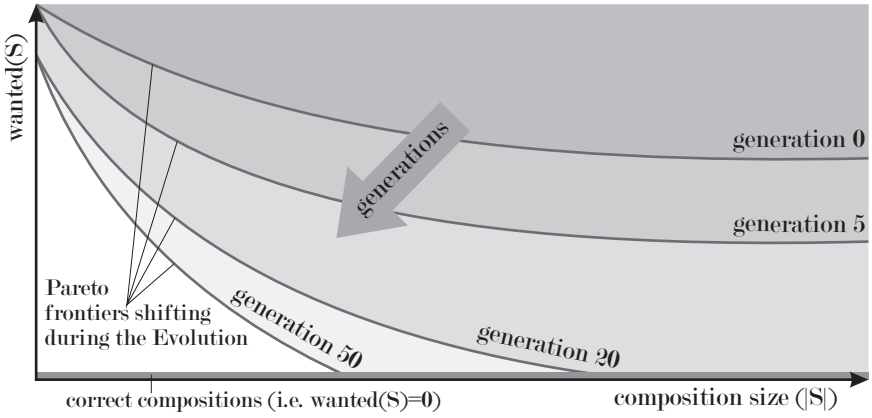


Fig. 18.9: A sketch of the Pareto front in the genetic composition algorithm.

Figure 18.9 visualizes the multi-objective 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

[10] See Section 1.3.4 on page 19 for a detailed discussion on the drawbacks of pure Pareto optimization.

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 Figure 18.9, 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 c_{wsc} has proven to be more efficient in focusing the evolution on this part of the search space. The genetic algorithm based on it is superior in performance and hence, is used in our experiments.

Experimental Results

In Table 18.4 we illustrate the times that the different algorithms introduced in this section needed to perform composition tasks of different complexity[11]. We have repeated the experiments multiple times on an off-the-shelf PC[12] and noted the mean values. The times themselves are not so important, rather are the proportions and relations between them.

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 it didn't happen in our experiments) could lead to a very long computation time in the worst case. 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.

[11] The test sets used here are available at `http://www.it-weise.de/documents/files/BWG2007WSC_software.zip` [accessed January 4, 2008]. Well, at least partly, I've accidentally deleted set 12 and 13. Sorry.

[12] 2 GHz, Pentium IV single core with Hyper-Threading, 1 GiB RAM, Windows XP, Java 1.6.0._03-b05

Table 18.4: Experimental results for the web service composers.

Test	Depth of Solution	No. of Concepts	No. of Services	IDDFS (ms)	Greedy (ms)	GA (ms)
1	5	56210	1000	241	34	376
2	12	56210	1000	-	51	1011
3	10	58254	10000	-	46	1069
4	15	58254	2000	-	36	974
5	30	58254	4000	-	70	6870
6	40	58254	8000	-	63	24117
7	1	1590	118	≤16	≤16	290
8.1	2	15540	4480	≤16	≤16	164
8.2	2	15540	4480	≤16	≤16	164
8.3	2	15540	4480	≤16	≤16	164
8.4	2	15540	4480	≤16	≤16	234
8.5	3	15540	4480	≤16	≤16	224
8.6	3	15540	4480	≤16	≤16	297
8.7	4	15540	4480	18	24	283
8.8	3	15540	4480	≤16	≤16	229
8.9	2	15540	4480	≤16	≤16	167
11.1	8	10890	4000	-	31	625
11.3	2	10890	4000	-	21	167
11.5	4	10890	4000	22021	≤16	281
12.1	5	43680	2000	200320	≤16	500
12.3	7	43680	2000	99	31	375
13	6	43680	2000	250	32	422

The genetic algorithm (population site 1024) 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 [1056] of our 2006 semantic composition system [1055]. The architecture of this composer, as illustrated in Figure 18.10, is designed in a very general way, making it not only a challenge contribution but also part of the ADDO web service brokering system [1048, 1046, 1047]: 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) [1043] or similar languages is used. Hence, this contest application is indeed also a real-world application.

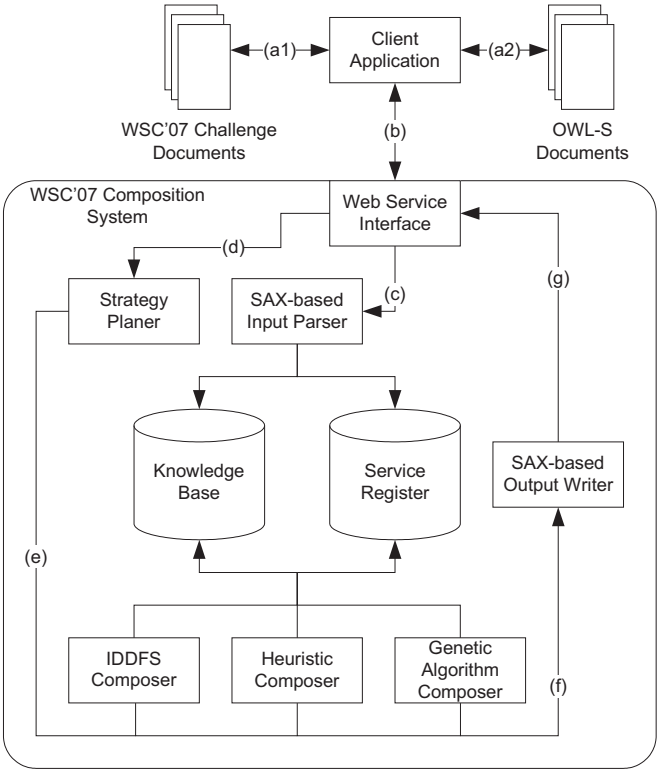


Fig. 18.10: The WSC 2007 Composition System of Bleul and Weise.

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 [1060] 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 *getPromisingServices* 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.

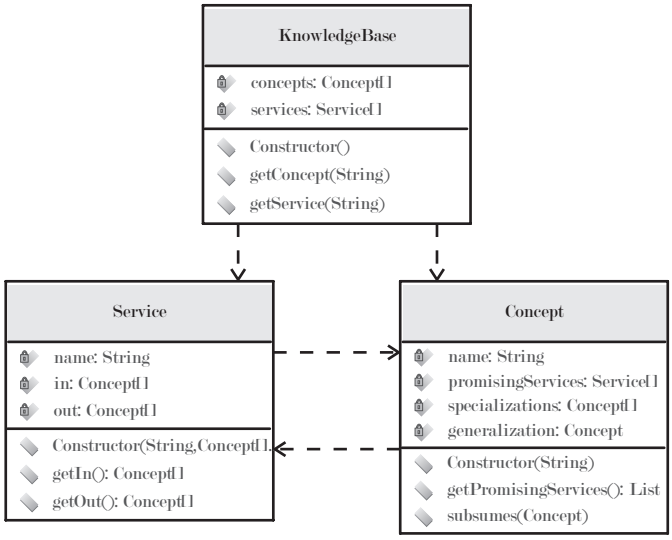


Fig. 18.11: The Knowledge Base and Service Registry of our Composition System.

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 18.11, 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 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 `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 meta-heuristic 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 genetic 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 [1061] processes. In this case, heuristic search will most probably become insufficient but genetic algorithms and genetic programming [11, 1062] will still be able to deliver good results.

In this report, we have discussed semantic composition in general way. The algorithms introduced here are not limited to semantic web service composition. Other applications, like the composition of program modules are also interesting. From general specifications what functionality is needed, a compiler could (in certain limits, of course) deduce the correct modules and code to be linked, using the same methods we use for building service processes.

19

Real-World Applications

In this chapter we will explore real-world applications of global optimization techniques. Some of the areas where global optimization algorithms can easily be applied in a productive fashion, aiding scientists and engineers with their work, are discussed here.

19.1 Symbolic Regression

In statistics, regression analysis examines the unknown relation $\varphi :\in \mathbb{R}^m \mapsto \in \mathbb{R}$ of a dependent variable $y \in \mathbb{R}$ to specified independent variables $\mathbf{x} \in \mathbb{R}^m$. Since ϖ is not known, the goal is to find a reasonable good approximation $f^\star$.

Definition 80 (Regression). Regression[1] [1063, 1064, 1065, 1066] 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 $f^\star : \mathbb{R}^m \mapsto \mathbb{R}$ that relates the m independent variables (subsumed in the vector $\mathbf{x}$ to one dependent variable $y \approx f^\star(\mathbf{x})$. The function $f^\star$ is the best estimator chosen from a set F of candidate functions $f : \mathbb{R}^m \mapsto \mathbb{R}$.

Regression is strongly related to the estimation theory outlined in Section 35.6 on page 549. 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 81 (Symbolic Regression). Symbolic regression [11, 506, 507, 1067, 1068, 1069, 1070, 1071, 508] is the most general case of regression. It is not limited to determining the optimal values for the set of parameters of

[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]

a certain array of functions. Instead, regression functions can be constructed by combining elements of a set of mathematical expressions, variables and constants.

19.1.1 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 19.1, 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 $\mathbf{x}$.

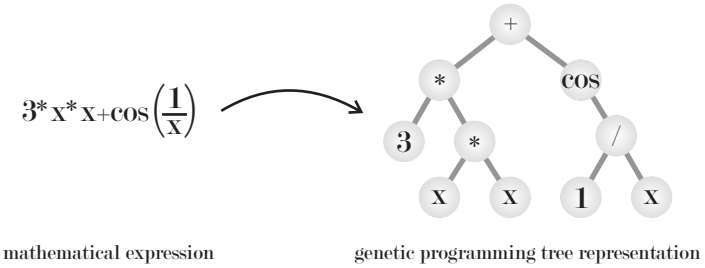


Fig. 19.1: An example genotype of symbolic regression of with $x = \mathbf{x} \in \mathbb{R}^1$.

The set F of functions f that can possible be evolved is limited by the set of expressions E available to the evolutionary process.

$$E = \{+, -, *, /, \exp, \ln, \sin, \cos, \max, \min, \ldots\} \tag{19.1}$$

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 $\frac{x}{x+x} + \frac{\ln x * x}{x}$. The evolution of such artificial constants however takes rather long.

Koza has therefore introduced the concept of ephemeral random constants [11].

Definition 82 (Ephemeral Random Constants). If a new individual is created and a leaf in its expression-tree is chosen to be a 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 are 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 177 on page 566):

$$c_{new} = random_n(c_{old}, \sigma^2) \tag{19.2}$$

$$\sigma^2 = e^{-random_u(0,10)} * |c_old| \tag{19.3}$$

Notice that the other reproduction operators for tree genomes have been discussed in detail in Section 4.3 on page 145.

19.1.2 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 35.6.1 on page 552.

Again, we are given a finite set of sample data S containing n 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 measurement errors η_i, see Equation 35.209 on page 552). Furthermore, we can access a (possible infinite large) set F of functions $f : \mathbb{R}^m \mapsto \mathbb{R} \in F$ which are possible estimators of φ. For the inputs $\mathbf{x}_i$, the results of these functions f deviate by the estimation error (see Definition 165 on page 551) from the y_i.

$$y_i = \varphi(\mathbf{x}_i) + \eta_i \; \forall 0 < i \leq n \tag{19.4}$$

$$y_i = f(\mathbf{x}_i) + \epsilon_i(f) \; \forall f \in F, \; 0 < i \leq n \tag{19.5}$$

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 S and thus, resemble the function φ, closely. Let us call this "driving force" quality function.

Definition 83 (Quality Function). The quality function $q(f, S)$ defines the quality of the approximation of φ by a function f. The smaller the value of the quality function is, the more precisely is the approximation of φ by f in the context of the sample data S.

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 35.210, Equation 35.211, and Equation 35.212 on page 552), we have shown in Section 35.6.1 that the best estimators minimize the mean square error MSE (see Equation 35.226 on page 555, Definition 172 on page 556 and Definition 168 on page 551).

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.

$$q_{\sigma\neq 0}(f,S) = \sum_{i=1}^{n=|S|} (y_i - f(\mathbf{x}_i))^2 \tag{19.6}$$

While this is normally true, there is one exception to the rule. If the values y_i are no measurements but direct results from φ. A common example for this situation is if we apply symbolic regression in order to discover functional identities [631, 670, 11] (see also Section 19.1.3). Different from normal regression analysis or estimation, we here know φ exactly and want to find another function $f^\star$ that is another, equivalent form of φ. Therefore, we will use φ to create sample data set S beforehand, carefully selecting characteristic points $\mathbf{x}_i$.

Then, the measurement errors η_i all become zero. If we would still regard them as normally distributed, their variance σ^2 would be zero.

The proof for the statement that minimizing the square errors maximizes the likelihood is based on the transition from Equation 35.221 to Equation 35.222 on page 554 where we cut divisions by σ^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 used in these circumstances are the sums of the absolute values of the estimation errors:

$$q_{\sigma=0}(f,S) = \sum_{i=1}^{n=|S|} |y_i - f(\mathbf{x}_i)| \tag{19.7}$$

19.1.3 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 functions f. In symbolic regression by genetic programming, the problem of code bloat (discussed in Section 4.11.3 on page 199) 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 just some sort of misfit decision table. This phenomenon is called overfitting and is initially discussed in Section 1.4.6 on page 27.

Let us for example assume we want to find a function similar to Equation 19.8. Of course, we would hope to find something like Equation 19.9.

$$y = \varphi(x) = x^2 + 2x + 1 \tag{19.8}$$

$$y = f_1^\star(x) = (x+1)^2 = (x+1)(x+1) \tag{19.9}$$

For simplicity, we choose randomly the nine sample data points listed in Table 19.1.

Table 19.1: Sample Data $S = \{(x_i, y_i) : i = 1 \ldots 9\}$ for Equation 19.8

i	x_i	$y_i = \varphi(x_i)$	$f_2^\star(x_i)$
1	−5	16	15.59
2	−4.9	15.21	15.40
3	0.1	1.21	1.11
4	2.9	15.21	15.61
5	3	16	16
6	3.1	16.81	16.48
7	4.9	34.81	34.54
8	5	36	36.02
9	5.1	37.21	37.56

As result of the symbolic regression we may obtain something like Equation 19.10, outlined in Figure 19.2, which represents the data points quite precisely but has nothing to do with the original form of our equation.

$f_2^\star(x) =$ (((((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))))) (19.10)

We obtained both functions $f_1^\star$ (in its second form) and $f_2^\star$ using the symbolic regression applet of Hannes Planatscher which can be found at

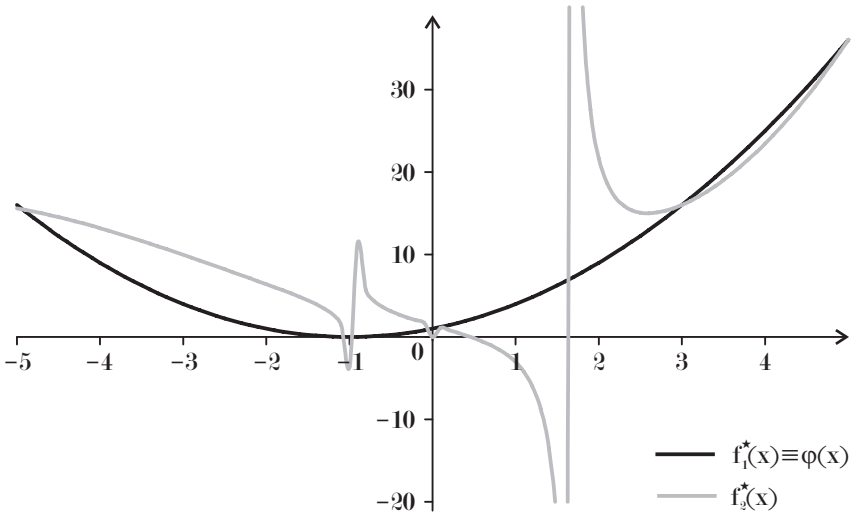


Fig. 19.2: $\varphi(x)$, the evolved $f_1^\star(x) \equiv \varphi(x)$, and $f_2^\star(x)$.

`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 $f_2^\star$.

Indeed there are some factors which further the evolution of such eyesores:

- 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.
- 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.
- 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 is often yielded.
- Allowing functions of large depth and putting low pressure against bloat (see Section 4.11.3 on page 199) 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

[4] Another good applet for symbolic regression can be found at `http://alphard.ethz.ch/gerber/approx/default.html` [accessed 2007-07-03]

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.

19.1.4 Limits of Symbolic Regression

In most cases, we cannot obtain an optimal approximation of f, especially if the function f that produced the data cannot be represented by the basic expressions available to the regression process. One of these cases has already been discussed before: if f has no closed arithmetical expression. Another possibility is that our regression method tries to generate a polynomial that approximates the f, but f does contain different expressions like sin or e^x or polynomials of an order higher than available. Another case is that the values y_i are not results computed by function directly but could be for example measurements taken from some physical entity and we want to use regression to determine the interrelations between this entity and some others. Then, the measurements will be biased by noise and systematic measurement errors. So there exist multiple situations where $q(f^\star, S)$ is will be greater than zero after a successful regression.

20

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. Here we often begin to obtain solutions that are on par or at least comparable with those obtained by the traditional methodologies [543, 591]. On the other hand, they differ from toy problems because they are not intended to be used as demonstration example or benchmark but are first steps into a new field of application of genetic programming.

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 evolutionary algorithms and other optimization techniques can be tested.

20.1 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.

20.1.1 Aggregation Protocols

Definition 84 (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 feature 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

[1] `http://en.wikipedia.org/wiki/Aggregate_function` [accessed 2007-07-03]
[2] `http://en.wikipedia.org/wiki/Aggregate_data` [accessed 2007-07-03]

in databases and knowledge bases [1072, 1073, 1074, 1075, 1076, 1077, 1078], 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 37.2.2) and sensor networks (discussed in Section 37.2.2), a whole new type of aggregation came into existence in the form of aggregation protocols. They are a key functional building block for such systems by providing the distributed components with access to global information including network size, average load, mean uptime, location and description of hotspots, and so on [1079, 1080]. 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.

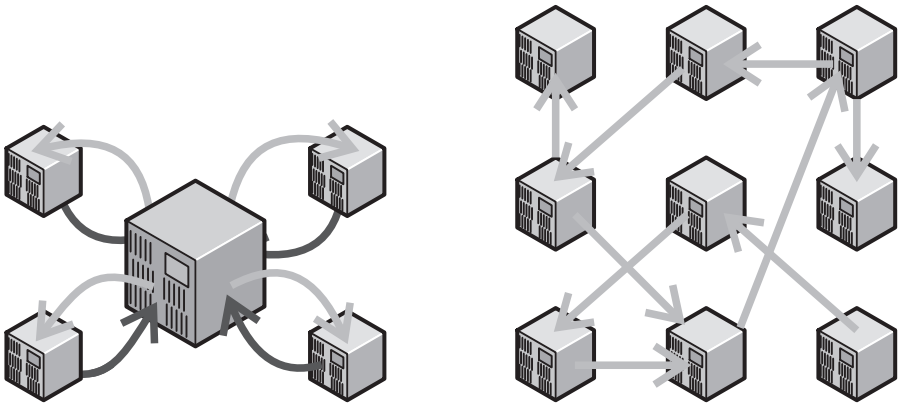
In aggregation protocols, the data vector $\mathbf{x}$ is no longer locally available but its elements are spread all over the network. When now computing the aggregate, we cannot just evaluate α. Instead, some form of data exchange must be performed by the nodes. This exchange can happen in two ways: 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 Figure 20.1a. A proactive aggregation protocol as outlined in Figure 20.1b one 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 refinement of estimates of the wanted value. Notice that the trivial solution would be that all nodes send their information to all other nodes – this is avoided generally and the data is disseminated step by step as part of the estimates.

Gossip-Based Aggregation

Jelasity, Montresor and Babaoglu [1080] propose a simple yet efficient type of proactive aggregation protocols [1081]. 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 $\mathbf{x} = (x_p, x_q, \ldots)$.

The nodes hold local states s (possible containing x) which they can exchange via communication. Therefore, each nodes knows picks its communication partners with the *getNeighbor()* method.



(a) reactive aggregation (b) proactive aggregation

Fig. 20.1: The two basic forms of aggregation protocols.

The skeleton of the gossip-based aggregation protocol is specified in Algorithm 20.1 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 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

Example – Distributed Average

Assume that we have built a sensor network measuring the temperature as illustrated in Figure 20.2. Each of our sensor nodes is equipped with a little display visible to the public. 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 and the estimate of another node, returns the mean of its inputs.

$$update_{avg}(s_p, s_q) = \frac{s_p + s_q}{2} \tag{20.1}$$

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

Algorithm 20.1: *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

```
   // active Thread
 1 begin
 2     while true do
 3         do exactly once in every δ units at a randomly picked time:
 4         q ⟵ getNeighbor()
 5         sendTo(q, s_p)
 6         s_q ⟵ receiveFrom(q)
 7         s_p ⟵ update(s_p, s_q)
 8 end
   // passive Thread
 9 begin
10     while true do
11         s_r ⟵ receiveAny()
12         sendTo(getSender(s_r), s_p)
13         s_p ⟵ update(s_p, s_r)
14 end
```

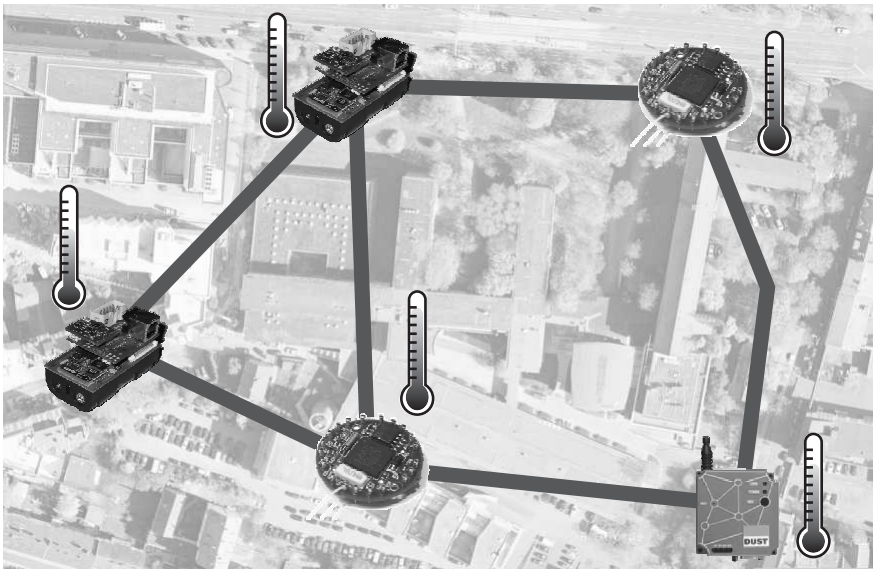
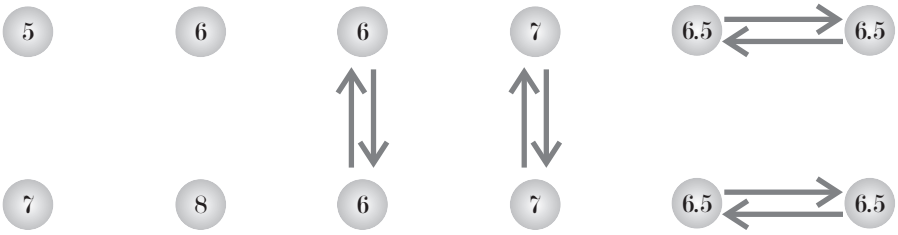


Fig. 20.2: An example sensor network measuring the temperature.

thus also the mean – of both states remains constant. Their variance, however, becomes 0 and so the overall variance in the network gradually decreases.



(a) initial state (b) after step 1 (c) after step 2

Fig. 20.3: 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 Figure 20.3a. The arithmetic mean here is

$$\frac{5+6+7+8}{4} = \frac{13}{2} = 6.5 \tag{20.2}$$

The initial variance is

$$\frac{(5-6.5)^2 + (6-6.5)^2 + (7-6.5)^2 + (8-6.5)^2}{4} = \frac{5}{4} \tag{20.3}$$

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 Figure 20.3b). Now the average of all estimates is still

$$\frac{6+6+7+7}{4} = 6.5 \tag{20.4}$$

but the variance has been reduced to

$$\frac{(6-6.5)^2 + (6-6.5)^2 + (7-6.5)^2 + (7-6.5)^2}{4} = 1 \tag{20.5}$$

After the second protocol step, outlined in Figure 20.3c, 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:

- **Minimum** and **Maximum**. The minimum and maximum of a value in the network can be computed by setting $update_{min}(s_p, s_q) = \min\{s_p, s_q\}$ and $update_{max}(s_p, s_q) = \max\{s_p, s_q\}$ respectively.

- **Count**. The number of nodes in a network 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 $\frac{1+0+0+\ldots}{n} = \frac{1}{n}$. Its inverse $\frac{1}{\frac{1}{n}} = n$ then corresponds to the number of nodes in the network.
- **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 in the network n simultaneously and multiplying both with each other: $n\overline{x} = \sum x$.
- **Variance**. As declared in Equation 35.59 on page 522, 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 $var \approx \overline{x^2} - \overline{x}^2$ 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 37.2.2 that are very challenging for distributed applications and lead to an inherent volatility of $\mathbf{x}$. The default approach to handle unstable data is to periodically restart the aggregation protocols [1080]. In our research we were able to provide alternative aggregation protocols capable of dealing with dynamically changing data. This approach is discussed in Section 20.1 on page 337.

20.1.2 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 19.1 on page 329 [548]. 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 where m is not a constant, i. e. α will return exact results for $m = 1, 2, 3, \ldots$. In Section 35.6.1 on page 555 we were able to 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 19.6 on page 332), so the value of m does not affect the correctness of the symbolic regression approach. Deriving aggregation functions for distributed systems however exceeds the capabilities of normal symbolic regression. Here, $m = |\mathcal{N}|$ is the number of nodes in a network $\mathcal{N}$. Each of the m nodes holds exactly one element of the data vector. Hence, 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 subject to our research here.

There are three use cases for such an automated aggregation protocol generation:

- 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.
- 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.
- The most probable use case is that we know how to compute the aggregate locally with a given 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 simple sum them up. The correct solution described in Section 20.1.1 on page 339 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.

20.1.3 Network Model and Simulation

For gossip-based aggregation protocols, [1080] supposes 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 20.1 on page 340, 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 $\mathcal{N}$ consisting of $m = 4$ nodes p, q, r, and t for example. 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, Montresor and Babaoglu have shown that different forms of pair selection influence the convergence speed of the protocol [1080]. However, 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 [1080], and, more importantly, will produce different outcomes in each test and make comparing the different evolved protocols complicated (as discussed in Section 1.5 on page 28). Therefore, choosing a deterministic selection scheme seems to be the best approach. Perfect matching according to [1080] 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.

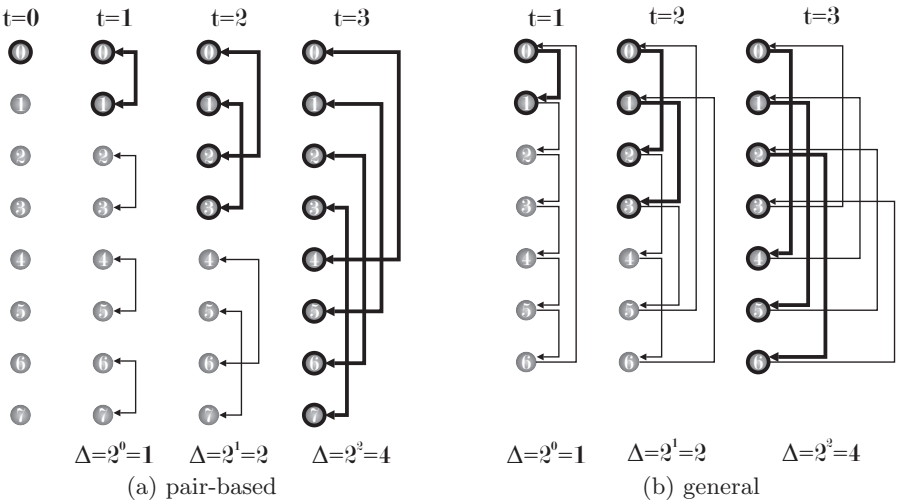


Fig. 20.4: Optimal data dissemination strategies.

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 $\mathcal{N} = m = 2^d$ 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 is the id-number of the node. This setup is optimal, as you can see in Figure 20.4a. 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 as

$$s_3(t=2) = \frac{s_3(t=1) + s_1(t=1)}{2} + \frac{\frac{s_3(t=0)+s_2(t=0)}{2} + \frac{s_0(t=0)+s_1(t=0)}{2}}{2} \quad (20.6)$$

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 restriction. 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 Figure 20.4b. This general communication rule abandons the restriction of strict pair-based data exchange but leaves any other feature of the aggregation protocols, like 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.

20.1.4 Node Model and Simulation

As important as modeling the network is the model of the nodes it consists of. In Figure 20.5, 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 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 is on one hand the local approximation of the aggregate value and on the other hand 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 in [1080], 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 approximations of aggregate values at positions $1 \leq i \leq n$. 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

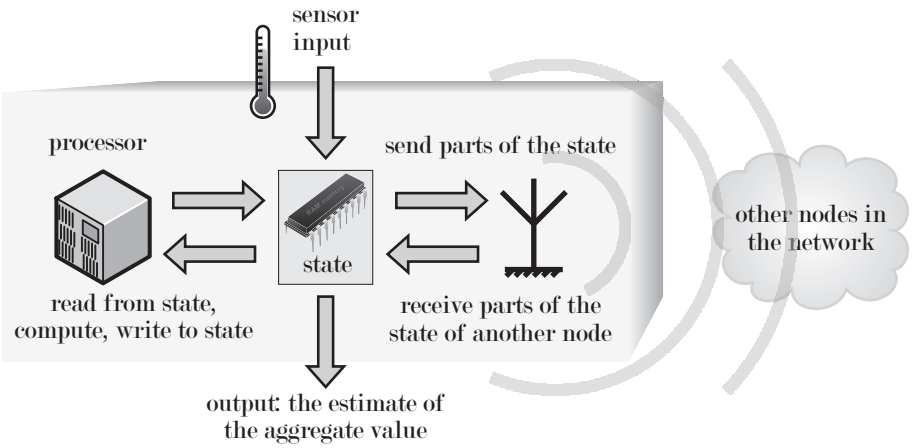


Fig. 20.5: The model of a node capable to execute a proactive aggregation protocol.

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 $\mathbf{s}[i]$ being the i^{th} component of the vector:

$$\mathbf{s}_p[r_j] \longleftarrow \mathbf{s}_q[e_j] \; \forall \; j = 1 \ldots |r| \tag{20.7}$$

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 [1080]. 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

$$\mathbf{s}_p(t)[I] \longleftarrow \mathit{getInput}(p,t) \tag{20.8}$$

The function $\mathit{getInput}(p,t)$ returns the input value of node p at time step t. With this definition, the state vectors $\mathbf{s}$ become time-dependent, written as $\mathbf{s}(t)$. Finally, *update* is now designed as a map $\mathbb{R}^n \mapsto \mathbb{R}^n$ to return the new state vector.

$$\mathbf{s}_p(t+1) = update(\mathbf{s}_p(t)) \tag{20.9}$$

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.

$$S(t) = (\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_m) \tag{20.10}$$

$$S_{j,k} = \mathbf{s}_k[j] \tag{20.11}$$

This notation is used in Algorithm 20.3.

In Algorithm 20.2 we specify how the model definitions that we have 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 20.1.3.

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.

20.1.5 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 20.1.2, we already know either 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 $S_{O,\star} \equiv \mathbf{s}_k[O] \; \forall \; k \in 1 \ldots n$ 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.

$$y(t) = \alpha\left(\left(getInput(1,t),\ldots,getInput(m,t)\right)'\right) \tag{20.12}$$

$$\varepsilon(k,t) = y(t) - S_{O,k}(t) = y(t) - \mathbf{s}_k[O] \tag{20.13}$$

Algorithm 20.2: $simulateNetwork(m, T)$

Input: m the number of nodes in the simulation
Input: T the maximum number of simulation steps
Input: Implicitly: *update* the update function
Input: Implicitly: I the index for the input values
Input: Implicitly: O the index for the output values
Input: Implicitly: e the index list for the send values
Input: Implicitly: r the index list for the receive values
Data: d communication step base according to 20.4b on page 344
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) ⟵ new n × m Matrix
       // initialize with local values
4      S(0)_{⋆,k} ⟵ getInput(k, 0)
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 20.4b on page 344
9          k ⟵ 1
10         while k ≤ m do
11             p ⟵ (k + Δ) mod m
12             S(t)_{r_j,p} ⟵ S(t − 1)_{e_j,k} ∀ j = 1 . . . |r|
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
```

We have already argued that the mean square error is an appropriate quality function for symbolic regression (see Equation 19.6). 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 aggregation protocol. It even tangents 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]:

$$f_1(u,e,r) = \left.\frac{1}{T*m}\sum_{t=1}^{T}\sum_{k=1}^{m}\varepsilon(k,t)^2\right|_{u,e,r} \tag{20.14}$$

This rather mathematical definition is realized indirectly in Algorithm 20.3, 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 20.1.3. Its realization in the Distributed Genetic Programming Framework [552] 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, 3 GHz off-the-shelf PC.

20.1.6 Input Data

In Algorithm 20.3 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 $getInput(k,0) = getInput(k,1) = \ldots getInput(k,T)\ \forall\ k$. If we have dynamic inputs on the other hand, we need to ensure that at least some elements of the input vectors $\mathbf{x}(t) = getInput(\star,t)$ 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

$$0.9 < \left|\frac{\alpha(\mathbf{x}(t))}{\alpha(\mathbf{x}(t))}\right| < 1.\overline{1} \tag{20.15}$$

[3] where $|_{u,e,r}$ means "passing u,e,r as input to Algorithm 20.3"

Algorithm 20.3: $f_1(u, e, r) = evaluateAggregationProtocol(u, m, T)$

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: Implicitly: *update* the update function
Input: Implicitly: I the index for the input values
Input: Implicitly: O the index for the output values
Input: Implicitly: e the index list for the send values
Input: Implicitly: r the index list for the receive values
Data: d communication step base according to 20.4b on page 344
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) ⟵ new n × m Matrix
       // initialize with local values
4      S(0)_{⋆,k} ⟵ getInput(k, 0)
5      t ⟵ 1
6      while t ≤ T do
7          S(t) ⟵ copyMatrix(S(t − 1))
8          ...
           // perform communication according to 20.4b on page 344
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 + (φ(i(t)) − S(t)_{o,k})^2
15             k ⟵ k + 1
16         t ⟵ t + 1
17     return res
18 end
```

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 $\alpha(\mathbf{x}(1,t)) \neq \alpha(\mathbf{x}(2,t)) \neq \ldots \neq \alpha(\mathbf{x}(l,t))$ 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:

$$\left(\left|\frac{\alpha(\mathbf{x}(i,t))}{\alpha(\mathbf{x}(j,t))}\right| \ll 1\right) \vee \left(\left|\frac{\alpha(\mathbf{x}(i,t))}{\alpha(\mathbf{x}(j,t))}\right| \gg 1\right) \; \forall \, i \neq j \tag{20.16}$$

We use z such data sets to perform z runs of Algorithm 20.3 and compute the true value of f_1 as arithmetic mean of the single results.

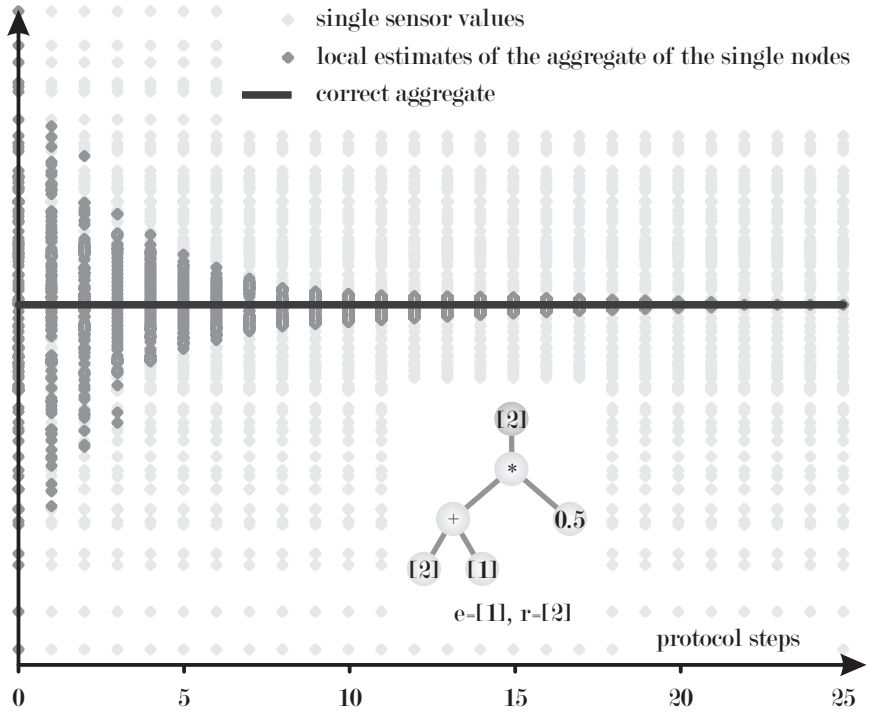
$$f_1(u,e,r) = \frac{1}{z}\sum_{i=1}^{z} f_1(u,e,r)|_{\mathbf{x_i}} \tag{20.17}$$

Of course, for each protocol that we evaluate we will use the same sample data sets because otherwise the results would not be comparable.

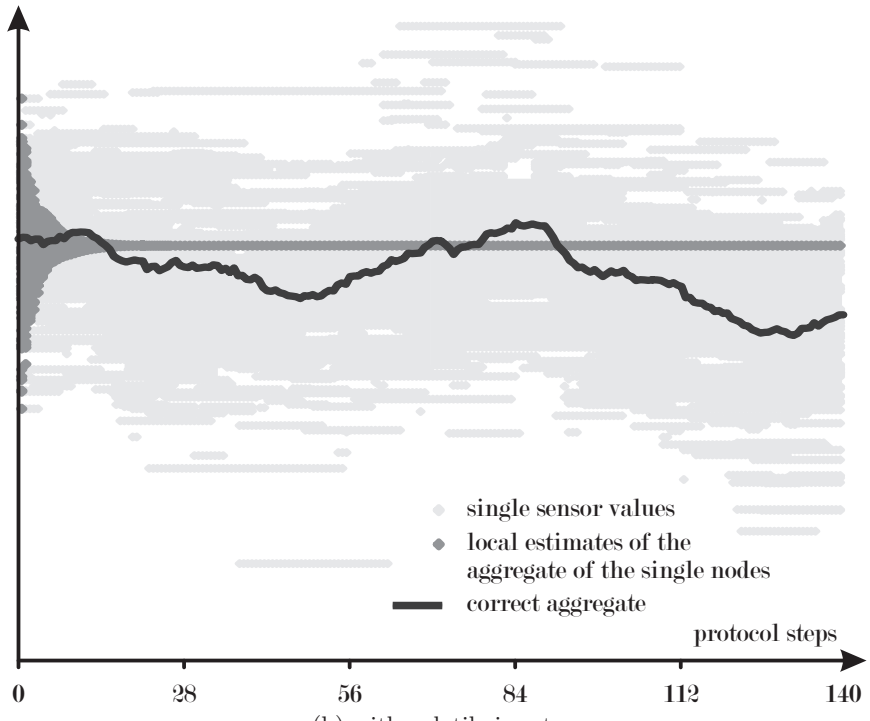
Volatile Input Data

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 [1080]. These protocols have good convergence properties, as illustrated in Figure 20.6a. They always converge to the correct results but will simple ignore changes in the input data (see Figure 20.6b).

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 they are volatile, the estimations of these protocols become more and more imprecise. 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 change slowly. If they 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 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,



(a) with constant inputs



(b) with volatile inputs

Fig. 20.6: The behavior of the distributed average protocol in different scenarios.

and the current input data into the calculations. A dynamic distributed average protocol like the one illustrated in Figure 20.7 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.

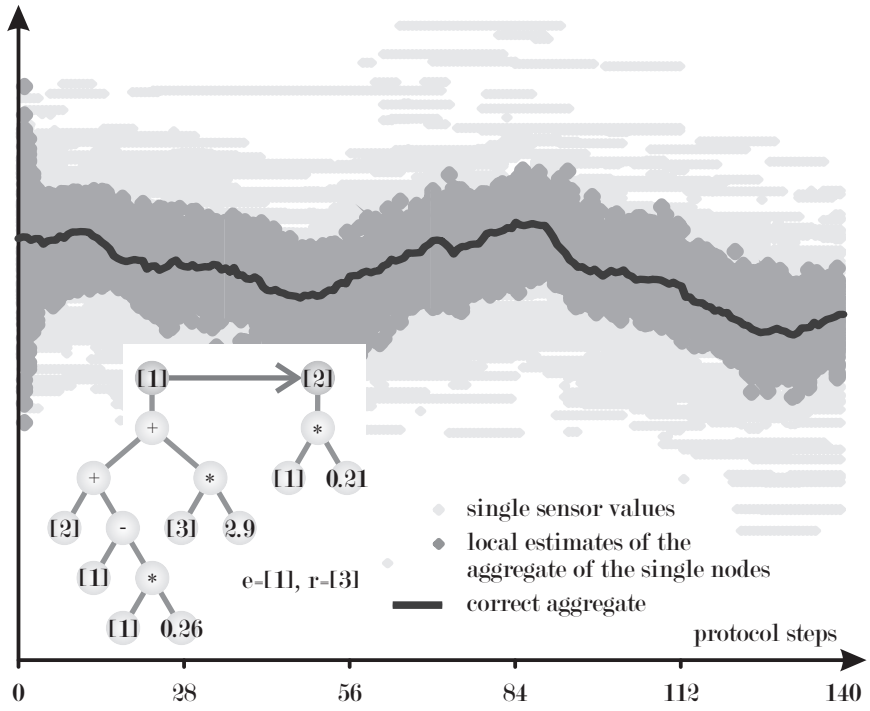


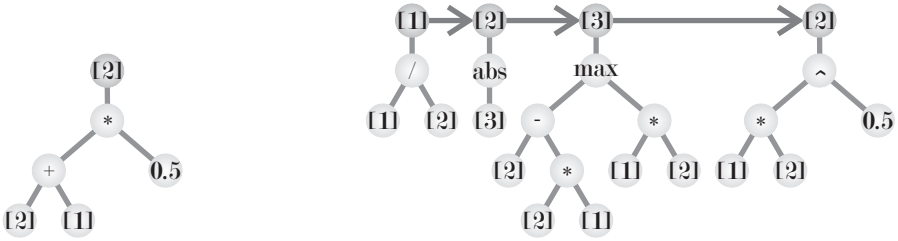
Fig. 20.7: A dynamic aggregation protocol for the distributed average.

20.1.7 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 $\mathbf{s}_k(t) \in \mathbb{R}^m$ of time step t as input. It returns the new state vectors $\mathbf{s}_k(t+1) \in \mathbb{R}^m$ 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 $j = 1, 2, \ldots, |l|$. In each step j, the result of the expression u_j is computed and assigned to the v_jth element of the old state vector $\mathbf{s}(t-1)$. In the simplest case, l will have the length $|l| = 1$. One example for this is the well-known distributed average protocol illustrated in Figure 20.8(a): 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 $\mathbf{s}_k(t+1)$. Figure 20.8b shows a more complicated protocol where



(a) distributed average

(b) square root of the distributed average

Fig. 20.8: Some examples for the formula series part of aggregation protocols.

update consists of $|l| = 4$ 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 20.1.8 on page 357 and Section 20.1.8 on page 358.

The important point is that we are able to provide a form for the first part of the aggregation protocol specification that is compatible to normal symbolic regression and which hence can be evolved using standard operators.

Besides a sequence of formulas computed repetitively in a cycle, we also need an additional sequence that is executed only once, in the initialization phase. This 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 20.2 and Algorithm 20.3, initially perform $S(0)_{\star,k} \longleftarrow getInput(k,0)$ instead of $S(0)_{i,k} \longleftarrow getInput(k,0)$. In all other time steps, only $S(t)_{i,k}$ is updated.

Straightforward, 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 ($|e_1| \neq |e_2|$) 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 3.4.1 on page 124. 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 3.4.2 on page 126. 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 it fixed or variable-length, can be applied.

When we closely examine our abstract protocol representation, we will see that it will work with epidemic [1082] or SPIN-based [1083] 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.

- 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, the element its results will be written to, and 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 145.
- 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.

20.1.8 Results from Experiments

For our experiments, we have used a simple elitist evolutionary algorithm with a population size of 4096 and an archive size of 64. In the simulations, 16 virtual machines were running, each holding a state vector $\mathbf{s}$ with five elements. For evaluation, we perform 22 simulation runs per protocol where each run is granted 28 cycles in the static and 300 cycles in the dynamic case.

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 individuals, 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 equivalently defined in Equation 20.18 and Equation 20.19 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 $c_{F,agg}$ leads to optimal sets with few members that most often (but not always) have equal objective values and only differ in their phenotypes.

$$c_{F,agg}(x_1, x_2) = \begin{cases} -1 \text{ if } f_1(x_1) < f_1(x_2) \\ 1 \text{ if } f_1(x_1) > f_1(x_2) \\ c_{F,pareto}(x_1, x_2) \text{ otherwise} \end{cases} \tag{20.18}$$

$$c_{F,agg}(x_1, x_2) = \begin{cases} -1 \text{ if } (f_1(x_1) < f_1(x_2)) \vee \\ \qquad ((f_1(x_1) = f_1(x_2)) \wedge (f_2(x_1) < f_2(x_2))) \vee \\ \qquad ((f_1(x_1) = f_1(x_2)) \wedge (f_2(x_1) = f_2(x_2)) \wedge \\ \qquad\qquad (f_3(x_1) < f_3(x_2))) \\ 1 \text{ if } (f_1(x_1) > f_1(x_2)) \vee \\ \qquad ((f_1(x_1) = f_1(x_2)) \wedge (f_2(x_1) > f_2(x_2))) \vee \\ \qquad ((f_1(x_1) = f_1(x_2)) \wedge (f_2(x_1) = f_2(x_2)) \wedge \\ \qquad\qquad (f_3(x_1) > f_3(x_2))) \\ 0 \text{ otherwise} \end{cases} \tag{20.19}$$

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. So the objective function f_3 that minimizes it 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 $\alpha_{avg} = \overline{\mathbf{x}}$ in many of the previous examples, for instance in Section 20.1.1 on page 339, Section 20.1.6 on page 351, and in Figure 20.8a.

The evolution of a static version such an algorithm is illustrated in Figure 20.9. The graphic shows how 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 then 50 generations.

Figure 20.10 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, so does the number of formula expressions. These peaks in f_2 are always followed by a recession caused by stepwise

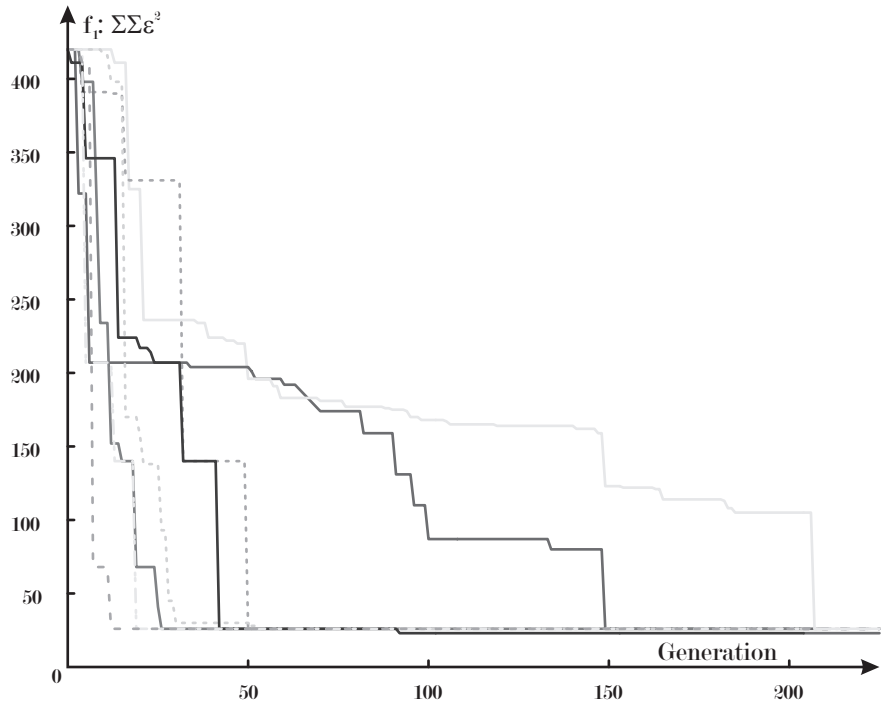


Fig. 20.9: The evolutionary progress of the static *average* protocol.

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 [548]. Here, a distributed average protocol for the aggregate function α_{ra} is to be evolved:

$$\alpha_{ra}(\mathbf{x}) = \sqrt{|\overline{\mathbf{x}}|} \tag{20.20}$$

One result of these experiments has already been sketched in Figure 20.8b. Figure 20.11 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

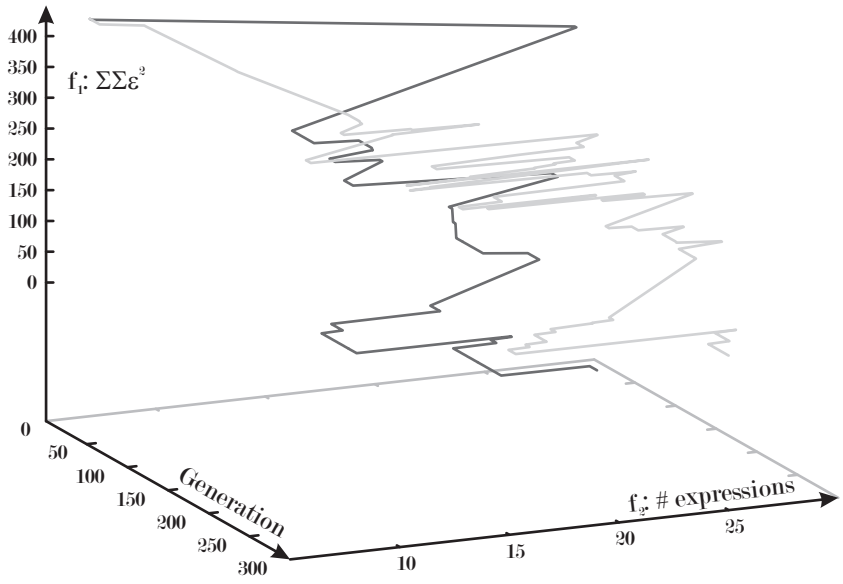


Fig. 20.10: The relation of f_1 and f_2 in the static *average* protocol.

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.

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 20.12 for the 84-generation solution.

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 20.13.

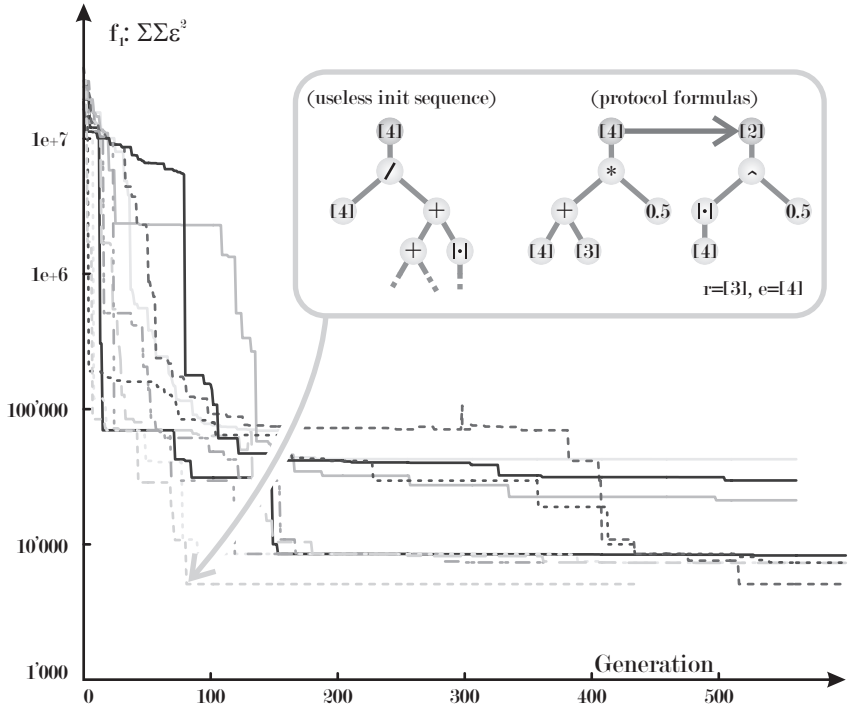


Fig. 20.11: The evolutionary progress and one grown solution of the static *root-of-average* protocol.

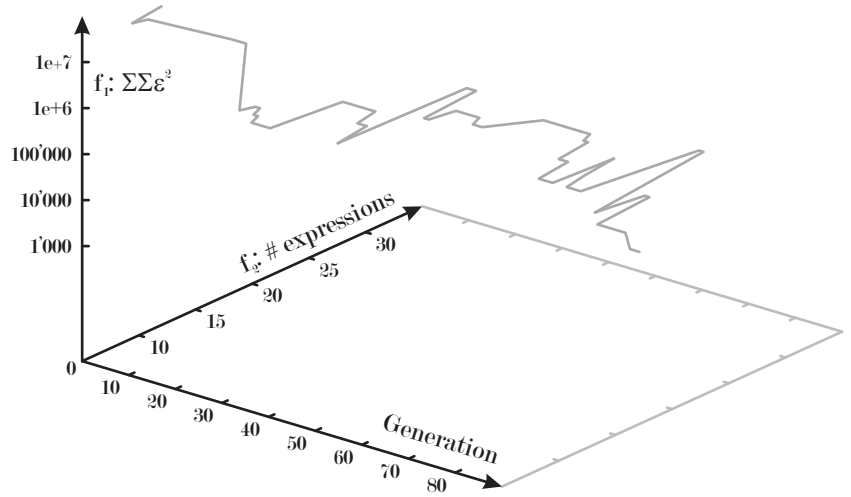


Fig. 20.12: The relation of f_1 and f_2 in the static *root-of-average* protocol.

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

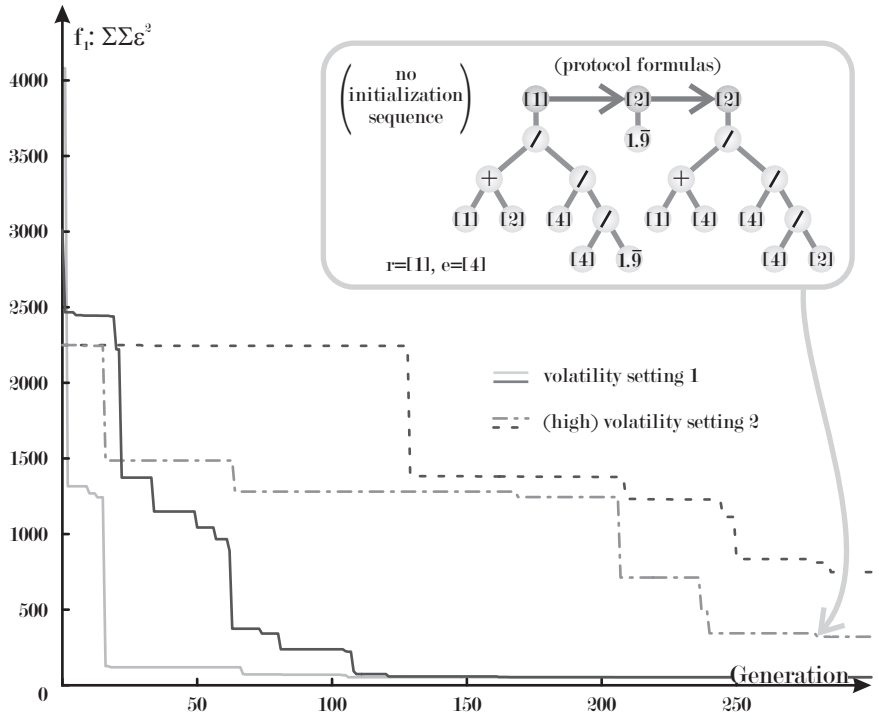


Fig. 20.13: The evolutionary progress of the dynamic *average* protocol.

error functions – the more unsteady the protocol inputs, the higher will f_1 be, as Figure 20.13 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 outlined in Figure 20.14.

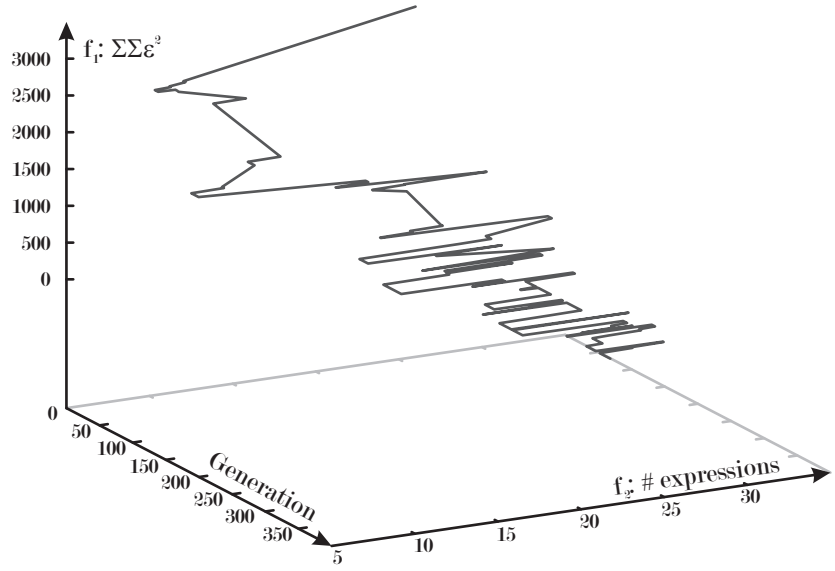


Fig. 20.14: The relation of f_1 and f_2 in the dynamic *average* protocol.

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 20.1.8.

Figure 20.15 shows how f_1 changes by time. Like in Figure 20.11, 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.

The relation of f_1 and f_2, outlined in Figure 20.16 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.

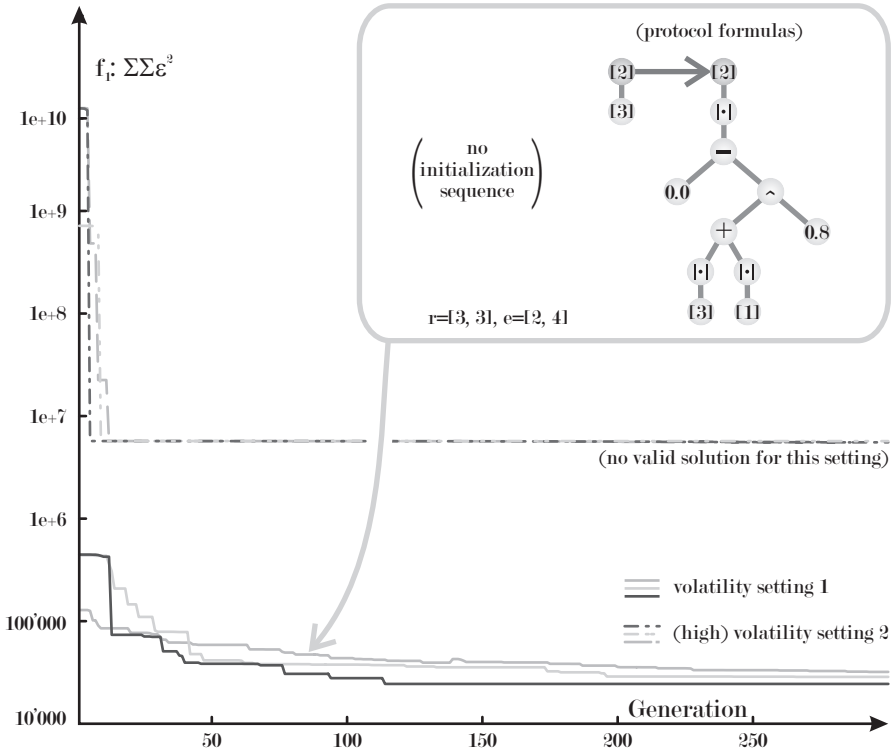


Fig. 20.15: The evolutionary progress and one grown solution of the dynamic *root-of-average* protocol.

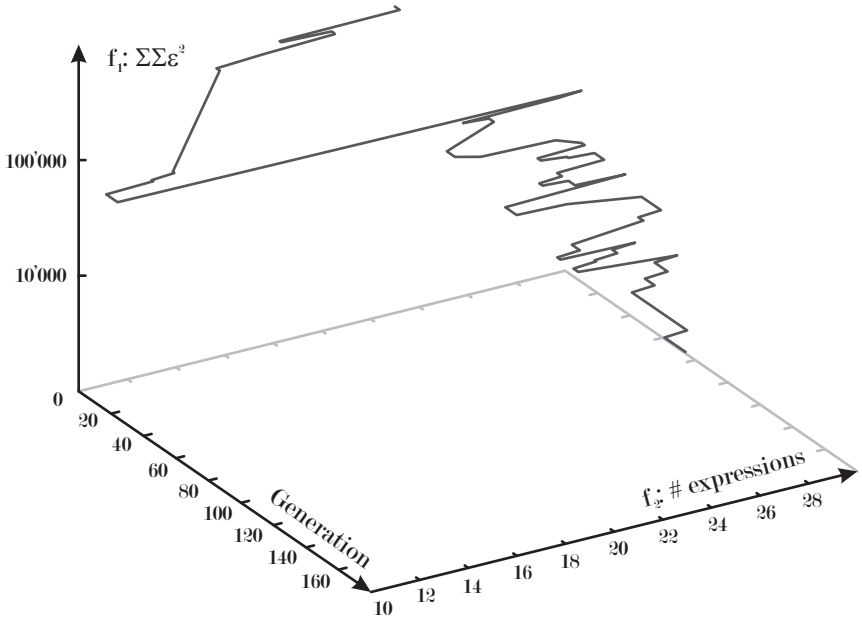


Fig. 20.16: The relation of f_1 and f_2 in the dynamic *root-of-average* protocol.

Part III

Sigoa – Implementation in Java

21

Introduction

Today, there exist many different optimization frameworks. Some of them are dedicated to special purposes like spacecraft design [228] or trading systems [428]. Others provide multi-purpose functionality like GALib [1084], Evolutionary Objects (EO) [1085, 1086] or the Java evolutionary computation library (ECJ) [664].

In this part of the book we want to introduce a new approach in form of the Sigoa, the Simple Interface for Global Optimization Algorithms[1]. Using this library, we want to demonstrate how the optimization algorithms discussed in the previous chapters can be implemented.

We decided to use Java [1087, 1088, 1089] for this project since it is a very common, object-oriented, and platform independent programming language. You can find more information on Java technology either directly at their website `http://java.sun.com/` [accessed 2007-07-03] or in books like

- *Javabuch* [1090], the best German Java learning resource in my opinion, is online available for download at `http://www.javabuch.de/` [accessed 2007-07-03].
- For the English reader, *Thinking in Java* [1091] would be more appropriate – its free, third edition is online available at `http://www.mindview.net/Books/TIJ/` [accessed 2007-07-03].
- As well as interesting are the O'Reilly books *Java in a Nutshell* [1092], *Java Examples in a nutshell* [1093], and *Learning Java* [1094].
- *Java ist auch eine Insel* [1095] – 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 software described in this book part can be found online at `http://www.sigoa.org/`. All the software described here is not only open-source, but licensed very liberally under the LGPL (see ap-

[1] `http://www.sigoa.org/`

pendix Chapter B on page 641) which allows for the integration of Sigoa into all kinds of systems, from educational to commercial, without any restrictions.

Sigoa has features that especially support optimizing complicated types of individuals which require time-consuming simulation and evaluation procedures. Imagine you want to grow algorithms with a genetic programming approach. In order to determine 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 simulate the algorithms grown multiple times and then use the minimum, median, average, or maximum objective values encountered.

Maybe the algorithms that we grow are no simple programs performed by a single processor but distributed algorithms that involve interaction and communication of multiple processors. In this case, it is again not sufficient to simulate one processor – 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. Furthermore, it is often the case that the individuals optimized are also large and costly considering storage and mutation expenses.

A framework capable of partially dealing with such aspects in an elegant manner has already been developed by the author in the past, see [550, 551, 552]. 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 to include new, experimental methods that may have not been discussed yet.

Before starting with defining the Sigoa specification, we performed a detailed study on different global optimization algorithms and evolutionary algorithms which is in principle provided in Chapter 2 on page 47. Additionally, we used the lessons learned from designing the Dgpf system to write down the following major requirements:

[2] See Section 4.11 on page 196 for further discussions.
[3] In Section 20.1 on page 337 you can find good example for this issue.

21.1 Requirements Analysis

21.1.1 Multi-Objectivity

Almost all real-world problems involve contradicting objectives. A distributed algorithm evolved should for example be efficient but 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 as discussed here is thus multi-objectivity.

21.1.2 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 47.

From this primary 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 Section 2.6.13 on page 110) which originally uses tournament selection to use roulette wheel selection instead (whether this would make sense or not).

This means that we will define Java-interfaces for all parts of optimization algorithms such as fitness assignment or clustering algorithms used for pruning the optimal sets. Thus, we reach a separation of the specification from the implementation. For all interfaces we will provide a reference implementation which, however, can easily be exchanged, allowing for different levels of complexity in the realizations.

21.1.3 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 nowadays off-the-shelf PCs already offer supportive hyper-threading or dual-core technology [1096, 971]. If Sigoa is used by different other software systems

which transfer optimization tasks to it, security issues arise. These aspects are orthogonal to the mathematical task of optimizing but vital for the system to work. They should be specified at different places and clearly be separated from the pure algorithms.

21.1.4 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 21.1.3), 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 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.

21.1.5 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 16 on page 263.

21.2 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.

Resulting 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 21.1 illustrates this top-level package hierarchy.

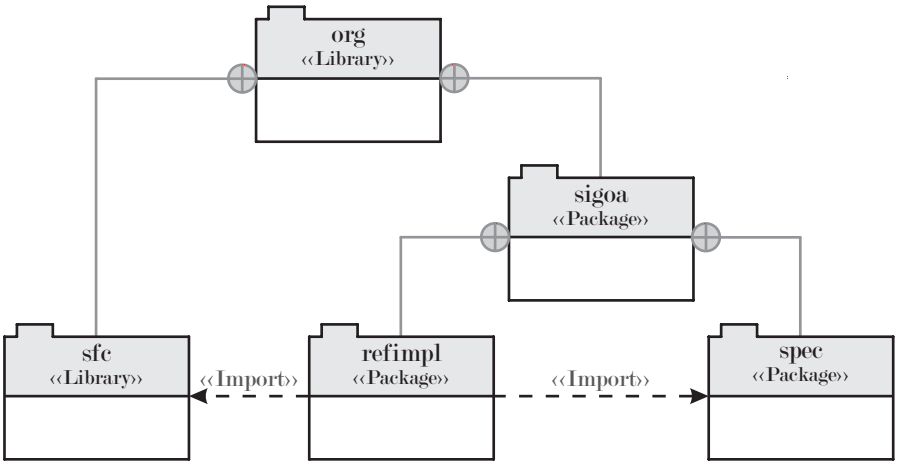


Fig. 21.1: The top-level packages of the Sigoa optimization system.

The specification 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 as can therefore also be implemented as wrappers that bridge to other, existing optimizing systems.

Most of the specification interfaces inherit from `java.io.Serializable` and hence can be serialized using the Java Object Serialization mechanisms[4]. We

[4] `http://java.sun.com/javase/6/docs/technotes/guides/serialization/` [accessed 2007-07-03]

do so to provide the foundation for the ability to create snapshots of a running optimization algorithm. This then allows them to be started, stopped, restarted and migrated.

The reference implementation uses an additional software package called Sfc, the Java Software Foundation Classes[5] – an open-source library (LGPL-licensed, see appendix Chapter B on page 641) 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[6]-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.

From here on we will adhere a short naming of the packages in order to increase comprehensibility. The package `org.sigoa.spec` will just be called `spec` in a context where it is clear that `org.sigoa.spec` is meant.

21.3 Subsystems

As illustrated in Figure 21.2, the Sigoa framework is constituted by nine subsystems:

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[7]: a rule could be defined that terminates an algorithm after a given time. A detailed explanation on this subsystem can be found in Chapter 23 on page 387.
2. In the `clustering`-package, interfaces for clustering-algorithms (as defined in Chapter 36 on page 571) are specified. Clustering algorithms can be used to reduce a large set of solution candidates to a smaller one without loss of generality. In Chapter 30 on page 437 more information on this package is provided.
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 the optimization process not treat as a mere mathematical procedure – it will always be part of some application. As such, not only the final results

[5] `http://sourceforge.net/projects/java-sfc/` [accessed 2007-07-03]

[6] `http://java.sun.com/javase/6/docs/technotes/guides/collections/` [accessed 2007-07-03]

[7] see Section 1.6.2 on page 31

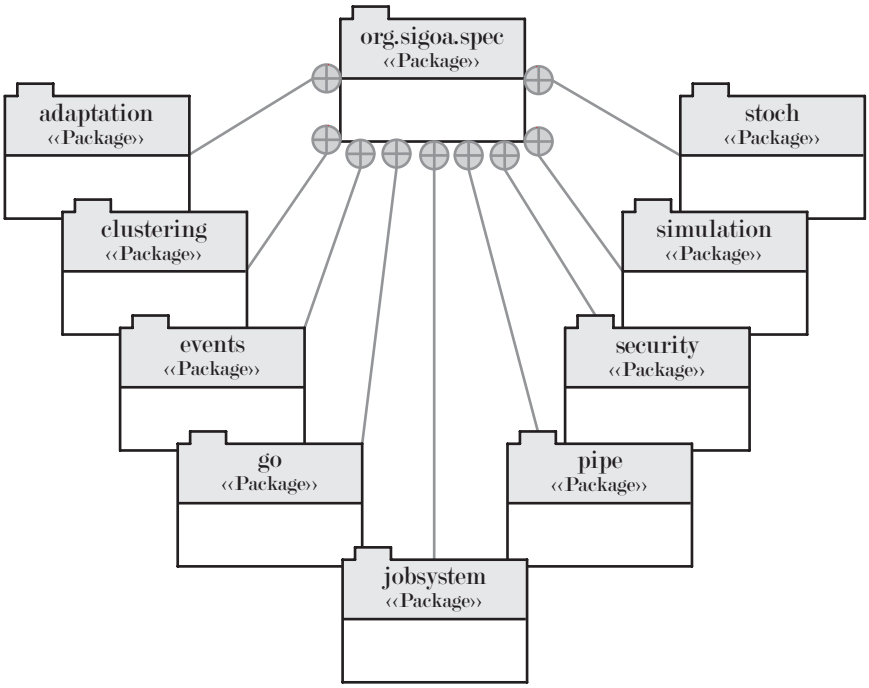


Fig. 21.2: The subsystem specification of the optimization framework.

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. The description of the Sigoa event API can be found in Chapter 24 on page 391.

4. The largest subsystem is the `go` package, where all components and sub-algorithms for global optimization are specified. Here you can find the interfaces specifications that cover the all the algorithmic and mathematical functionality of global optimization algorithms. Chapter 31 on page 445 provides a deeper insight into this subsystem.
5. In the `jobsystem` package, we place the specification of the means to *run* optimization algorithms. An optimization algorithm may be parallelized or run sequentially, and it therefore may use multiple threads. The algorithm itself should however be separated from the parallelism issues. Applying the definitions of the `jobsystem` package, algorithms 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 are run in different threads or performed sequentially. This way it also possible to manage multiple optimization algorithms and to specify which one will be assigned to how many processors. The implementations of the job system

specifications could also perform accounting and performance monitoring of the work load. More details of the job system can be found in Chapter 28 on page 413.

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 optimization algorithm completely – totally new components, like statistic monitors can easily be inserted into a system with minimum changes. The package `pipe` is explained in Chapter 29 on page 427.
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[8]. The security model of Sigoa is described in Chapter 25 on page 395.
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. The simulation subsystem is precisely described in Chapter 27 on page 405.
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 some sort of statistical evaluation. This data may also be used by the optimization algorithms themselves or by adaptation rules. Furthermore, the global optimization algorithms as discussed here are randomized in the major. They thus need random number generators as introduced in Section 35.7 on page 559. Such utilities are specified in the `stoch` package which is elaborated on in Chapter 26 on page 399.

[8] `http://java.sun.com/javase/6/docs/technotes/guides/security` [accessed 2007-07-03]

22

Examples

But before we are going into detail about the different packages and utilities of the Sigoa software, we will present some application examples hare. 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. The more specific discussion of the Sigoa packages following after this chapter then rounds up the view on this novel optimization system.

22.1 The 2007 DATA-MINING-CUP

As an application example for genetic algorithms, the 2007 DATA-MINING-CUP Contest has been introduced in Section 18.1.2 on page 300. We strongly recommend reading this section first. We there have talked about the basic principles behind the challenge, while we here 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.

First, we start by representing the three groups using a simple Java `enum` like in listing 22.1.

Our approach in Section 18.1.2 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 18.1). Thus, we simple strip the LCS from its "learning" capability and directly maximize the profit directly.

```
1 public enum EClasses {
2   /** class A */
3   A,
4   /** class B */
5   B,
6   /** class N */
7   N;
8 }
```

Listing 22.1: The enum EClasses with the possible DMC 2007 classifications.

22.1.1 The Phenotype

What remains are mere classifier systems which are the phenotypes of a genetic algorithm. They consist of a set of rules `m_rules` (the single classifiers, `byte` arrays containing the conditions) and rule outputs `m_results` (instances of `EClasses`).

Listing 22.2 illustrates how the method `classify` works which classifies a set of 17 relevant features `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 18.2 on page 306. If so, the corresponding classification `m_results[i]` is returned. `classify` keeps also track on the rule which is violated the least by `data` in the variables `lec` and `leci`. If no perfectly matching rule could be found, the $\frac{1}{5}$-threshold 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.

22.1.2 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 3.4.2 on page 126 extensively. In Figure 18.4 we have introduced the genotype-phenotype relation 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* of our genome. A classifier system in turn may consist of an arbitrary number of such classifiers.

In Sigoa, we represent variable-length as fixed-length bit strings as byte arrays (`byte[]`) for which predefined operations exist. So in principle, we do

```
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 22.2: The structure of our DMC 2007 classifier system.

not have to deal with their reproduction 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 are subsumed under the area of genotype-phenotype mapping (see Section 3.5 on page 127) and its sub-division artificial embryogeny discussed in Section 3.5.1 for which Sigoa offers a core interface interface `IEmbryogeny` (see Section 31.1.6 on page 456) and a reference implementation `Embryogeny` (see Section 31.2.5 on page 474) along with a special embryogeny extension for bitstrings, `BitStringEmbryogeny` (see Section 32.2.2 on page 493) which provides special streams for input and output from structured data from and to bit strings. We simple need to extend this class by providing (at least) the transformation function *hatch* from genotypes to phenotypes and (optionally) vice versa. Listing 22.3 shows this extension in form of the class `ClassifierEmbryogeny`. The constant `CLASSIFIER_EMBRYOGENY` provides a globally shared instance of this new embryogeny.

22.1.3 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 at least good share of them and keep another part to check if our classifier systems generalize well). For these test 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 the, in this case zero-based, indices 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** – $2'799$ in the example matrix M_{ex} of Equation 22.1. From M_{ex} we can furthermore read that $1'087$ of the samples belonging to class **B** were correctly classified whereas 777 were assigned to class **A** and $1'462$ to class **N**.

$$M_{ex}(C) = \begin{pmatrix} 4'062 & 856 & 3'794 \\ 777 & 1'087 & 1'462 \\ 5'484 & 2'799 & 29'679 \end{pmatrix} \tag{22.1}$$

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 Section 27.1 on page 405) and its standard implementation `Simulation` (see Section 27.2 on page 408). This default implementation just needs to be extended so it uses the test samples, which 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 22.4 shows the most important code of the new class `ClassificationSimulation`. In order to allow us to publish the new simulation later to the simulation manager of the optimization job system, we also

```
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 22.3: The embryogeny component of our DMC 2007 contribution.

```
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 22.4: The simulation for testing the DMC 2007 classifier systems.

provide a globally shared factory in form of an `ISimulationProvider`-instance with the constant `PROVIDER` in line 3.

22.1.4 The Objective Functions

On the foundation of the new simulation for classifier systems, we can define the objective functions that should guide the evolution. In Section 18.1.2 on page 306 we have introduced the two most important objective functions:

```
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 22.5: The profit objective function $f_1(C) = -P(C)$ for the DMC 2007.

one that minimizes $f_1(C) = -P(C)$ and hence, maximizes the profit, and $f_2(C) = |C|$, that minimizes the number of rules in the classifier systems.

All objective functions in Sigoa are instances of the interface `IObjectiveFunction` (see Section 31.2.3 on page 470). They can be derived from its default implementation `ObjectiveFunction` (see Section 31.2.3 on page 470) which implements the basic functionality so only the real mathematical computations need to be added.

In listing 22.5, we implement f_1. The method `endEvaluation` needs to be overridden. Here we store negated profit into a state record passed in. This state record is later on used by the optimization system to compute the objective value and to store it into the individual records.

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 question to this is relatively simple: In line 3 we have defined an instance of `SimulationProvider` for the `ClassificationSimulation`. This provider will later be introduced into the optimization job system. It uses `ClassificationSimulation.class` as identifier per default. With the method `getRequiredSimulationId` on line 16, we tell the job system that we need a simulation which is made available by an

```
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 classifier system |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 22.6: The size objective function $f_2(C) = |C|$ for the DMC 2007.

provider with exactly this id. Before passing the simulation to our objective function, the job system will call its `beginIndividual` method which will in turn build 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 22.6, we do not need any simulation. Instead, we query directly the number of rules in the classifier system via the method `getSize`. In listing 22.2, we have omitted this routine for space reasons, it simple returns `m_rules.length`. Again, this value is stored into the state record passed in from the job system's evaluator component which will then do the rest of the work.

22.1.5 The Evolution Process

Now the work is almost done, we just need to start the optimization process. Listing 22.7 presents an according `main`-method which is called on startup of a Java program.

First we have to decide which global optimization 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 need to construct an `IOptimizationInfo`-record with all the information that will guide the evolution[2]. To this information belongs how

[1] see Section 31.3.1 on page 482

[2] `IOptimizationInfo` is discussed in Section 31.1.9 on page 460, its reference implementation `OptimizationInfo` in Section 31.2.9 on page 481.

the solution candidates should be evaluated. Therefore we use an instance of `Evaluator`[3] (line 15) which is equipped with a `List` containing our two new objective functions (see 10 and 12). We furthermore tell the system to perform a pure Pareto-optimization as discussed in Section 1.3.2 on page 14 by passing the globally shared instance of `ParetoComparator`[4] (line 16) into the info record. We then define that our new 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 for example 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 22.1.3 on page 378 by adding its provider to the simulation manager in line 27. The job system that we create 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, all instances of `IJobSystem` specified in Section 28.1.2 on page 415, are discussed in Section 28.2.3 on page 422. We add our optimizer to the system in line 29 and finally start it in 30. Since have added no means to halt the evolution, the system will basically run forever in this example.

In order to get information on its progress, we have added two special output pipes (see Section 29.2.3 on page 434) 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 the information 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 this information, we can later easily reconstruct the complete individuals and, for instance, integrate them into the real applications. The second printer pipe, an instance of

[3] The class `Evaluator`, discussed in Section 31.2.4 on page 473, is the default implementation of the interface `IEvaluator` specified in Section 31.1.5 on page 456.

[4] The class `ParetoComparator`, elaborated on in Section 31.2.1 on page 462, implements the interface `IComparator` defined in Section 31.1.1 on page 448.

[5] These operations are introduced in Section 32.2.3 on page 493 implement the interfaces `ICreator`, `IMutator`, and `ICrossover` specified in Section 31.1.2 on page 450.

[6] see Section 27.2 on page 408

[7] see Section 27.1 on page 405

`ObjectivePrinterPipe`, stores only the objective values in a comma-separated-values format. The output files are put into the directors *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 and can also be read into spread sheets or graphical tools in order to produce fancy diagrams.

```
1  public static void main(String[] args) {
2      EA<byte[], ClassifierSystem> opt;
3      IOptimizationInfo<byte[], ClassifierSystem> oi;
4      IJobSystem s;
5      SimulationManager m;
6      List<IObjectiveFunction<ClassifierSystem, ?, ?,
           ISimulation<ClassifierSystem>>> l;
7
8      opt = new ElitistEA<byte[], ClassifierSystem>(10 *
           1024, 0.4d, 0.4d);
9
10     l = new ArrayList<IObjectiveFunction<ClassifierSystem,
           ?, ?, ISimulation<ClassifierSystem>>>();
11     l.add(new ProfitObjectiveFunction());
12     l.add(new SizeObjectiveFunction());
13
14     oi  = new OptimizationInfo<byte[], ClassifierSystem>(
15             new Evaluator<ClassifierSystem>(l),
16             ParetoComparator.PARETO_COMPARATOR,
17             ClassifierEmbryogeny.CLASSIFIER_EMBRYOGENY,
18             new VariableLengthBitStringCreator(36),
19             new VariableLengthBitStringMutator(36),
20             new VariableLengthBitStringNPointCrossover(36),
21             null);
22
23     opt.addNonPrevailedPipe(new
           IndividualPrinterPipe<byte[],
           ClassifierSystem>(new FileTextWriterProvider(new
           File("c"),"c"), false));
24     opt.addNonPrevailedPipe(new
           ObjectivePrinterPipe<byte[], ClassifierSystem>(new
           FileTextWriterProvider(new File("bo"), "bo"),
           false));
25
26     m = new SimulationManager();
27     m.addProvider(ClassificationSimulation.PROVIDER);
28     s = new MultiProcessorJobSystem(m);
29     s.executeOptimization(opt, new JobInfo<byte[],
           ClassifierSystem>(oi));
30     s.start();
31   }
```

Listing 22.7: A `main` method that runs the evolution for the 2007 DMC.

23

The Adaptation Mechanisms

Adaptation is an important aspect in the field of optimization algorithms [1097, 1098, 1099, 750, 1100]. Especially interesting is self-adaptation. An evolutionary algorithm for example has a mutation- and a crossover rate. If it now detects for a longer time span that the crossover of created by mutation operations if good in many cases whereas mutation yields dysfunctional individuals, it may increase the crossover- and decrease the mutation rate. An optimization algorithm may also notice its own convergence. If the optimizer detects that its result approximation does not change anymore, it may terminate (or reset) itself. Self-termination may also be performed if a predefined runtime elapses or if a given maximum count of iterations has been performed.

23.1 Specification

The `org.sigoa.spec.adaptation` package provides the means to implement such behavior in a versatile, reusable manner. Its interface hierarchy is displayed in Figure 23.1.

An object that can be adapted or (auto-adapt itself) using the Sigoa adaptation mechanisms must implement the `IAdaptable`-interface. Such an object owns a list of rules (instances of `IRule`) which it exposes through the `getRules`-function. Rules can be added to or removed from this list, changing the object's adaptation behavior. *It lies in the responsibility of the adaptable object (the implementation of the `IAdaptable`-interface) to apply all the rules in the list in a regular basis*, i. e. to check if their conditions are met and if so, to perform their actions on itself.

A rule consists an instances of each of the two basic interfaces `ICondition` (accessed by `getCondition`) and `IAction` (accessed by `getAction`). `ICondition` represents a condition that must be true in order for the rule to be applied. This condition is checked by invoking the `evaluate` method on the adaptable object which owns the rule. `IAction` represents an action executed (by invoking the `perform`-method) on the object if the condition is met.

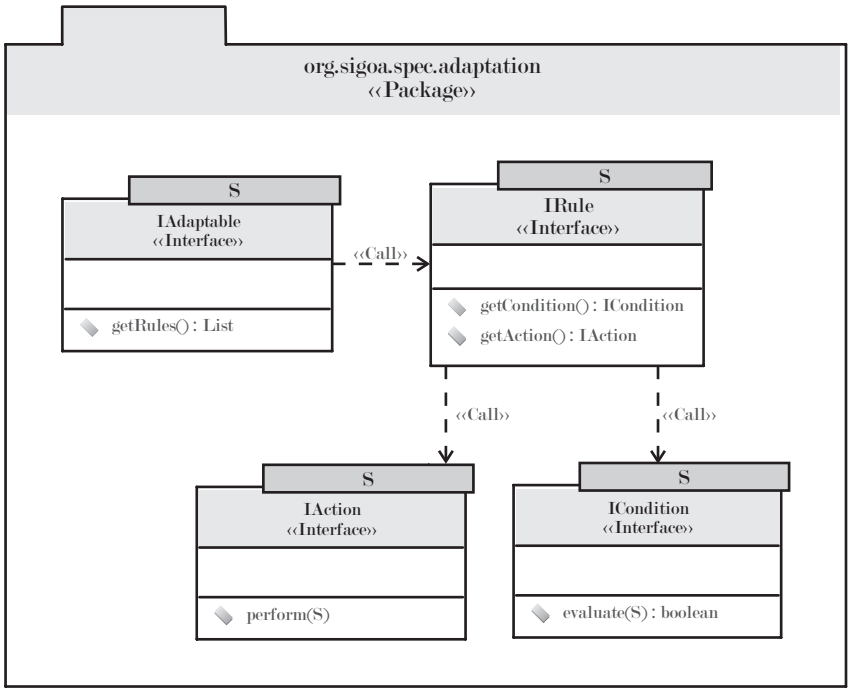


Fig. 23.1: The specification of the Sigoa adaptation mechanisms.

Conditions and actions are specified separately in order to allow complex conditions and complex actions to be reused.

23.2 Reference Implementation

The reference implementation of the Sigoa adaptation mechanisms can be found in the package `org.sigoa.refimpl.adaptation` which is illustrated in Figure 23.2. The class `CompoundRule` implements the `IRule`-interface by having final member variables for the condition- and actions parts.

The interfaces `IAction` and `ICondition` do not require any standard implementation since the only methods they define consider their direct behavior for which no defaults can be given. The packages `conditions` and `actions` exist to provide some predefined, precasted conditions and actions. If an optimizer wants to terminate after a given time for instance, it could compose a rule of an instance of `TimeCondition` (which evaluates to `true`after the time span specified in the constructor is elapsed) and `AbortAction` (which aborts the activity it is invoked on).

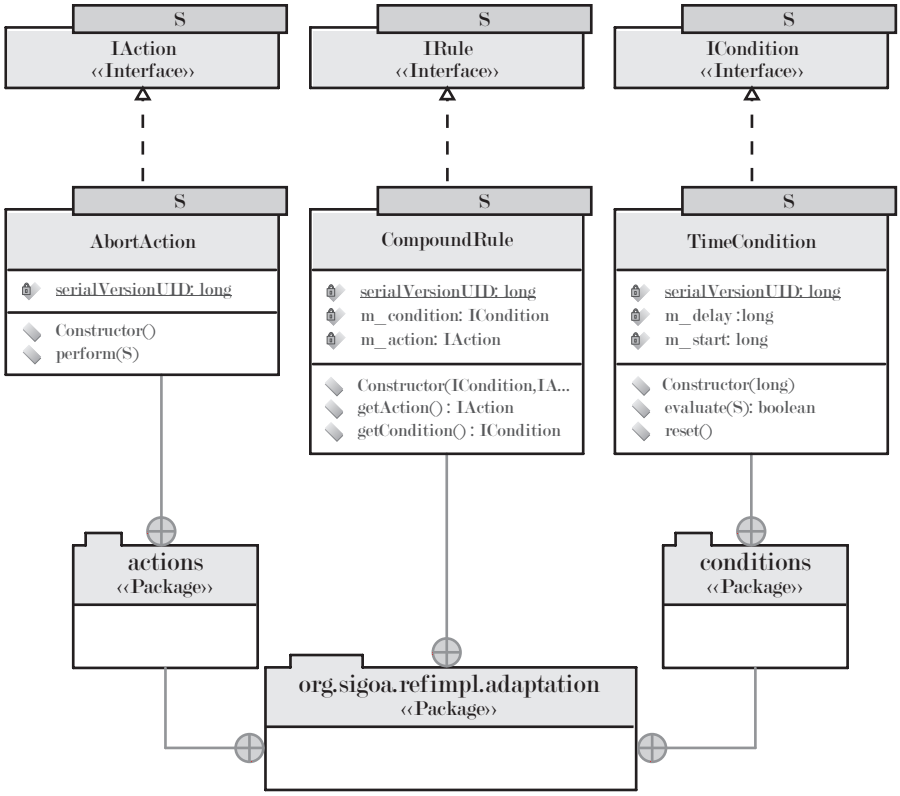


Fig. 23.2: The reference implementation of the adaptation mechanisms.

24

The Events Package

The Sigoa framework is not stand-alone – it will be integrated into applications that make use of its optimization capabilities. It is a component that performs its work in parallel and has its own security management. Since most of the optimization algorithms it provides will be strongly randomized, there is probably no deterministic behavior – optimizations of the same problem with the same parameters will maybe produce different results and may differ in time-consumption if executed twice in a row.

For all these features there should be a way for the application that uses Sigoa to obtain status information, statistics, and error messages. In principle, there exist two methods to realize information transfer for software components: there can either be interfaces to the component allowing active inspection from the user application or a passive interface through which the component provides its information to the outside world.

Sigoa provides the passive mechanism, since it is more flexible and can be realized side-effect free. As already said, the Sigoa implementations will run asynchronously. Active introspection may result in inconsistent data since the internal state may change between two requests belonging to the same query.

Passive information transfer means that the component (Sigoa, in this case) invokes a method of the user application, whenever information is available. This is realized in the Sigoa system by the event API.

24.1 Specification

The `event` package provides the means to create events containing information, to define objects that produce such events and to specify other objects able to receive them. Its interface hierarchy is displayed in Figure 24.1.

A component of the Sigoa framework that is able to provide information to the outside world implements the `IEventSource`-interface. It is an event source and internally manages a list of `IEventListener`-instances. Objects that implement `IEventListener` can be added to an event source using the

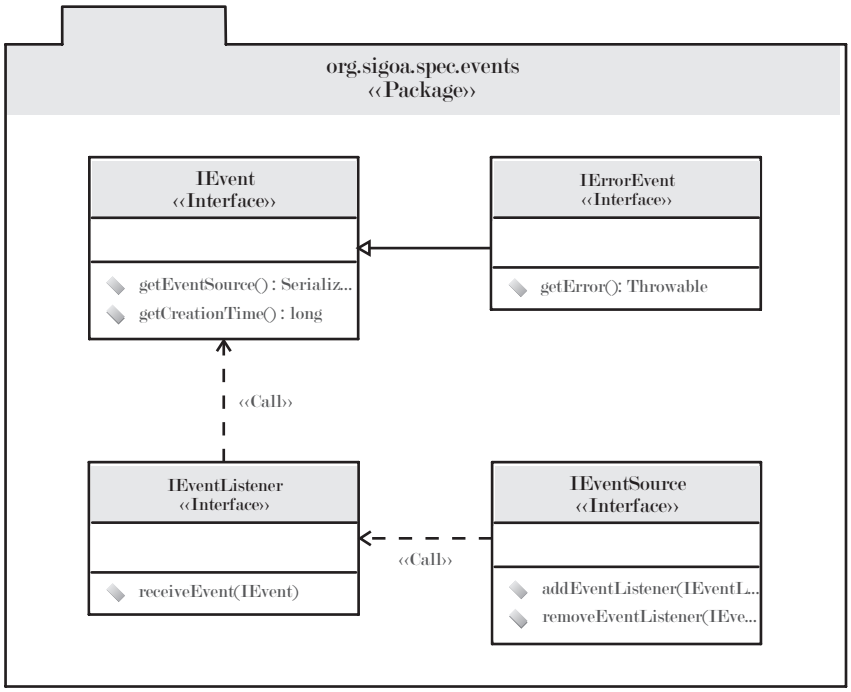


Fig. 24.1: The specification of the Sigoa event objects.

`addEventListener`-method. Whenever the event source has new information available, it packages it into an `IEvent`-record and passes this record to the `receiveEvent`-methods of all its listeners. Notice that there is nothing said about parallelism here – it may be possible that an event source provides multiple events to an event listener in parallel. Synchronization must be performed on the `IEventListener`-side. If an event listener does no longer want to receive information from an event source, it can be unregistered using the `removeEventListener`-method of the event source.

`IEvent`-records provide their creation time (`getCreationTime()`, in Java time[1]) along with the event source identifier (`getEventSource`). The event source identifier may lightly be mistaken for the event source itself. This is not the case for a very simple reason: events may be serialized and transmitted over a network to a remote location. This will usually be the case if the optimizer is running on a different machine in order to provide full computational power to it without interference with other components. If the event record would store a direct reference to its (serializable) source, this would lead to the whole event source being transmitted over the network. If the source is the

[1] see `System.getCurrentMillis()`-documentation

optimizer itself, a snapshot of the whole process (probably many Mibibytes in size) would be copied over the network with an event that maybe just says "optimization finished". Implementors of the Sigoa-event API should thus use another identification scheme and store event source identifiers different from the objects actually creating the events.

The `IEvent`-interface can be extended for events with different semantics; the `IErrorEvent`-interface for example specifies an event that is generated because an error was caught. This interface provides the error information in form of a `java.lang.Throwable`-instance which can be obtained via the `getError`-method.

24.2 Reference Implementation

The package `org.sigo.refimpl.events` contains the reference implementation of the Sigoa event API, the most important features are illustrated in Figure 24.2.

The class `Event` implements the `IEvent`-interface. It inherits from `java.util.EventObject`, so the standard Sigoa events are compatible with the Java event handling[2]. The event source can either be set directly in the constructor or be determined automatically with the static method `getCurrentId`. The automatic source determination requires that the event is created inside an activity running in a job system compatible to the Sigoa job system see Chapter 28 on page 413. It uses the automatically created, unique identifier of the current optimization job. The event's creation time will automatically set to the current system time. The methods `equals` and `toString` have been overridden and do now compare/create textual representations of the event fields.

The class `ErrorEvent` implements the `IErrorEvent`-interface and can be used to propagate a caught exception.

`IEventSource`s have to maintain a list of `IEventListener`s and forward events to them. The class `EventPropagator` is a utility for this purpose: it implements both interfaces. If it receives an event with its `receiveEvent`-method, it forwards it to all its listeners. Instances of this class can be used by all real event sources to manage and notify their listeners or as event-multiplexer.

The `stat` package provides support to transform events to other output formats. Its `EventPrinter`-class for example is an `IEventListener` that stores the textual representations of all the events it receives into a stream. This utility may be useful for writing log files.

[2] Examples for Java event handling are Swing and AWT-events: `http://java.sun.com/docs/books/tutorial/uiswing/learn/example2.html#handlingEvents` [accessed 2007-07-03]

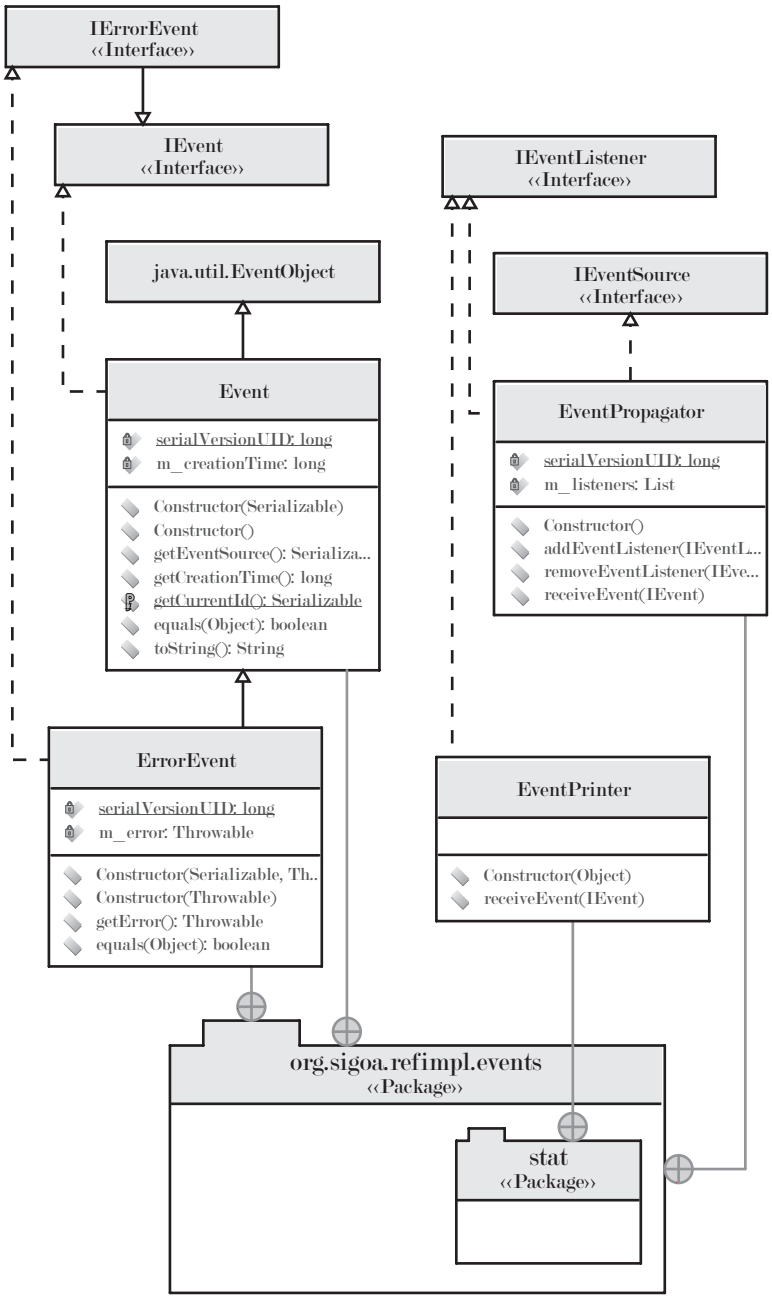


Fig. 24.2: The reference implementation of the Sigoa event objects.

25

The Security Concept

The applications of the Sigoa framework are manifold. It could be used as optimization component inside a program or run on a large cluster and provide its capabilities in the form of web services. A corporation could rent runtime for optimization processes to users who would upload optimization jobs in the form of jar-Archives or class files.

A wide variety of applications requires security management by nature, so there exists a simple but efficient security concept for the Sigoa too.

25.1 Specification

Java comes already shipped with a security technology[1]. Its security model is based on a class named `java.lang.SecurityManager`. This class can check permissions in the form of `java.security.Permission`-instances. If the VM lacks the permission requested, the security manager throws an `java.lang.SecurityException`. (If) There exists one global instance of the class `SecurityManager`, those of the Java-Api methods that are security-sensitive always request permission before they perform their actions.

Sigoa extends this security model by packing the basic method of `java.lang.SecurityManager`, `checkPermission` (and an overloaded variant of this method), into an interface `ISecurityInfo`. For each optimization job, such security information must be specified. The security information should of course not be created by the user – a process stage between the user process and the job system should build it. Events are a general security risk: they are most probably processed by other threads or even sent over a network. This could be used by the event records created by a user job to perform malicious work. The `getEventSource()`-method (see also Chapter 24) could be overridden to shut down the Java virtual machine, for example. `ISecurityInfo`

[1] `http://java.sun.com/javase/6/docs/technotes/guides/security` [accessed 2007-07-03]

therefore provides one additional method: `checkEvent`. With this method, all events generated by the optimization will be checked before being passed out of the job system. This way it is easily possible to limit the allowed events to instances of some known, secure classes.

The Java security manager must be replaced by an instance that also implements the `ISecurityInfo`-interface and defers its methods to those of the current job's security info record. This can easily be accomplished by providing the thread that executes the optimization job with the security info. This way, the security manager just needs to obtain the current thread instance and query its current security info record. This query is encapsulated in the `SecurityUtils`-class in the static method `getCurrentSecurityInfo`. More information on the structure of Sigoa execution threads can be found in Chapter 28 on page 413. The structure of the Sigoa security concept package is illustrated in Figure 25.1.

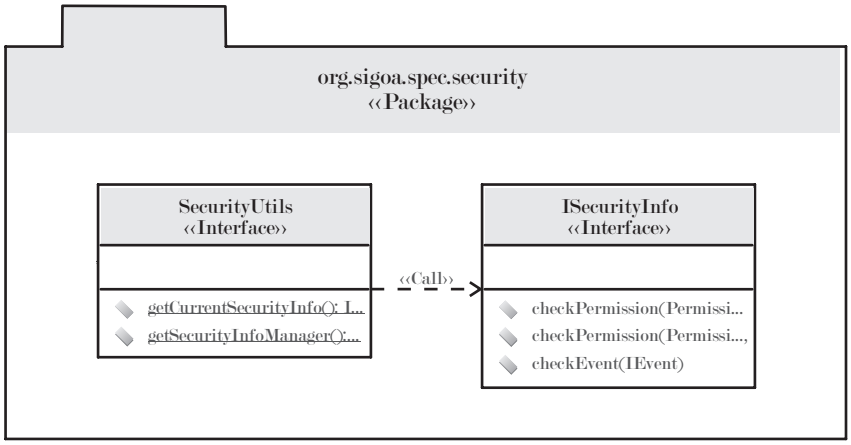


Fig. 25.1: The specification of the Sigoa security concept.

25.2 Reference Implementation

The reference implementation package for the security concept, `org.sigoa.refimpl.security`, is sketched in Figure 25.2.

The two basic classes `SecurityInfo` and `SecurityInfoManager` both inherit the behavior of their `checkPermission`-methods from the Java default security manager class `java.lang.SecurityManager`. Both implement the `ISecurityInfo`-interface, but have different semantics: The `SecurityInfo`-records can be created per-optimization job and represent the security information of the job they are created for, whereas the `SecurityInfoManager`

replaces the default security manager and defers its `checkPermission` calls to the current security info records (If no security info record is available, the inherited default security manager behavior is used).

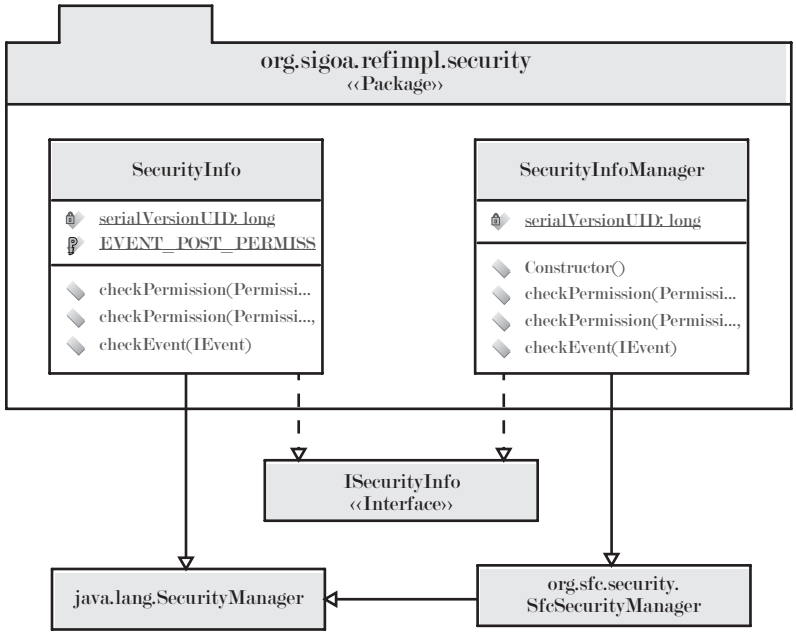


Fig. 25.2: The reference implementation of the Sigoa security concept.

26

Stochastic Utilities

Most optimization algorithms for complex problems are randomized and therefore need random numbers. Optimization algorithms are either population based or iterative or both. Thus it is often helpful to perform statistic analyses in order to determine the average fitness of the solution candidates of one population or the progress of the algorithm over time, for instance.

The mathematical foundations are explained in Chapter 35 on page 513 – the Sigoa stochastic utility package defines interfaces to encapsulate them.

26.1 Specification

The Sigoa stochastic utility package (depicted in Figure 26.1) contains specification for both, random number generators and statistical information presentation.

26.1.1 Random Number Generators

There are two different types of random number generators: those which create a sequence of random numbers according to a particular distribution function and those which provide multi-purpose functionality.

The first case is represented by the `IRandomNumberGenerator`-interface. Its `nextDouble`-method can subsequently be called and will return random numbers according to a certain distribution. It may be backed by a list containing all the numbers to be returned (and thus be deterministic) or, for example, return uniformly distributed random numbers obtained using specific algorithms.

If a random number generator creates repeatable sequences of random numbers, its `getSeed`-method may return a valid seed. The current seed of a (deterministic) random number generator determines all future numbers generated by it. Thus, if the seed is stored and restored later (with `setSeed`),

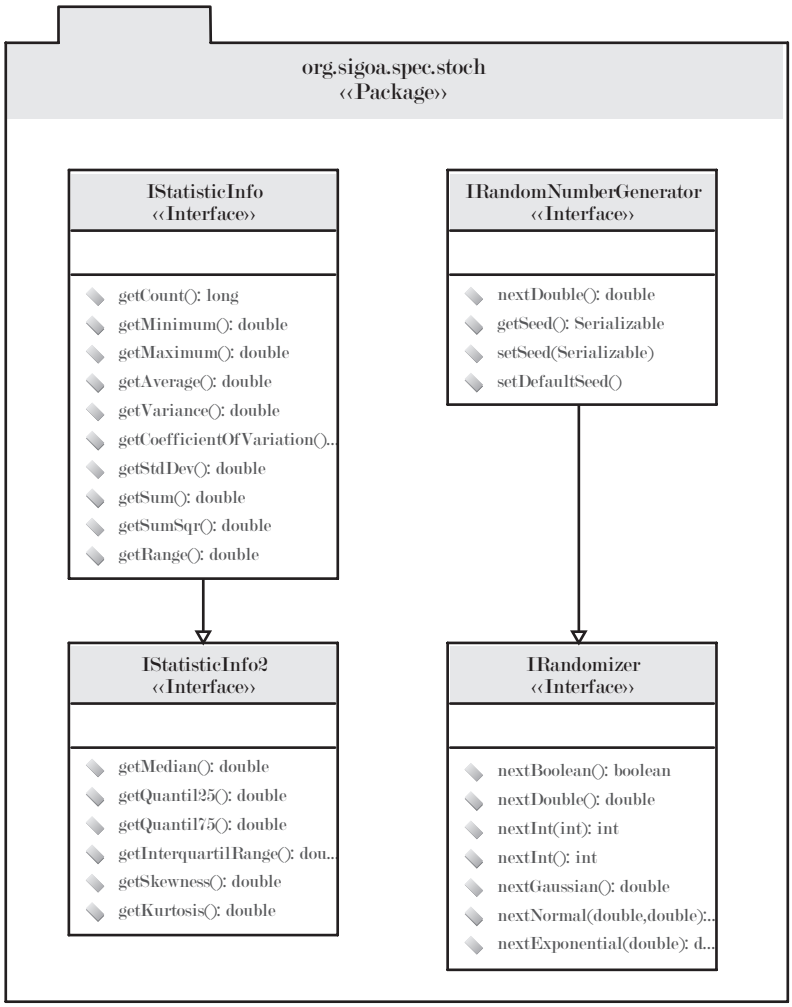


Fig. 26.1: The specification of the Sigoa stochastic utilities.

the sequence of pseudorandom numbers can be created twice. This way, experiments can be repeated under same *random* circumstances. The method `setDefaultSeed` initializes the random number generator with a constant default seed.

The `IRandomizer`-interface extends `IRandomNumberGenerator` to a multipurpose random number generator. Its `nextDouble`-method is now specifically bound to create uniformly distributed random numbers in the interval $[0, 1)$. It is an implementation of the parameterless $random_u()$ algorithm (see Definition 176 on page 565). The specification of this method is com-

patible with Java's `java.util.Random.nextDouble()`[1]. `IRandomizer` also defines other methods compatible to `java.util.Random`, allowing implementations to reuse the java class.

26.1.2 Statistic Data Representation

The most important statistical parameters are defined in Section 35.2 on page 519. The aim of the statistic utilities of Sigoa is to enable objects to provide such parameters in a structured manner. We define two interfaces, `IStatisticInfo` and `IStatsticInfo2`, for that purpose. While `IStatisticInfo` presents parameters that can be computed incrementally from a stream of data without needed access to all data elements (see Table 26.1), `IStatisticInfo` extends it by additionally representing those which can only be computed with immediate access to all data elements (seeTable 26.2).

Table 26.1: The methods of `IStatisticInfo`

method	stat. parameter	definition
`getCount`	count	Definition 140 on page 520
`getMinimum`	minimum	Definition 141 on page 520
`getMaximum`	maximum	Definition 142 on page 520
`getRange`	range	Definition 143 on page 521
`getSum`	sum	Definition 145 on page 521
`getAverage`	arithmetic mean	Definition 146 on page 521
`getSumSqr`	sum of squares	Definition 148 on page 522
`getVariance`	variance	Definition 147 on page 522
`getStdDev`	standard deviation	Definition 149 on page 523
`getCoefficientOfVariation`	coefficient of variation	Definition 150 on page 523

Table 26.2: The (additional) methods of `IStatisticInfo2`

method	stat. parameter	definition
`getSkewness`	skewness	Definition 154 on page 524
`getKurtosis`	kurtosis	Definition 155 on page 524
`getMedian`	median	Definition 156 on page 524
`getQuantil25`	$quantil_{100}^{25}$	Definition 157 on page 525
`getQuantil75`	$quantil_{100}^{75}$	Definition 157 on page 525
`getInterquartilRange`	interquartile range	Definition 158 on page 526

[1] `http://java.sun.com/javase/6/docs/api/java/util/Random.html` [accessed 2007-07-03]

The author currently does not know any method to compute the parameters *skewness* or *kurtosis on the fly* on a data stream. It may be possible that there is one and we'll find out some day. In this case we'll move this method to the `IStatisticInfo`-interface.

Notice that it may be very useful if an `IRandomNumberGenerator`, which works according to a certain probability distribution function, would also implement the statistic interfaces (`IStatisticInfo` or `IStatisticInfo2`). The random number generator could this way provide information about the parameters of the underlying distribution function.

26.2 Reference Implementation

The structure of the Sigoa stochastic utilities reference implementation package `org.sigoa.refimpl.stoch` is sketched in figure Figure 26.2.

26.2.1 Random Number Generators

The default random number generator type for Sigoa optimizers is implemented in the `Randomizer` class. It extends the Java class `java.util.Random` by the additional methods defined in the `IRandomizer`-interface. Other than instances of `Random`, those of `Randomizer` are not synchronized. All calls to them either have to happen in a single thread or need to by synchronized manually. Waive the synchronization in the `Randomizer` class increases its performance up to three times.

In order to obtain uniformly distributed pseudorandom numbers, a linear congruent generation scheme is applied, as discussed in Section 35.7.1 on page 561. In order to generate normally distributed random numbers in the method `nextGaussian`, the Box Muller method is applied as mentioned in Section 35.7.2 on page 562.

26.2.2 Statistic Data Representation

We define two classes for statistic data collection: `StatisticInfo`, which implements `IStatisticInfo`, and `StatisticInfo2` which implements `IStatisticInfo2`. `StatisticInfo` works on a stream of data: by repeatedly calling the `append`-method, the internal data is steadily updated and made available through the `IStatisticInfo`-methods. `StatisticInfo2` needs full access to all data elements in order to provide parameters like the *median* and the other two *quartiles*. Such records thus cannot be filled incrementally. To abstract from the data access in we define the interface `IValueExtractor`-interface. An implementation of this interface provides the means to access one special collection type. An implementation for `double[]` is already defined as a constant (`DOUBLE_ARRAY_VE`) in `StatisticInfo2`. Such instances can be passed to the

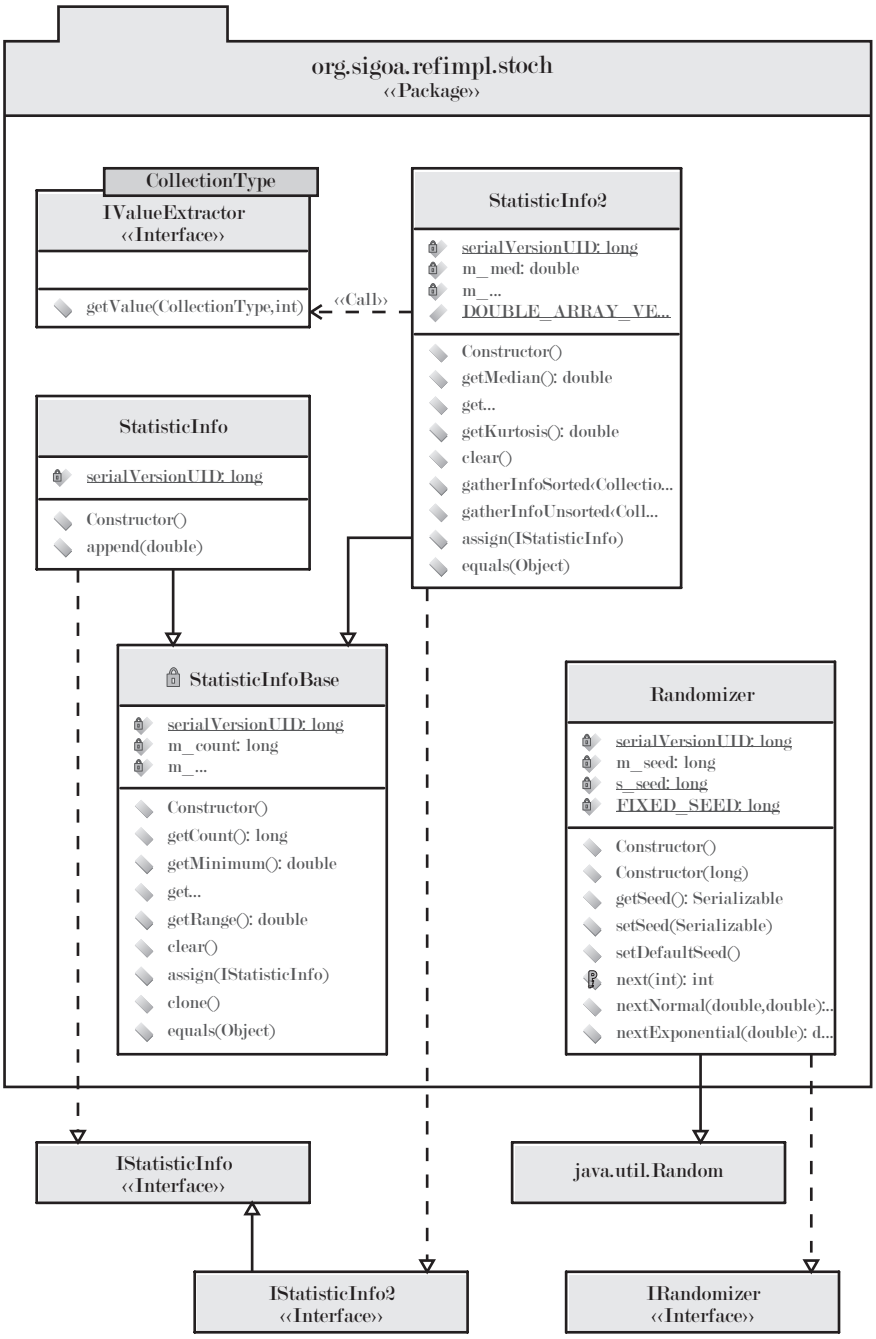


Fig. 26.2: The reference implementation of the Sigoa stochastic utilities.

methods `gatherInfoSorted` and `gatherInfoUnSorted` along with an instance of the collection-type they are intended for in order to read the data into the `StatisticInfo2`-object. Both, `StatisticInfo` and `StatisticInfo2`, inherit from the package-internal (invisible-for-the-outside-world) class `StatisticInfoBase` which additionally provides the means to copy the records (via `clone`), to clear them (via `clear`) to copy the content of one record into another, existing one (via `assign`) and to compare them for equality (using `equals`).

27

The Simulation Interface

Many optimization problems require complex evaluation of their individuals. Objective functions can often not be computed by just evaluating a mathematical expression. Instead, the individuals provide some sort of model which behavior needs to be determined using a complex simulation. Evolving a hardware/software-codesign [1101, 1102] may require simulating a VHDL[1]-design, the fitness of a wind turbine can only be determined with a complex physical simulation [1103], and a pattern recognition system grown [1104] needs to be checked against many test samples. There are even optimization algorithms that require human interaction in order to test their solution candidates like the evolution of music [1105] or images [1106]. If we apply multi-objective optimization, it may become possible that different objective functions require different simulations. If optimizing the design of a steel bar, one it is important to know how much weight a specific design can hold. Another objective function may further designs that are capable to withstand short but heavy impacts and thus needs another sort of simulation. A third objective would be to construct steel bars in a way that they can be stored and transported properly and therefore may test if they fit into certain containers or through the holes of some machines.

The Sigoa simulation interface allows arbitrary simulations to be managed and accessed by in a structured manner.

27.1 Specification

The specification package of the Sigoa simulation interface, `org.sigoa.spec.simulation`, is outlined in Figure 27.1.

[1] `http://en.wikipedia.org/wiki/Vhdl` [accessed 2007-07-03]

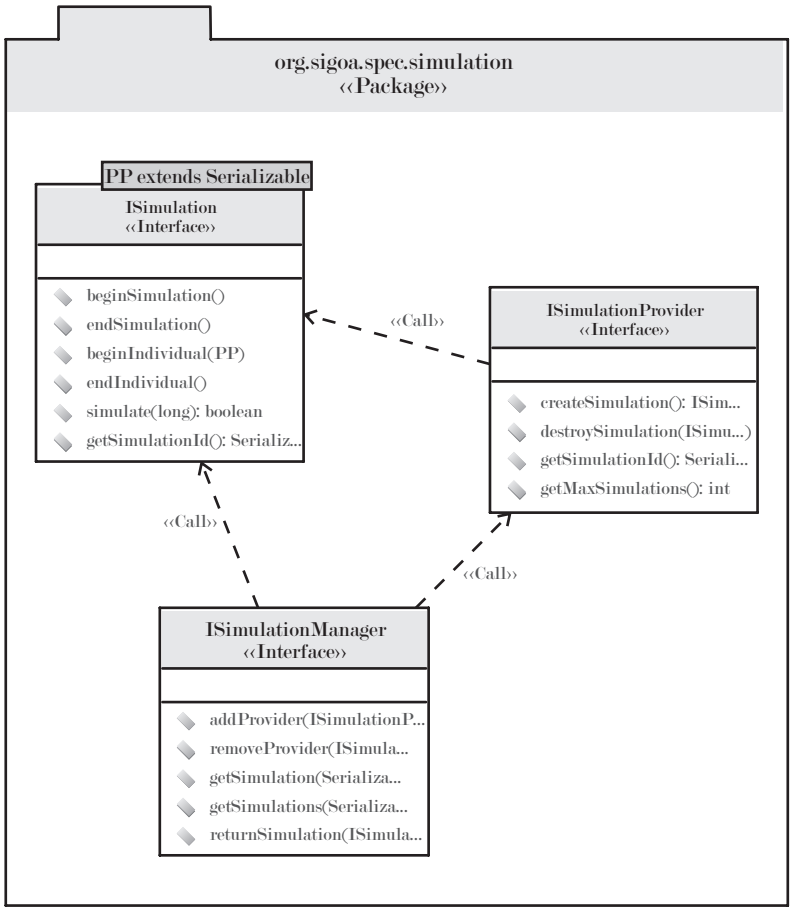


Fig. 27.1: The specification of the Sigoa simulation interface.

27.1.1 The Simulations

`ISimulation` represents the basic interface to a simulation. In Sigoa, simulations are reusable objects. Simulating an individual will always proceed in three steps.

1. Before a simulator is used, it is initialized with the method `beginIndividual` which also receives the solution candidate as a parameter to be tested as parameter. This method is called only once per individual.
2. The single simulations are initialized with a call to `beginSimulation`. The simulation now can set up internal data structures. This method is called for each simulation run and thus, possible multiple times per individual.

3. The `simulate`-method is called with a `long`-parameter containing the count of steps to perform in the simulation. This method may be called multiple times. Simulations in Sigoa can always be performed in discrete steps. `simulate` returns a `boolean` value which is `true`if further simulating would possible change the state of the simulation, and `false`if the simulation has come to a final, terminal state which cannot change anymore.
   If we simulate for example how a steel bar breaks, we have to use discrete time steps. `simulate` may be called several times. At some point in (simulated) time, the (simulated) steel bar breaks. After the bar is broken, further simulating makes no sense and `simulate` will always return `false`. Although this is a good method to detect whenever a simulation is done, it is not always a good idea to call `simulate` until it returns `false`. Sometimes you will want to limit the simulated time steps - if a program simulated, for instance, it may contain an unconditional loop and run infinitely. Therefore, can specify a count of steps to be performed in each call to `simulate`.
4. After the simulation has been performed long enough so that the objective functions could come to a conclusion about their values, a call to `endSimulation` tells the simulator that it may free all the simulation data.
5. Before being handed back to the simulation manager, the method `endIndividual` is invoked, again exactly once per solution candidate. This method should perform a final cleanup.

More information on how simulations are incorporated in the process of determining objective functions can be found in Section 31.1.4 on page 453.

27.1.2 Simulation Provider and Simulation Manager

Simulations come with a unique identifier, the *simulation id* which can be obtained using the `getSimulationId`-method. codeiliISimulationProvider-instances act as factories for simulations. A new instance of a simulation may be created using their `createSimulation`-method. If the (reusable) simulation created by this method is no longer needed, it may be freed with an invocation to `destroySimulation`. Each simulation provider is responsible for one single type of simulation which id it makes available by also implementing the `getSimulationId`-method (this method must be consistently return the same id as the `getSimulationId`-method of the simulation type represented). Using the simulation provider, many instances of a simulation type may be created and used in parallel. This can provide a speed-up if the optimizer runs on a multi-processor machine, for example. One could also imagine that the `ISimulation`-instances created are just local proxies for remote servers that actually perform the simulation work. It would make sense to create such a simulation-proxy for each remote server, so all servers could be equally loaded. To create more simulation-proxies however would not yield any gain of speed, if the computer running the optimizer had more processors. Therefore, the `getMaxSimulations`-method of the simulation provider can be used to limit the count of simulation instances created.

The third interface provided by the `simulation`-package is `ISimulationManager`. Instances of this interface are intended to manage multiple simulation providers and cache simulations, in order to provide them to other modules with high performance. Simulation providers may be added to a simulation manager using `addProvider` and removed using the `removeProvider`-method. A module that wants to perform a simulation calls `getSimulation`, passing in the id of the simulation required. The simulation manager will first check if it can provide the wanted simulation. If yes, it checks if it has a cached simulation instance of the given type available. If there is no unused, cached instance, a new one can be created and returned if the maximum count of active instances of the given type has not yet been exceeded. If (due to the `getMaxSimulation`-restriction) no new instance can be created, `getSimulation` will block until an instance in use has been returned. Simulations are *returned* to the simulation manager using `returnSimulation` when they're no longer needed (usually after a call to `endSimulation`). If, for some reason, more than one simulation type is needed in order to evaluate the objective functions, the `getSimulations`-method allows to query multiple, different simulations at once. It will block until instance of all simulation types requested are available. One should mention that the *caching* of simulations, as described here, is just a suggestion. The behavior of `ISimulationManager` could as well be realized without such a mechanism, though it could provide strong performance gains, especially if the simulation objects are quite heavy-weight (like one can imagine the simulators for steel bar physics to be).

27.2 Reference Implementation

The reference implementation of the Sigoa simulation interface can be found in the package `org.sigoa.spec.simulation`, sketched in Figure 27.2.

27.2.1 The Simulation

The base class `Simulation` realizes the `ISimulation`-interface. It does not provide any special simulation capabilities. The only thing is does is to store the solution candidate passed into `beginIndividual` into an internal variable and making it available through the method `getSimulated`.

The id system of the Sigoa simulation interface implementation is class based –the `getSimulationId`-method of an instance of `Simulation` will return its class.

27.2.2 Simulation Provider and Simulation Manager

`SimulationProvider`, the standard implementation for simulation providers, takes a `Class`-object as parameter in its constructor. This class is used as

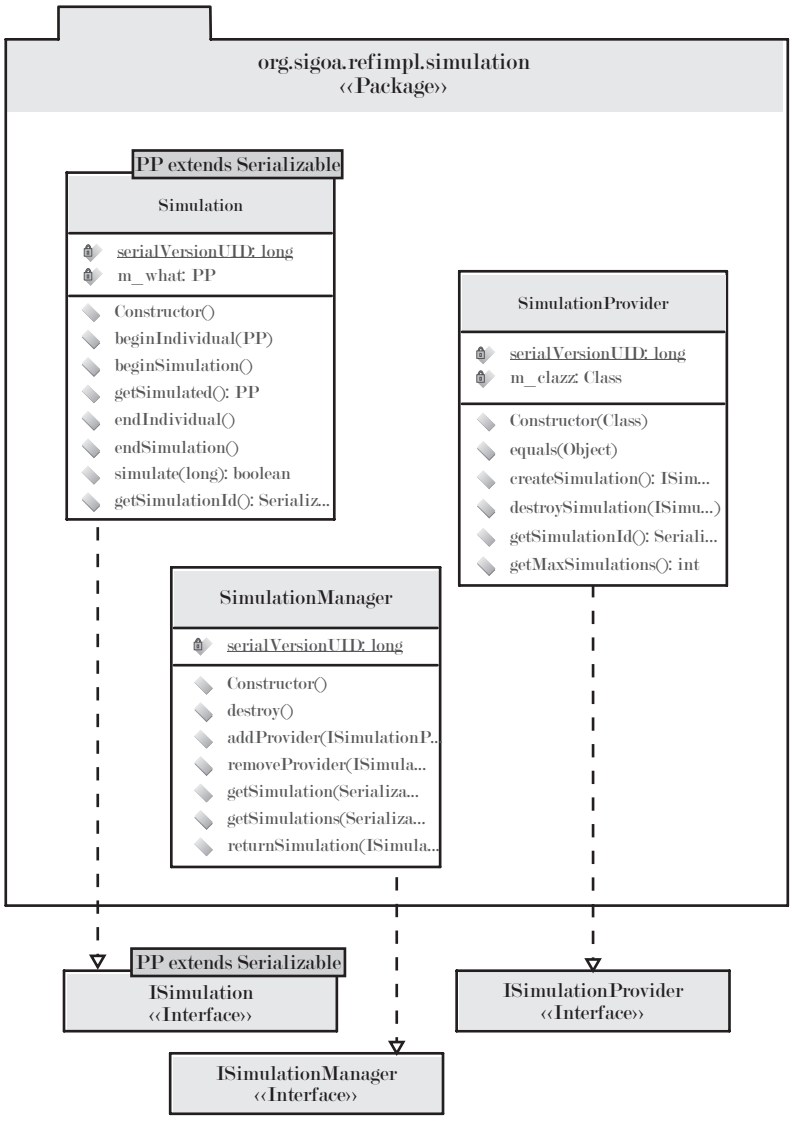


Fig. 27.2: The reference implementation of the Sigoa simulation interface.

simulation id and also in its `createSimulation`-method, where the instances of this class created with Java reflection[2] and returned. Thus, the class passed in to the constructor needs to be a class of an implementation of the `ISimulation`-interface which has a parameterless constructor itself.

The simulation manager default implementation `SimulationManager` provides the full functionality discussed in the specification section. Arbitrary simulation providers (`ISimulationProvider`-instances) may be added to it, not limited to instances of `SimulationProvider`. Their ids also may be arbitrary objects, not limited to classes.

27.2.3 Simulation Inheritance

Using classes as simulation ids however has a striking advantage: the support of inheritance. Figure 27.3 demonstrates a case where this feature can be used. Imagine we perform genetic programming and let algorithms grow. In order

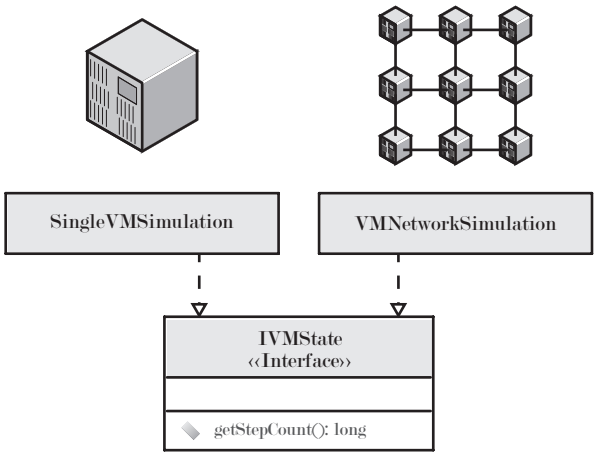


Fig. 27.3: Simulation inheritance in the reference implementation.

to determine their fitness for a specific purpose, we have to simulate them. Of course we want algorithms that run fast, so we add an objective function that minimizes the count of algorithm steps performed. Therefore, we add a function `getStepCount` to our simulation that returns this step count. When applying genetic programming, you will normally grow algorithms that are interpreted on one single (virtual) machine. What if we now want to evolve distributed algorithms? Well, instead simulating a single virtual machine running

[2] `http://java.sun.com/javase/6/docs/technotes/guides/reflection/` [accessed 2007-07-03]

the algorithm, we would have to simulate a whole network of such virtual machines. Again, algorithms that need lesser steps to come to a result should be preferred. Normally, we would re-implement the objective function discussed above. Using the simulation inheritance how ever, we can choose another option: we define an interface `IVMState` which provides the `getStepCount`-method. Now we implement this interface into both simulations, the one for the single virtual machine (`SingleVMSimulation` and into the one for the whole network `VMNetworkSimulation`), where it would simple add up the steps performed by all nodes in the simulated network. Our objective function now would only query a simulation that provides the `IVMState`-interface and could be applied to both cases. The simulation manager would look up which of the ids of the simulations it can provide is a class that is assignment-compatible to `IVMState`. Besides that more complicated example, a simple other application of simulation inheritance is to make simulations exchangeable. If a distributed algorithm should be simulated, we also need to simulate a network. Simulating a network can either be done very precisely using tools like *ns2*[3] or very simple without regarding latency, possible data loss and interference and such and such. If an objective function needs access to such network simulation data, we can simple define a base class for virtual machine network simulators that is derived for both cases. Now we can exchange the simulations whenever we want without interfering with the objective function which works on the base class for network simulation.

The reference implementation of the simulation manager supports such simulation inheritance. It whatsoever can, as already said, still process simulations with id types other from classes.

[3] The Network Simulator – ns-2 `http://www.isi.edu/nsnam/ns/` [accessed 2007-07-03]

28

The Job System

The job system is the backbone of the Sigoa optimization framework. It abstracts from parallelization and separates all aspects concerning this from the optimization algorithms. It provides a basic API with two main features:

1. Each optimization request is handled as one separate job which is processed by the job system. The job system may be able to assign worker threads/processors to the optimization according to the some internal strategy.
2. Optimization requests may often be considered as a set of tasks. In a population-based algorithm, the evaluation of the objective functions for each individual may be treated as such a task. The job system provides the optimization requests with an interface that allows them to execute such tasks in a transparent manner.

Transparency is one the main feature of the job system. The job system may own a simple queue for optimization requests and handle all of them sequentially. It may be implemented in a way that it can use multiple processors and thus execute tasks on each processor in parallel. The job system also may just be a gateway to a network of computational resources and directly distribute optimization requests to that network.

28.1 Specification

28.1.1 The Activity Model

Definition 85 (Activity). An activity is an entity that may run or may be run in parallel. It can (but does not necessarily need to) be executed on a (virtual) processor.

The job system package defines a default model for all Sigoa activities. This definition includes the states through which an activity may transcend

together with their semantics and the methods that lead to the state transitions (see Figure 28.1). The job system itself and all optimization requests are instances of Sigoa activities, and so will are all the other parallel-running parts like communication facilities and threads that regularly perform some tasks.

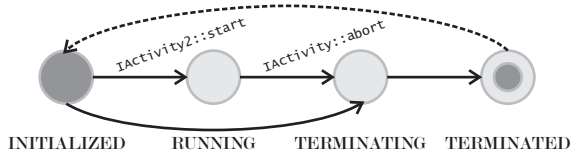


Fig. 28.1: The states and life cycles of an activity.

- `INITIALIZED` After the activity has been created, it will be in the state `INITIALIZED` (if it is an instance of the interface `IActivity2`). By calling the `start`-method, the activity will transcend to `RUNNING`. `start` normally can only be called for activities in the state `INITIALIZED`, if it is called for activities in another state a `java.lang.IllegalStateException` will be thrown. Therefore, `start` can be called at most once. Of course, an activity can be aborted (with a call to `abort`) directly before being started. In this case, the next state will be `TERMINATING`.
- `RUNNING` After being start, the activity performs its work while being in the state `RUNNING`. Notice that instances of `IActivity` which do not implement the `IActivity2`-interface enter this state directly after their construction. The `RUNNING`-state can only be left be transcending into the transitional state `TERMINATING`. This transition can be performed either because the activity has been aborted (by a call to `abort`) or because the activity has finished its work and wants to terminate itself.
- `TERMINATING` The transitional state `TERMINATING` denotes that the activity is terminating, either due to being aborted or due to having finished its work. The activity will dwell in this state until all sub-activities and all threads or processes started by it have terminated too. After everything run by the activity is finished, it will transcend to the state `TERMINATED`.
- `TERMINATED` This is the final state of an activity. It will be reached after all activities and all threads or processes started by it have terminated too. All other threads that wait for the activity (with `waitFor`) will now be notified and released. Normally, the activity will never leave this state. In some cases it may however bear advantages making heavy-weight activities reusable. In that cases, manual a transition to `INITIALIZED` may be allowed.

The basic interfaces of the activity model are illustrated in Figure 28.2. The possible states of an activity as discussed above are defined by the enumerate

`EActivityState`. `IWaitable` provides the method `waitFor` which waits for the activity to be terminated (i. e. reaches the state `TERMINATED`). If its `boolean` parameter `interruptible` is `true`and the waiting is somehow interrupted, this method will return `true`. Otherwise, it returns `false`. `IActivity` provides methods to query the current state of the activity.

- `isRunning` returns `true`if and only if the activity is currently in the state `RUNNING`.
- `isFinal` returns `true`if the activity has entered one of the two final states `TERMINATING` or `TERMINATED`.
- `isTerminated` becomes `true`if the activity has terminated, i. e. is in the state `TERMINATED`.

With the method `abort` a transition into the state `TERMINATING` can be initiated at any given time and is thus the way to end an activity manually in a graceful manner. Activities that can and must explicitly be started support the `IActivity2`-interface.

28.1.2 The Job System Interface

A job system can be accessed through the `IJobSystem`-interface. It extends the `IActivity2`-interface discussed before and the `IEventSource`-interface discussed in Chapter 24 on page 391). Inheriting from these two interfaces provides the functionality to start and stop job processing as well as a port to receive events generated by the system and the jobs that it runs.

A new optimization job is handed over to the job system with the `executeOptimization`-method. Additional to the job itself, this method needs some additional information which defines how the optimization job is handled. The method returns an instance of `IOptimizationHandle` which extends `IWaitable` and enables the user to wait for the job's completion. Furthermore, the job system assigns a unique identifier to each job it processes, which can be obtained by the `getId`-method of the optimization handle. This id will also be the source of all events created by the job systems (again, see Chapter 24). The job information passed to `executeOptimization` is packed into an `IJobInfo`-record (see Figure 28.3). This job information record provides three other information sets:

1. `IExecutionInfo` basically provides the information for the job system how to treat an optimization job. The basic version of this interface comes with one method: `getMaxProcessorCount` returns the maximum number of processors (working-threads) to assign to the job. The interface could be extended in order to support priorities, runtime restrictions and such and such.
2. We the `ISecurityInfo`-record is discussed in Chapter 25. The security information assigned to an optimization job defines what operations the optimization job is allowed to do.

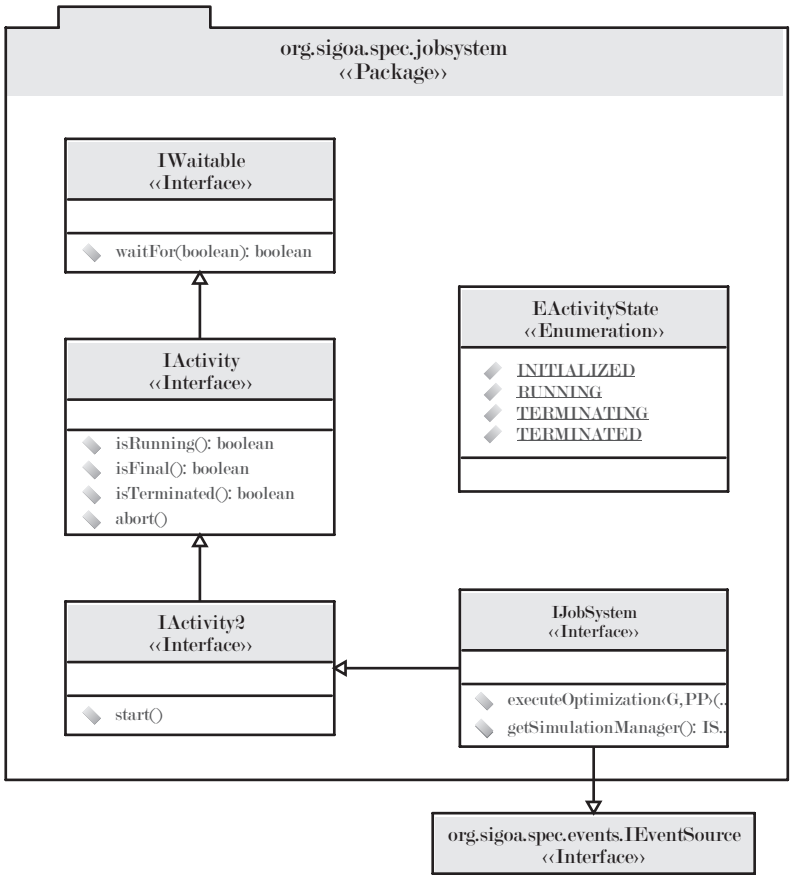


Fig. 28.2: The specification Sgioa activity model.

3. The `IOptimizationInfo`-record contains all the information that is used by the optimization job. It is discussed in Section 31.1.9 on page 460.

Each job system is equipped with a simulation manager (accessible via the `getSimulationManager`-method) by which it provides the optimization tasks it runs with access to simulations as discussed in Chapter 27 on page 405.

28.1.3 The Interface to the Optimization Tasks

The job system provides all optimization tasks that it runs with an internal access interface: the `IHost` (see Figure 28.4).

But how can an optimization job running access this interface? Well, if it runs its tasks sequentially, in parallel or in a distributed manner, each

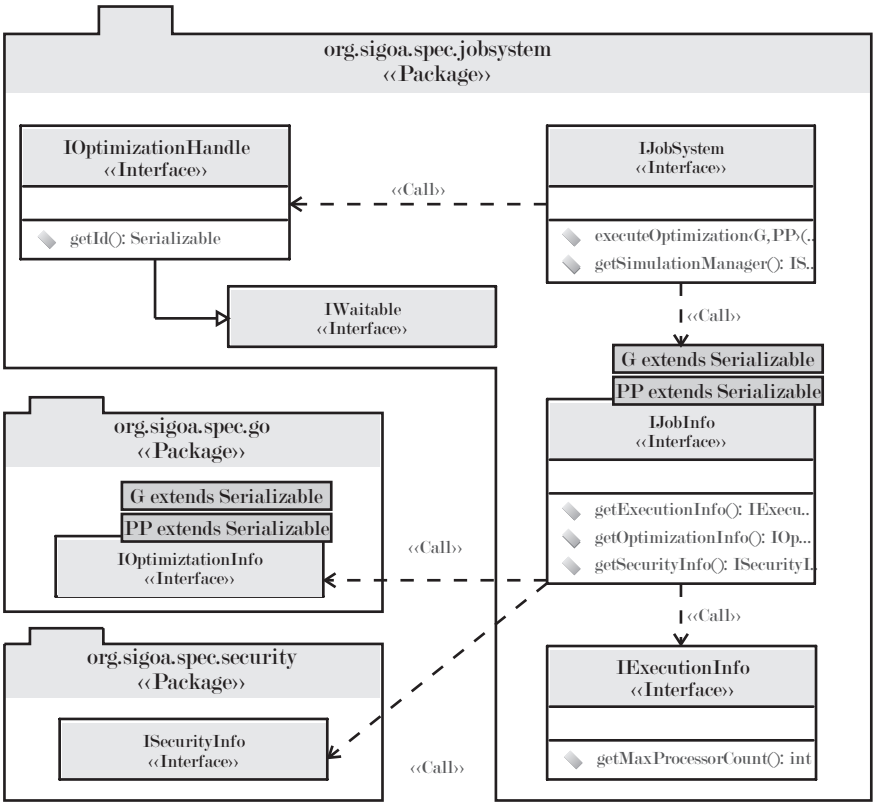


Fig. 28.3: The job system information record.

job system must use threads in order to do so. All these worker threads that possible execute optimization code must implement the `IHost`-interface. Similar to Java's `Thread.currentThread`-method there is a utility class called `JobSystemUtils` with a method `getCurrentHost`-method which returns the current host interface (which is identical to the current thread if called from within an optimization task, and `null` in all other cases).

The `IHost`-interface provides access to the `IJobInfo`-record passed to `executeOptimization` via its `getJobInfo`-method. Events posted to its `receiveEvent`-method are forwarded to the job system and can be received by all event listeners registered. The simulation manager of the job system can be obtained by calling `getSimulationManager` and the optimization job id which has been assigned by the job system and should be used as event source is made available through the `getOptimizationId`-method. Furthermore, the host thread also provides an instance of `IRandomizer` for random number generation to the jobs it runs (see Section 26.1.1 on page 399).

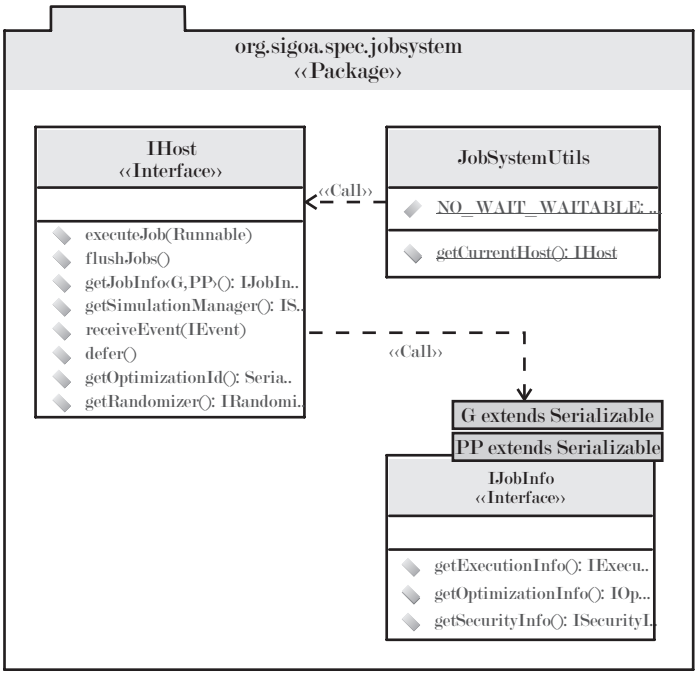


Fig. 28.4: The interface of the job system to the jobs.

The main task of the job system, to provide transparent parallelization, is constituted by three methods:

1. With `executeJob`, an optimization tasks can pass a job in form of an instance of `java.lang.Runnable`-instance to the job system. This sub-job then may be executed by the job system in any given way, sequentially, in parallel or even distributed (see Section 28.1.4 on the facing page) to another computational resource in a network. `executeJob` returns an instance of `IWaitable` which can be used to wait for the sub-job's completion.
2. If an optimization task creates many sub-jobs in a row, it normally does not want to use each single `IWaitable`-returned. Instead, it may call the `flushJobs`-method. This method waits until all sub-tasks created by the calling tasks have finished.
3. If an optimization job has nothing to do or waits for the completion of an outside event, it may release the processor temporarily by calling `defer` which has quite similar semantics like `Thread.yield`.

The interesting feature of this form of transparent parallelization is that sub-tasks can be nested arbitrarily. Since all tasks can access their host using `JobSystemUtils.getCurrentHost`, no additional parameters have to be passed to the tasks. Furthermore, since optimization jobs also implement the `java`

`.lang.Runnable`-interface, they can also be executed as sub-jobs, allowing optimization algorithms to use other optimization algorithms as back-ends.

The host is also the primary access point for the Sigoa security system (see Chapter 25 on page 395).

28.1.4 Notes on Distribution

If the job system works in a distributed manner, i.e. assigns the jobs it receives to different machines, a few things have to be regarded:

1. If a whole optimization job is to be send to a server which will perform it, the job information record also has to be sent along. Furthermore, there must be a proxy pipe installed to receive the jobs results (since the job would write them out on the wrong machine, see Section 31.1.8 on page 459).
2. It is completely possible that the server chosen to carry out the job misses some classes needed, so these need to be send to it and have to be loaded before.
3. If even sub-jobs (i. e. those received via the hosts with `executeJob`) are distributed, the optimization information record also has to be present on the executing machines. Also, the sub-jobs must not interact directly with other objects of the optimization job, since these would not be present there.

28.1.5 Using a Job System

In order to use a job system, one first needs to get an instance of `IJobSystem`. Reference implementations of this interface are discussed in Section 28.2.3 on page 422.

After creating this job system instance, we need to start it. This is done by invoking `start` since `IJobSystem` inherites from `IActivity2`.

Tasks are passed to the job system using the `executeOptimization`-method. This method takes an instance of `IOptimizer` and a `IJobInfo`-record as parameters. The job system now may queue the job internally and processes it as soon as there is free computation capacity. The optimization job can be restricted in the amount of processors it can get assigned to in parallel with the `getMaxProcessorCount`-method which is part of the `IExecutionInfo`-interface which in turn can be accessed via the method `getExecutionInfo` of the job info record.

When the job system has performed everything we want it to perform we can either let it run infinitely or abort it using `abort`. Invoking this method will order the job system to terminate all its internal threads, to abort all tasks currently performed or in an internal task queue, and to free all internal stores. If we want to wait until the job system is completely terminated, we can use the method `waitFor` which returns after all internal activities have terminated.

28.2 Reference Implementation

28.2.1 The Activity Model

The reference implementation of the Sigoa job system including the activity model can be found in the `org.sigoa.refimpl.jobsystem`-package illustrated in Figure 28.5. A handy base class which can be used to derive

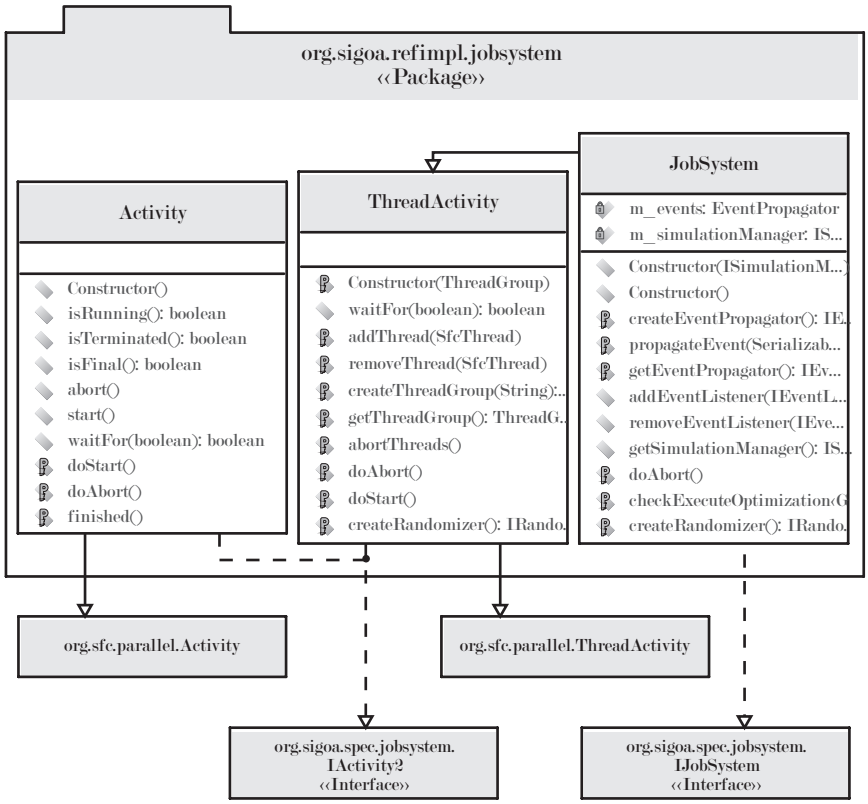


Fig. 28.5: The reference implementation of the Sgioa activity model.

activities in the Sigoa environment is the class `Activity` which implements the `IActivity2`-interface. It is derived directly from the Sfc-class `org.sfc.parallel.Activity` which realizes the exact same activity model as specified in Section 28.1.1. Here we will discuss the methods inherited from this utility class and illustrated in Figure 28.5. A new instance of `Activity` is always in the state `INITIALIZED`. The internally synchronized method `start` first checks whether the activity is currently in the state `INITIALIZED`. If so,

it invokes the protected method `doStart`, otherwise it throws an `java.lang.IllegalStateException` (see Section 28.1.1 on page 413). `doStart` can be overridden by derived classes in order to provide specific startup behavior. `abort` also checks the internal state. If it is `INITIALIZED` or `RUNNING`, the state will be set to `TERMINATING` and `doAbort` will be invoked. Nothing will be done otherwise (assuming that the activity is either already shutting down (terminating) or terminated fully). This `doAbort`-method must be overridden by derived classes in order to provide shutdown behavior, and, most important, call `finished`. The also protected method `finished` sets the state of the activity to `TERMINATED` and releases all threads that wait for this activity using the `waitFor`-method. If the activity may terminate itself without enforcing the use of `abort`, it must call `finished` at the end as well. This would for example be the case if the activity is a thread that does some work and terminates after that work is finished.

The class `ThreadActivity` inherits its behavior from the Sfc utility `org.sfc.parallel.ThreadActivity`. This class realizes the specification of an activity that consists of any number of threads. Threads can be added to the activity using the protected method `addThread` and removed via `removeThread`. The method `doStart` will start all threads added to the activity and `doAbort` will stop them. `waitFor` will not return until all threads have terminated. `ThreadActivity` also stores a thread group which can be accessed via `getThreadGroup`. This thread group can either be specified in the constructor, or is otherwise created by the method `createThreadGroup`.

28.2.2 The Job System Base Classes

The class `JobSystem` can be used as base for `IJobSystem`-implementations. It is a subclass of `ThreadActivity` and provides additional utilities:

- It uses an internal instance of `EventPropagator` (see Section 24.2 on page 393) which manages the event listeners that register to the job system. This internal event propagator can be accessed via `getEventPropagator`. Events are send to the listeners over the event propagator using the `propagateEvent`-method which takes the id of the sending optimization job as well as the event itself as parameter. The `org.sigoa.spec.events.IEventSource`-interface (which is one of the super-interfaces of `IJobInfo`) is realized using this internal event propagator: the `addEventListener` and `removeEventListener` forward to it. The event propagator is created by the `createEventPropagator`-method which can be overridden if an instance of a class derived from `EventPropagator` should be used instead of the default implementation.
- All job systems need to create at least one thread in order to obey to the specification properly. All threads of a job system should belong to a single thread group. This thread group is provided by the `getThreadGroup`-method inherited from `ThreadActivity`. If a normal `java.lang.ThreadGroup`

does not suffice for some reasons, the method `createThreadGroup` may be overridden in order to provide another class.

- A job system has to offer a simulation manager via the method `getSimulationManager`. This simulation manager can either be passed in the constructor of `JobSystem` or it uses the default simulation manager implementation (`org.sigoa.refimpl.simulation.SimulationManager`).
- The `checkExecuteOptimization`-method which can be used by derived classes performs default tests on a job to be executed. If everything is ok it simple returns. In the case of an error, it throws the correct exception.
- The host threads need to provide random number generators in the form of instances of `IRandomizer`. Therefore `JobSystem` provides the method `createRandomizer` which creates such a randomizer. This method returns a new instance `Randomizer` (see Section 26.2.1 on page 402).

The reference implementation of the job system information records is sketched in Figure 28.6. The `getMaxProcessorCount`-method of `ExecutionInfo` always returns `java.lang.Integer.MAX_VALUE` so optimization jobs will utilize all available processors. The default implementation of the `IJobInfo`-interface, `JobInfo`, comes with two constructors. The first one takes three parameters: an instance of `org.sigoa.spec.security.ISecurityInfo`, an instance of `org.sigoa.spec.jobsystem.IExecutionInfo`, and an instance of `org.sigoa.spec.go.OptimizationInfo`. These parameters can later be accessed through the `getSecurityInfo`, `getExecutionInfo` and `getOptimizationInfo`-methods. The second constructor only takes the mandatory `IOptimizationInfo`-instance as variable and creates new instances of `ExecutionInfo` and `org.sigoa.refimpl.SecurityInfo`.

Another class of information objects provided is `JobId`. As already stated in Section 28.1.2 on page 415, unique identifiers will automatically be assigned to new jobs by the job system. `JobId` represents a 128-bit identifier which can be used for that purpose. Instances of this class will always be unique inside a single Java virtual machine and have high probability of also being unique in a set of different Java VMs. `JobId` also re-implements the `equals`, `hashCode`, and `toString`-methods so job ids are comparable and have a human readable representation.

28.2.3 Job System Implementations

The Sigoa reference implementation provides two basic job systems as shown in Figure 28.7. The `SingleProcessorJobSystem` uses one single worker thread and thus executes all optimization jobs sequentially whereas the `MultiProcessorJobSystem` is able to start multiple workers and can perform different jobs (and sub-jobs) in parallel.

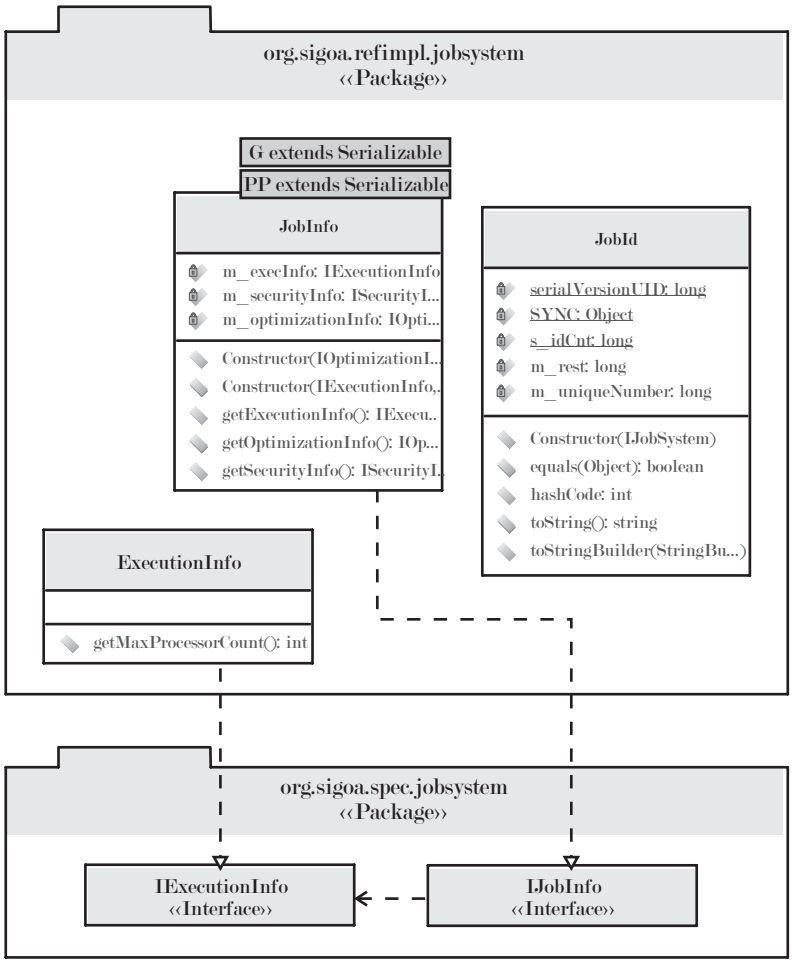


Fig. 28.6: The implementation of the job system info records.

The Single-Processor Job System

On computers with only one processor or in Java virtual machines that make use of only one processor, a job system that also uses only exact one processor will perform best. Furthermore, managing concurrent worker threads is more complicated and needs more control data structures so it makes only sense in an environment where parallelization will yield substantial performance gain. Such data and control structures are spared in the `SingleProcessorJobSystem` which exists to provide maximum performance in cases where only one processor is available for executing the optimization jobs.

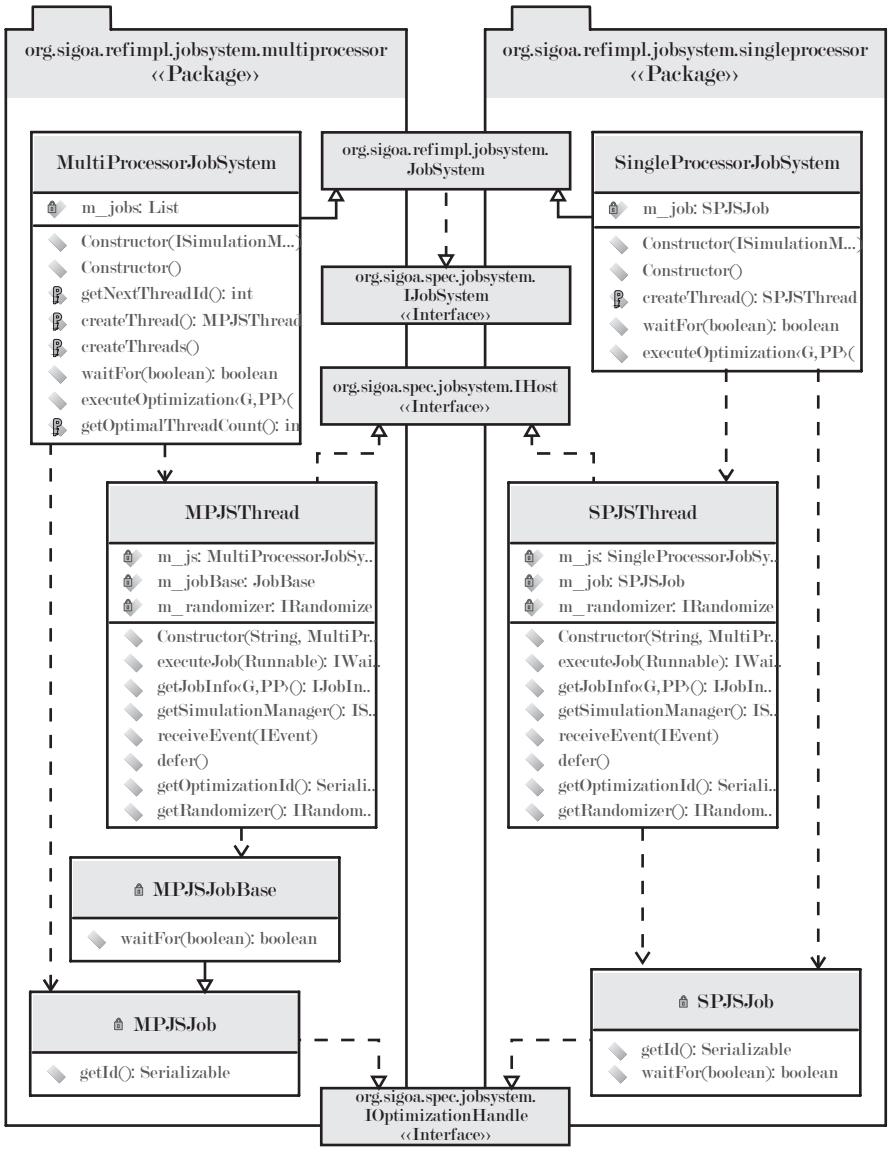


Fig. 28.7: The two basic job system reference implementations.

All such optimization jobs passed to the single-processor job system are put in a simple LIFO[1]-queue. Each `SingleProcessorJobSystem` uses exactly one `SPJSThread` which performs all the work. The worker thread of the single-processor job system is created using the internal `createThread`-method. It can be overridden in order to provide worker threads with enhanced behavior.

The `SPJSThread` will take the jobs out of the job system's internal job queue and work on them. Furthermore, it also represents the interface to the optimization jobs by implementing the `IHost`-interface (see Section 28.1.3 on page 416). Through this `IHost`-interface, an optimizer running may send events or execute sub-jobs. Of course, since the job system is intended to run as a single thread, the sub-job executing method `executeJob(java.lang.Runnable)` simple invokes the `Runnable`'s `run`-method directly. Since all sub-jobs are executed directly, the `flush`-method of `SPJSThread` does nothing – there can never be sub-jobs to wait for. Another features is its internal exception handling: if a job causes a `java.lang.Throwable`, the exception is caught, put into an `IErrorEvent`-record[2] which is propagated to all event listeners subscribed to the job system. The erroneous job then is terminated.

A job passed to a single-processor job system is internally represented by an instance of `SPJSJob`. Instances of this internal class realize the `org.sigoa.spec.jobsystem.IOptimizationHandle`-interface and are therefore returned by `executeOptimization`. The application that handed the optimization job over to the job system can wait for its completion using this handle.

The optimization Ids generated by this job system are instances of the class `JobId` discussed in the previous section.

The Multi-Processor Job System

The `MultiProcessorJobSystem` uses multiple worker threads in order to provide high performance in environments where more than one processor is available. When it is constructed, it uses the internal `getOptimalThreadCount` method to determine the count of threads to use. In the standard implementation, this method returns `Runtime.getRuntime().availableProcessors()`–the count of available to the Java virtual machine. It then creates that many `MPJSThread`s via its `createThread`-method. Like in of `SingleProcessorJobSystem`, the worker threads implement the `IHost`-interface and act as a gateway to the job system for the running optimization jobs. It also comes with the same internal exception handling, and, exactly like `SPJSThread`, inherits from `DefaultThread` discussed in Section 33.1.1 on page 497.

The optimization jobs are represented by instances of `MPJSJob`-instances which implements the `IOptimizationHandle`-interface. Unlike in the jobs of the single-processor job system they are not kept in a LIFO-queue but in a list to which processing power is assigned in a scheduling algorithm similar to

[1] Last In First Out, `http://en.wikipedia.org/wiki/LIFO` [accessed 2007-07-03]
[2] see Section 24.1 on page 391

Round Robin[3]. The `MPJSJob` however are instances of another internal class, `MPJSJobBase`. Instances of this class represent a single sub-job but also contain a LIFO-queue of again, instances of `MPJSJobBase`. This allows the optimization job to execute sub-jobs, which in turn can again execute sub-jobs and so on. The `flush`-method of `MPJSThread` waits until all sub-jobs of the calling (sub-)job are executed. `executeJob` always returns the instance of `MPJSJobBase` assigned to the sub-job, since `MPJSJobBase` implements the `org.sigoa.spec.jobsystem.IWaitable`-interface. The `waitFor`-method of this interface will return when the sub-job it is belongs to has finished. This way, an optimization tasks can wait for specific sub-jobs to be completed.

[3] `http://en.wikipedia.org/wiki/Round-robin_scheduling` [accessed 2007-07-03]

29

The Pipeline System

Optimization algorithms in the Sigoa framework are realized as a concatenation of pipes according to the pipes and filters[1] design pattern [1107, 1108, 1109]. Pipes and filters means basically that a stream of data is processed in a form very similar to the way work is done at an assembly line: there are several stations (filters) in a row each performing one special task, illustrated in Figure 29.1. These stations are connected by pipes which transport the data. If we apply this principle to genetic algorithms as depicted in Figure 3.1 on

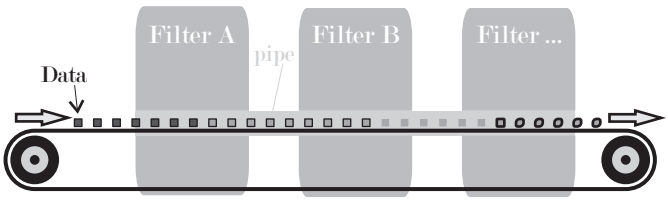


Fig. 29.1: The pipes and filters software design pattern.

page 119, for example, we can model the selection, reproduction and evaluation phases as filters. The evaluation filter then would determine the fitness of the incoming solution candidates. The selection filter only allows so and so many individuals to pass. In the reproduction phase, the solution candidates are replaced by their offspring. The only difference to the normal pipes and filters approach is that the solution candidates that come out of the end of the pipeline are put back into its beginning. Such a realization of an evolutionary algorithm in the form of a pipes and filters-based system is illustrated in Figure 29.2.

[1] `http://de.wikipedia.org/wiki/Pipes_and_Filters` [accessed 2007-07-03]

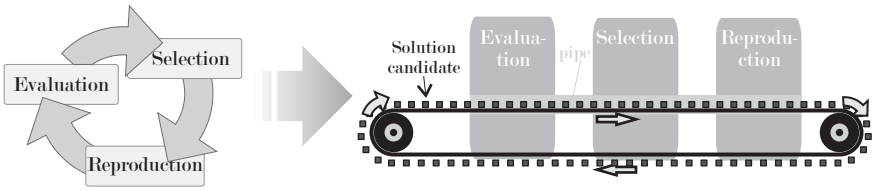


Fig. 29.2: A evolutionary algorithm realized with pipes and filter.

29.1 Specification

The pipeline in Sigoa transports solution candidates in form of individual records (`org.sigoa.spec.go.IIndividual`, see Section 31.1.1 on page 445). Each filter in has an entry and an exit interface. The stream of individuals enters the codeiliIPipeIn-interface through the `write`-method. Whenever all solution candidates have been written, `eof` is called in order to tell the filter that it has received a complete chunk of data. The codeiliIPipeOut-interface provides the methods codeilisetOutputPipe and codeiligetOutputPipe. Using `setOutputPipe`, an `IPipeIn`-instance can be specified to which the `IPipeOut`-object should write all the individuals that come out of it to. Both interfaces are combined in `IPipe`. Implementors of that interface are the stations of the pipe, the filters. An arbitrary count of `IPipe`-instances can be concatenated together and form a pipeline. There are many options what an instance of `IPipe` can do with the individuals written to it, some of them are:

1. it can modify the incoming individuals and put out the result
2. it can copy them directly to its output while only gathering statistical information
3. it can have secondary output destinations and fork the individual stream
4. it can have secondary output destinations and copy the individual stream to both
5. it may buffer all individuals and process all them together at once when receiving a call to `eof` before them putting out

Calls to `eof` will normally be propagated from pipe stage to pipe stage. Multiple instances of `IPipeOut` can have set the same instance of `IPipeIn` as output destination – joining multiple individual streams into one. In this case it may be useful not to propagate all the `eof`s but only one single `eof` after all input pipes have reported the end of their input data. It should be noted that an invocation `eof` does not mean that no data will follow in future – it simple stands for the end of one data chunk. A sequence of new individual records can be written right after calling `eof` and after its end, `eof` can be called again.

The specification package of the Sigoa pipeline system, `org.sigoa.spec.pipe`, is outlined in Figure 29.3. Additional to the default pipeline system it contains the `IPipeSource`-interface that may be implemented by all entities

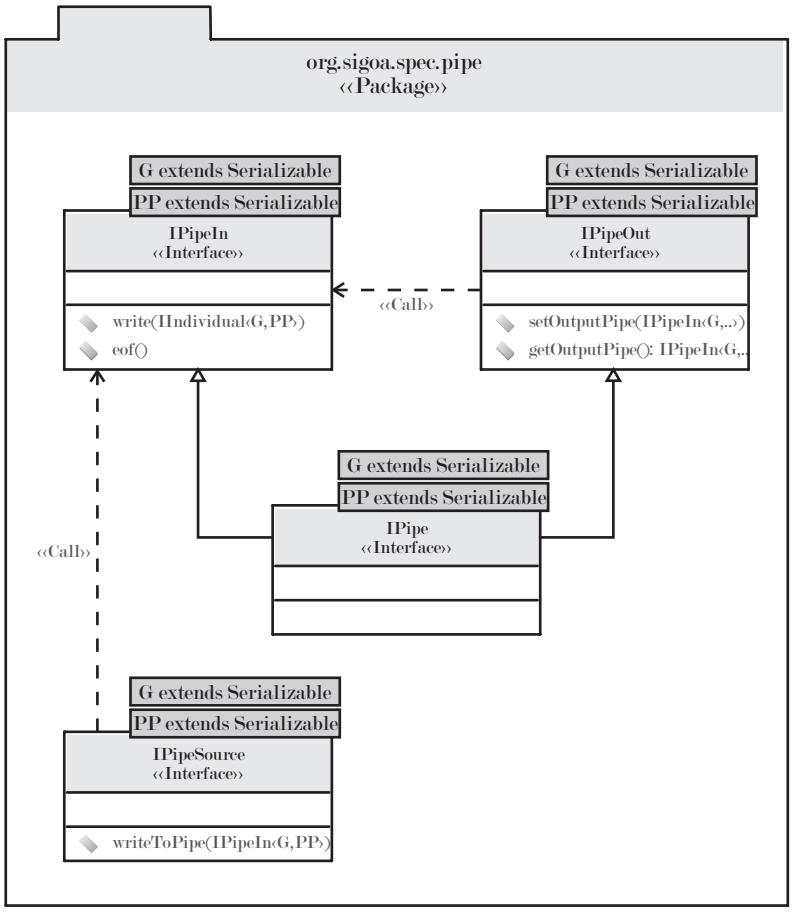


Fig. 29.3: The specification of the Sigoa pipeline system.

that own a set of solution candidates which can be written to an instance of `IPipeIn`. On the other hand, there also may be entities that solely implement `IPipeIn`. A collection of individuals which implements this interface can mark the end of a pipeline – all solution candidates can be appended to it that way without the need of passing them along to a subsequent pipe stage.

29.2 Reference Implementation

29.2.1 Basic Classes

The basic classes of the pipe system reference implementation are depicted in Figure 29.4. The interface `IPipeOut`, which represents the output-end of a pipe

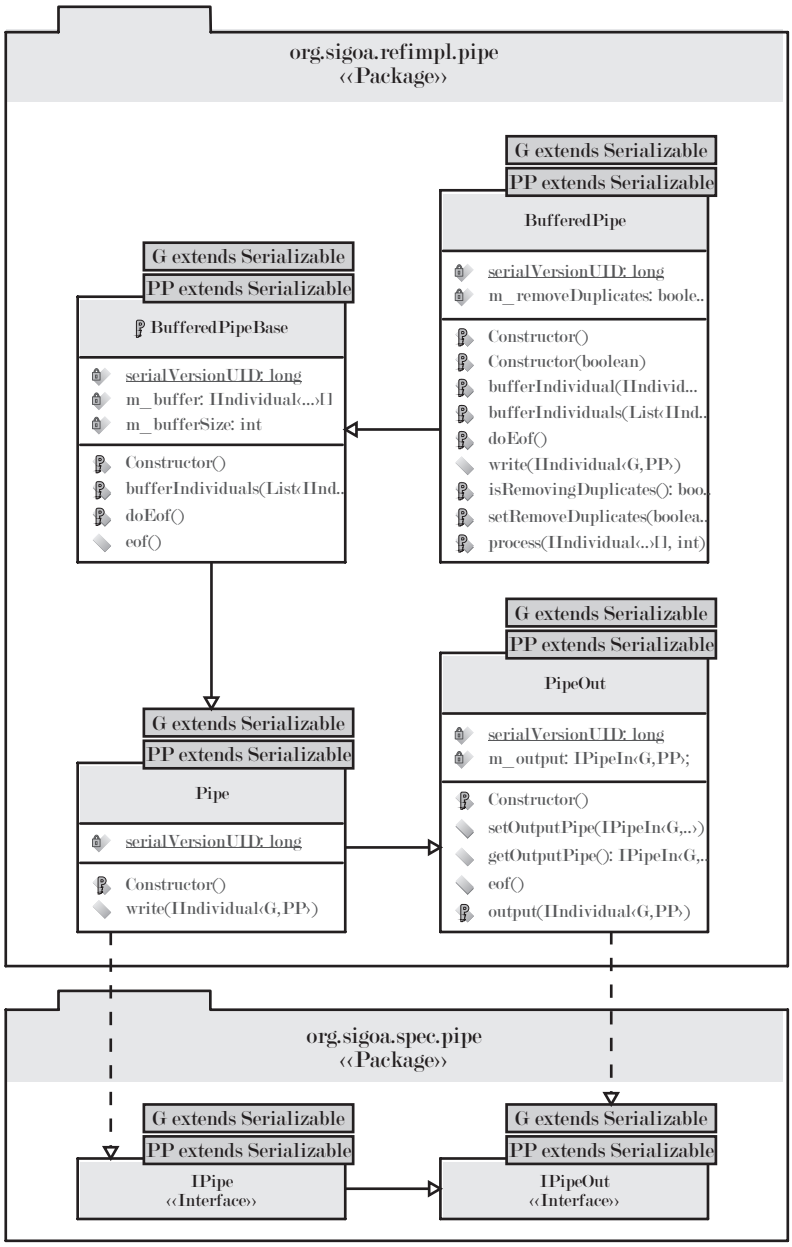


Fig. 29.4: The classes of the pipeline system reference implementation.

stage, is implemented in the `PipeOut`-class which additionally comes with the methods `eof` and `output`. Both are used for the communication with the next pipe stage (which can be set by `setOutputPipe`). The method `output` writes an individual record to it (by calling the `write`-method) while `eof` invokes its `eof`-method, telling that one stream of individuals is finished. (Remember at this point that a call to `eof` does not mean that no future data will follow, see Section 29.1 on page 428 for more information).

The class `Pipe` extends `PipeOut` by also implementing the `IPipe`-interface. It is therefore equipped with the method `write`, which simple passes the individual it receives directly to the inherited `output`-method. The default behavior of all classes that are descendants of `Pipe` is thus

- propagate calls to `eof` directly to the next pipe stage
- propagate individuals received via `write` directly to the next pipe stage

In order to provide some special behavior, like mutating all incoming individuals, the method `write` has to be overridden in subclasses.

Pipes realize sequential processing. One individual is worked on, passed to the next stage, and then the next individual is dealt with. Under many circumstances, such a sequential approach is insufficient. If we want to perform a selection for example, we need to compare the solution candidates to determine which one survives. In order to do so, we must access the whole set of individuals of a generation. At this point, the method `eof` comes into play. As already stated, it denotes the end of a chunk of solution candidates – the end of a generation, for instance. We now buffer all incoming individuals (that we receive through out `write`-method) internally and wait until `eof` is called. We can now process them as a whole in `eof` and pass on the selected ones to `output` (which then invokes `write` of the next pipe stage). After all work is done, we clear our buffer and propagate the `eof`-call, again to the next pipe stage.

This behavior is realized by the class `BufferedPipe`. It stores all individuals it receives in an internal array, optionally removing duplicates[2] on the way. When `eof` is invoked, it calls the `protected` method `doEof` which in turn invokes `process` if at least one individual was buffered. After that, the internal array is cleared and `eof` of the next pipe stage is invoked. `process` must be overridden in subclasses in order to provide the wanted behavior, like performing a selection, for example. `isRemovingDuplicates` tells whether the automatic duplicate removing is applied, which can be turned on and off with `setRemoveDuplicates`. Additional to the now buffering `write`-method, a list of individuals can be buffered using `bufferIndividuals`.

To ease the editing of long concatenations of pipes, the class `Pipeline` is introduced. Pipes attached naturally to each other form linked lists[3]. `Pipeline`

[2] individuals that are referencial identical, are that same instances, do not mix up with value equality

[3] `http://en.wikipedia.org/wiki/Linked_list` [accessed 2007-07-03]

presents this list through the Java Collections Framework[4]-interface `java.util.List`, as sketched in Figure 29.5.

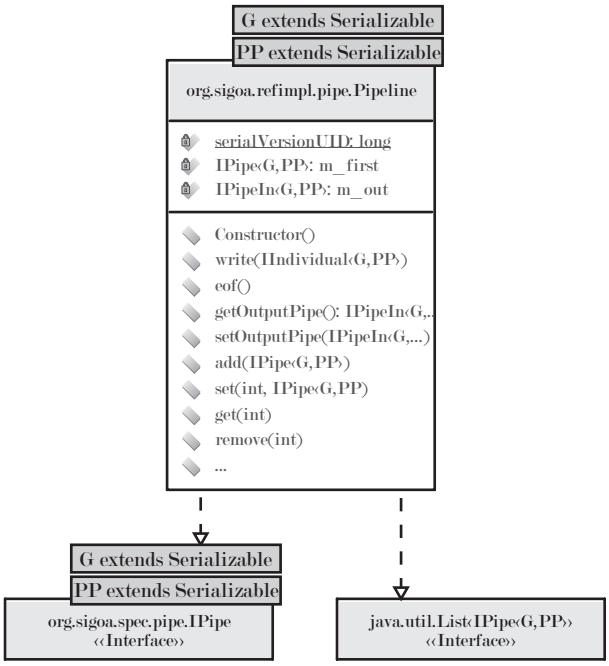


Fig. 29.5: The utility class `Pipeline`.

29.2.2 Some Basic Pipes

The package `org.sigoa.refimpl.pipe` provides some predefined pipes (illustrated in Figure 29.6) for reoccurring tasks. The simplest one is the `NoEofPipe` which alters the behavior of the normal pipe by not propagating the `eof`-calls. This can be useful if solution candidates enter the optimizer from the outside in an asynchronous way, from an internet connection, for example. They can then be integrated into the optimization process without interfering with its normal flow.

While this was an instance for union of two individual flows, it is sometimes useful to fork a flow into two pipes. An evolutionary algorithm with elitism (see Definition 37 on page 55) for example will fork the population into two streams. One of them will be used for the next generation, the other one will be

[4] `http://java.sun.com/javase/6/docs/technotes/guides/collections/` [accessed 2007-07-03]

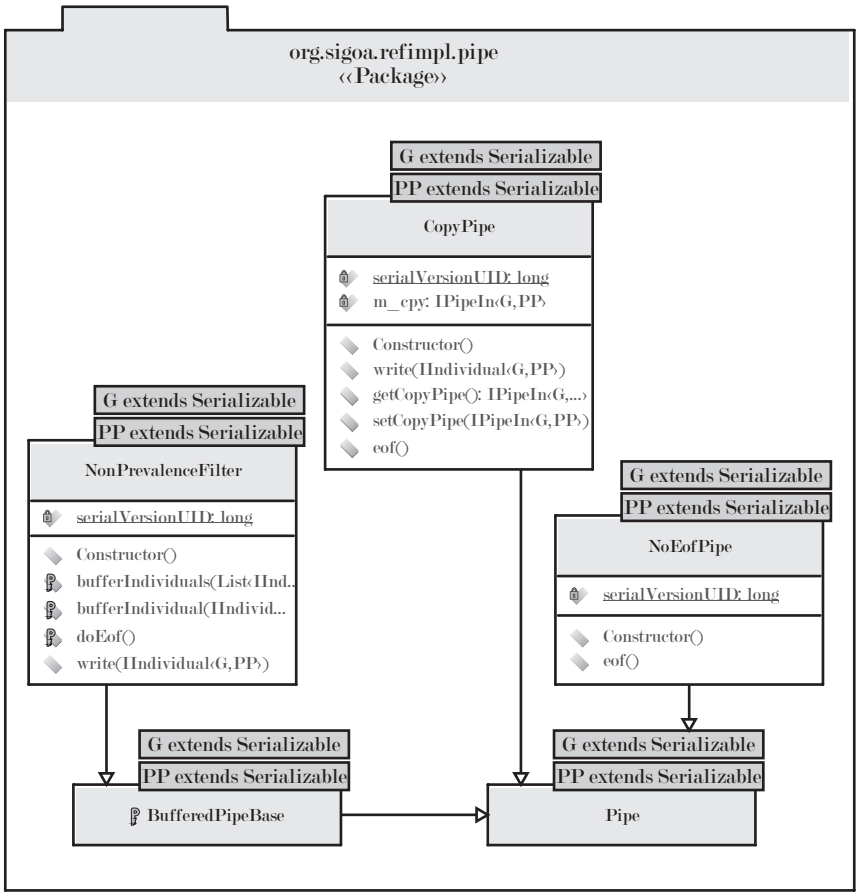


Fig. 29.6: Some other pipeline classes.

stripped of all prevailed individuals (see Section 1.3.5 on page 20), letting only the non-prevailed ones remain, and stored into the archive. For the purpose of forking an individual stream, the `CopyPipe` is used. All incoming individuals are copied to both, the next pipe stage and to a secondary destination which can be obtained with `getCopyPipe` and set by `setCopyPipe`.

In order to allow only the non-prevailed individuals to pass, a `NonPrevalenceFilter` can be used. It first buffers all the solution candidates of one chunk, deletes the prevailed ones and then writes the rest to the next pipe stage.

29.2.3 Pipes for Persistent Output

The persistent output of a stream of individuals is another task special pipes are provided for (see Figure 29.7). They could be attached to the optimizer to

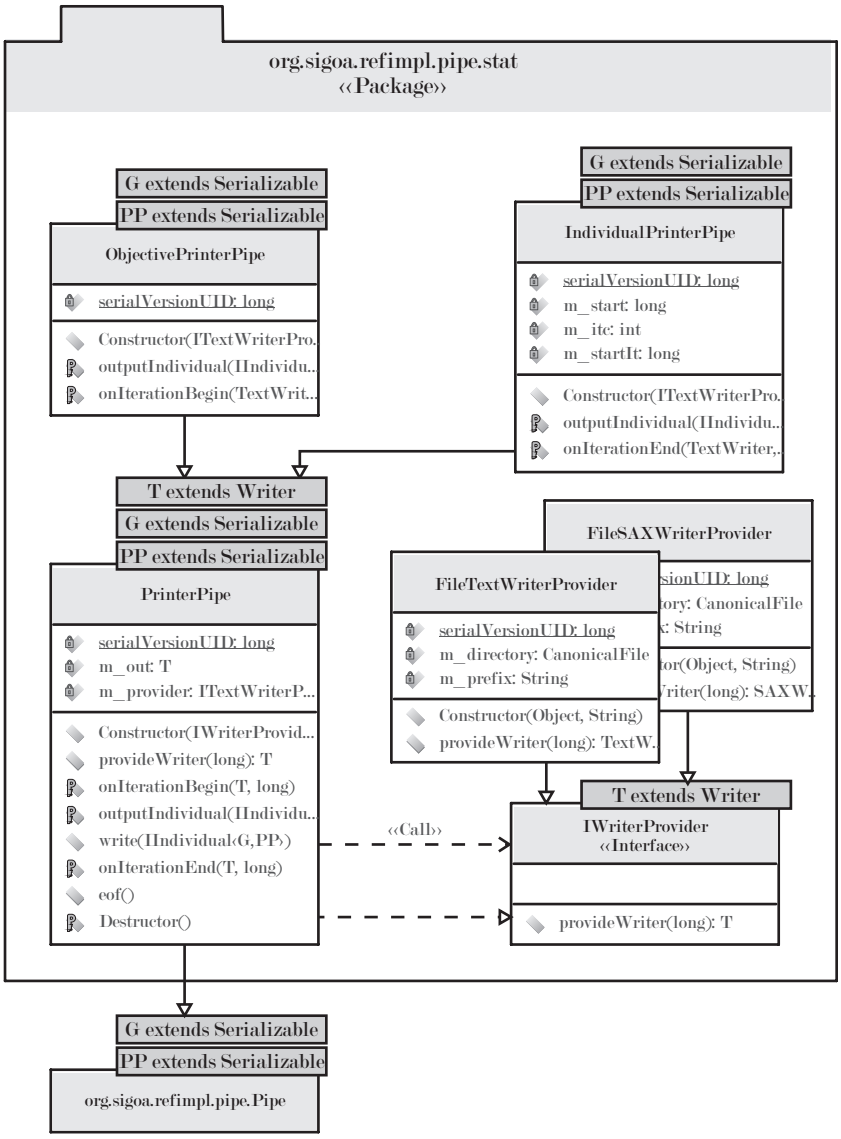


Fig. 29.7: Pipe stages that print out statistical data.

store its results or be plugged right into the optimization algorithms internal pipe to print all the solution candidates evaluated in its course.

The package `org.sigoa.refimpl.pipe.stat` contains such utility classes. Their base is the `PrinterPipe`, especially suitable for iterative algorithms and has three methods, all parameterized with the current iteration index and a `java.io.Writer` to write to, that can be overridden for solution candidate processing:

1. `onIterationBegin` is called at the beginning of each iteration. Here some stuff like the start time or headlines could be printed.
2. For each individual that passes through the pipe, `outputIndividual` is called. (The individuals will automatically be written to the next pipe stage, this is done elsewhere outside of `outputIndividual`.)
3. At the end of each iteration, an invocation of `onIterationEnd` is issued. This method provides the opportunity to print some statistics of the iteration.

An *iteration* is defined as the time between two `eof`s. For each of these iterations, `PrinterPipe` uses `provideWriter` to obtain a writer to which the data should be written. This method is also parameterized with the current iteration index which could for example be used to create the names of the destination files.

In order to make `PrinterPipe` a very versatile class, we define the interface `IWriterProvider` which specifies the aforementioned method `provideWriter`. The printer pipe now uses such an `IWriterProvider`-instance to create the `Writer`s it uses in the single iterations. We now can use writers that store their contents into a file or such that transmit the data over a TCP/IP[5]-connection. On the other hand, `PrinterPipe` is generic in terms of the type of the `Writer` to be used. This allows us to make use of the special utility classes provided by the Software Foundation Classes library, such as

- `sfc.io.TextWriter` is an extension of `java.io.Writer` with methods that allow us to write the primitive types like `int` or `double`, for Base64[6]-encoded output of raw binary data, for writing times and time differences in human-readable format, and so on.
  The provider for `TextWriter`s that write their content into files is called `FileTextWriterProvider` which creates new files named after the iteration index in a specified directory.
- Derived from `TextWriter` is the class `org.sfc.xml.sax.SAXWriter`. `SAXWriter` is the reversion of the Java SAX[7]. Instead of using SAX to

[5] `http://en.wikipedia.org/wiki/Tcp/ip` [accessed 2007-07-03]
[6] `http://en.wikipedia.org/wiki/Base64` [accessed 2007-07-03]
[7] `http://en.wikipedia.org/wiki/Simple_API_for_XML` [accessed 2007-07-03], `http://java.sun.com/javase/6/docs/api/` [accessed 2007-07-03]

read XML[8] it is here applied to write it. With `SAXWriter` you can create XML files plus use all the functionality of `TextWriter` inside the tags.
The provider for `SAXWriter`s that write their content into files is called `FileSAXWriterProvider` which creates new files named after the iteration index in a specified directory.

Two subclasses of `PrinterPipe` are provided: the `ObjectivePrinterPipe` stores the values of the objective functions of the individuals in form comma-separated values[9] whereas `IndividualPrinterPipe` stores the complete data provided by the individual records (objective values, fitness value, genotype, ...) in a text file.

[8] `http://en.wikipedia.org/wiki/Xml` [accessed 2007-07-03]
[9] `http://en.wikipedia.org/wiki/Comma-separated_values` [accessed 2007-07-03]

30

Clustering

In Chapter 36 on page 571 we have discussed the different clustering algorithms. A clustering algorithm divides a set A of elements a into disjoint subsets $b = \{a_1, a_2, ...\} \in B$. The unison of all the sets b then again equals A (see Definition 182). Optimization algorithms may use clustering in order to reduce large sets of solution candidates with a minimum loss of diversity. Clustering is applied in order to receive groups of individuals. From these, representative individuals are chosen and kept while the rest is disposed. Basing on the Sigoa pipelining architecture elaborated in Chapter 29, we provide the basic means to implement clustering algorithms for that purpose.

30.1 Specification

Figure 30.1 displays the class diagram of the clustering algorithm specification. In Sigoa, we use clustering only for the purpose of downsizing sets of individuals. Therefore, clustering can be implemented as a pipe stage. The implementation of `IClusteringAlgorithm` is thus a filter where some individuals go in and fewer individuals come out. It subclasses the `IPassThroughParameters`-interface from the package `org.sigoa.spec.go` which provides methods to get and set the count of individuals allowed to pass as well as the interface `IIndividualDistanceMeasureParameters` which allows to get/set an individual distance measure.

Distance measures are in Section 36.1 on page 574. In Sigoa, they are denoted by the interface `IDistanceMeasure` providing the method `distance`. `distance` returns the distance between two instances of `T` in the form of a `double` value. `T` is a generic parameter which allows distance measures to be specified for arbitrary entities.

One possible replacement for `T` is the individual record `IIndividual`, as in the `IIndividualDistanceMeasure` interface. Its implementors compute the distance between two solution candidates. This could be the euclidean dis-

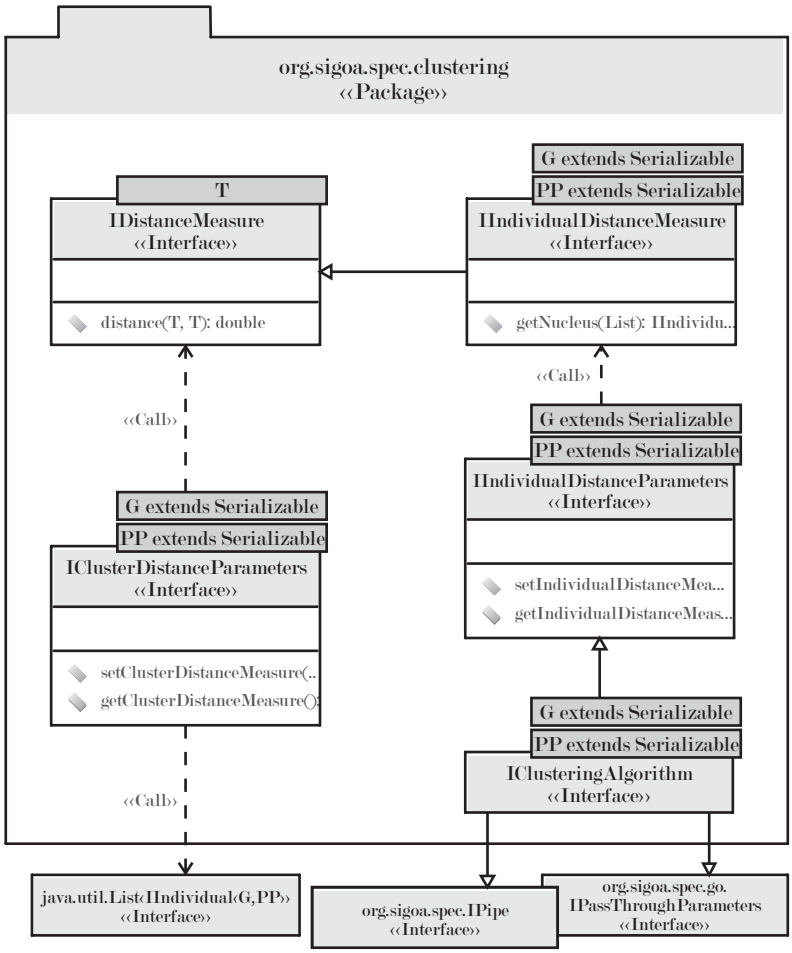


Fig. 30.1: The specification of clustering algorithm interfaces.

tance[1] of their objective values, the Hamming[2] distance of their genotypes, the difference of their fitness values, etc. A special function provided by this interface is the method `getNucleus`. Taking a list of individuals, it returns the most representative one according to the distance measure, according to the function *nucleus* defined in Section 36.2 on page 577.

In the terms of the clustering module of Sigoa, we consider clusters as lists of individual records. Thus, distance measures that work on clusters instead

[1] see Definition 188 on page 574
[2] see Definition 186 on page 574

of single individuals, as introduced in Section 36.1.3 on page 576, can be implemented by replacing `T` with `List<IIndividual<G,PP>>`. The

Clustering algorithms that make use of cluster distances like linkage clustering (see Section 36.3.4 on page 579) should additionally implement the interface `IClusterDistanceParameters` which defines functions for getting and setting a cluster distance measure.

30.2 Reference Implementation

In Figure 30.2 the base classes of the reference implementation of the Sigoa clustering algorithms are illustrated.

`ClusteringAlgorithm` implements the interface `IClusteringAlgorithm`. It is simple a class to derive real clustering algorithms from since it itself does not provide any functionality itself. It is derived from `BufferedPassThroughPipe` discussed in Section 31.2.1 on page 467 and therefore indirectly inherits from `BufferedPipe` (see Section 29.2.1 on page 429). Therefore, subclasses have to override the method `process` in which they have access to all solution candidates in order to implement a clustering algorithm. If such an algorithm also needs a distance measure for clusters instead of using on one for solution candidates only, it should be derived from `ClusteringAlgorithm2` rather than `ClusteringAlgorithm`.

A special utility class is `ObjectiveCluster` which basically is a list of individuals, a cluster. It additionally provides the method `getCenterIndividual` which returns an individual record with the objective values that mark the center of the multi-dimensional objective space spanned by its single elements (see Definition 184 on page 573). This individual record will not contain any other data, like a phenotype or a fitness value, only its objective values are set. `ObjectiveCluster` should be used by clustering algorithms that need to perform computations involving the cluster centers. It computes the center incrementally internally and can save a lot of processing time and thus reduce algorithmic complexity.

30.2.1 Clustering Algorithms

Some default clustering algorithms are provided in the package `org.sgioa.refimpl.clustering.algorithms` and illustrated in Figure 30.3. `NNearestNeighborClustering` realizes the n^{th} nearest neighbor clustering algorithm introduced in Section 36.3.3 on page 578. `NearestNeighborClustering` bases on the same algorithm but sets $n = 1$ and represents a very fast solution for this special case.

Using also cluster distance measures, `LinkageClustering` is an example subclass of `ClusteringAlgorithm2` and implements the linkage clustering algorithm defined in Section 36.3.4 on page 579. Its performance however is not very good, it normally runs very slow.

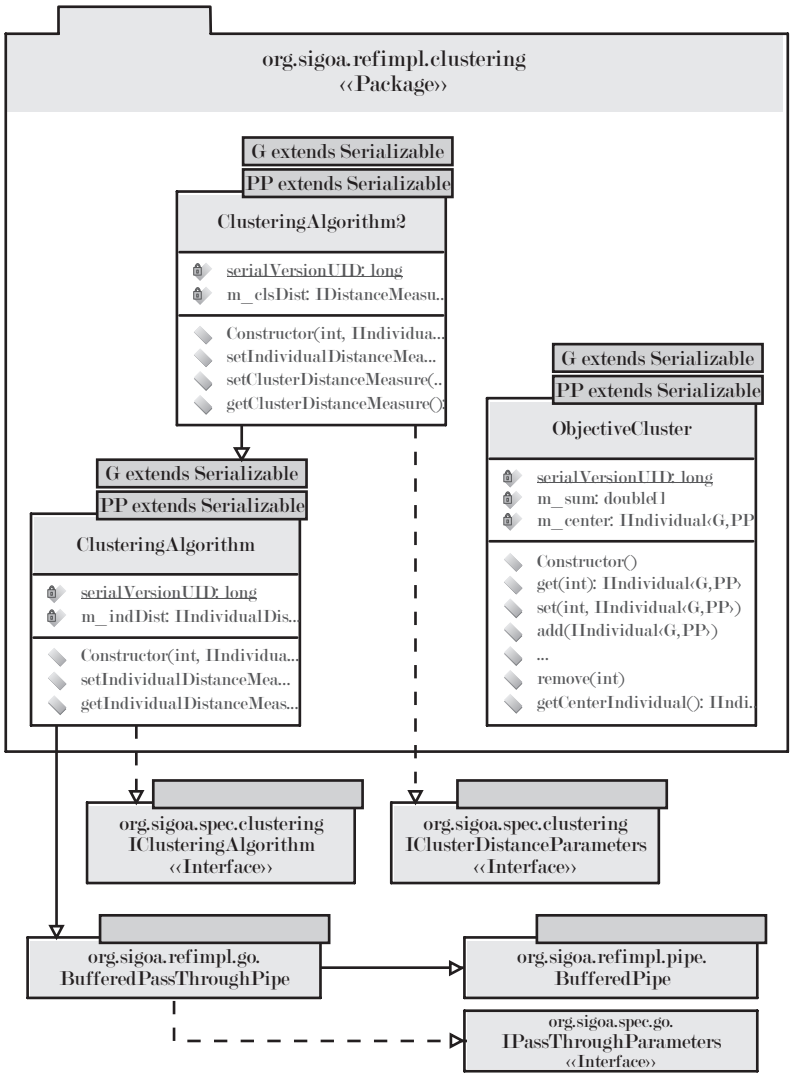


Fig. 30.2: Bases classes for clustering algorithms.

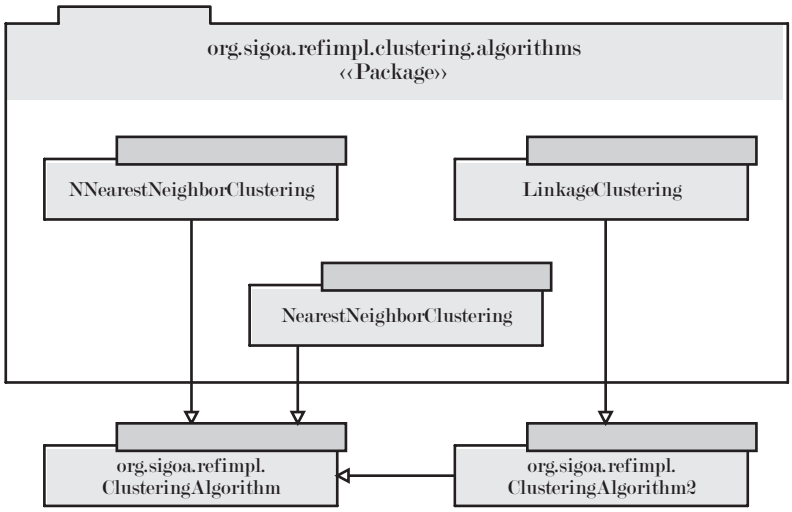


Fig. 30.3: Some clustering algorithms provided in Sigoa.

30.2.2 Distance Measures

In the package `org.sigoa.refimpl.clustering.distanceMeasures` we define some standard distance measures as shown in Figure 30.4. Section 36.1 on page 574 describes that distance measures can be defined for elements as well as for clusters of elements. We reflect this by specifying two sub-packages, `individual`, for element-based distance measures and `cluster` for cluster-based ones. Of course, the elements in our context are individual records.

In the package `individual`, the class `ObjectiveDistanceMeasure` is defined as base for all distance measures that use objective values. It defines the function `getNucleus` in a way that returns the individual closest to the cluster's objective centroid (see Definition 184 on page 573) according the distance measure.

Derived from this class is `ObjectivePNorm` which defines the p-norm on the objective space (see Definition 189 on page 575). Whereas `ObjectivePNorm` provides all possible p-norms, `ObjectiveNorms` holds some very common special cases like the euclidian, the manhattan distance and the infinity norm.

Distance measures for clusters, found in the `cluster`-package, will normally be derived from `ClusterDistanceMeasure`. They use a secondary distance measure which can be obtained/set by the methods `getIndividualDistanceMeasure` and `setIndividualDistanceMeasure`. `ClusterCenterDistance` for example uses this secondary measure to determine the distance between the centers of two clusters whereas `ClusterMaxDistance` uses it to return the longest distance between any element of the first and any element of the second cluster.

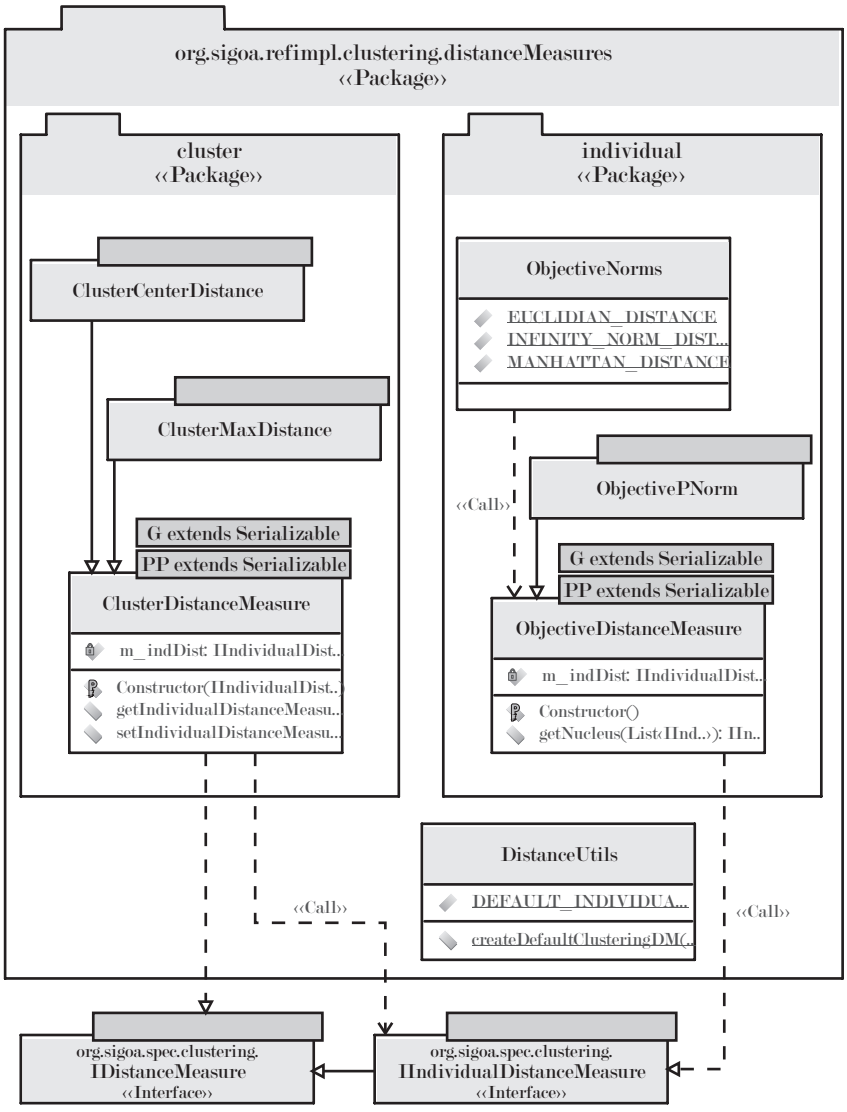


Fig. 30.4: Some distance measures provided in Sigoa.

The class `DistanceUtils` provides the default distance measures to be used for both, individual and cluster distance calculation.

31

Global Optimization

The intent of the Sigoa framework is to provide interfaces allowing to implement the optimization algorithms introduced in Part I on page 3. Furthermore, the interface architecture enables the programmer to replace all modules with realization different from the reference implementation. The global optimization package uses the pipe-technology provided by the package `pipe` discussed in Chapter 29. We regard optimization algorithms as composition of different filters that can be exchanged, removed or to which new ones can be attached, making the composition of specialized optimizers very simple.

In this chapter we introduce the optimization core of Sigoa.

31.1 Specification

31.1.1 Basic Interfaces

We strictly concentrate on multi-objective, prevalence-based optimization since it is the most general optimization technique (see Section 1.3 on page 12). Single-objective optimization, pareto-optimization and most other methods are special cases that can easily be realized with it. Figure 31.1 illustrates the basic interfaces of the Sigoa global optimization specification package.

The Individual Records

The crux of the whole Sigoa framework is the interface `IIndividual`. An individual record stores one solution candidate along with all information about it (see Table 31.1).

The solution candidate itself is described by the phenotype $x \in \tilde{X}$, which is parameter of the objective functions. Reproduction operations[1] may work on another representation of the individual, the genotype $g \in \mathbb{G}$. This is

[1] see Section 2.5 on page 99

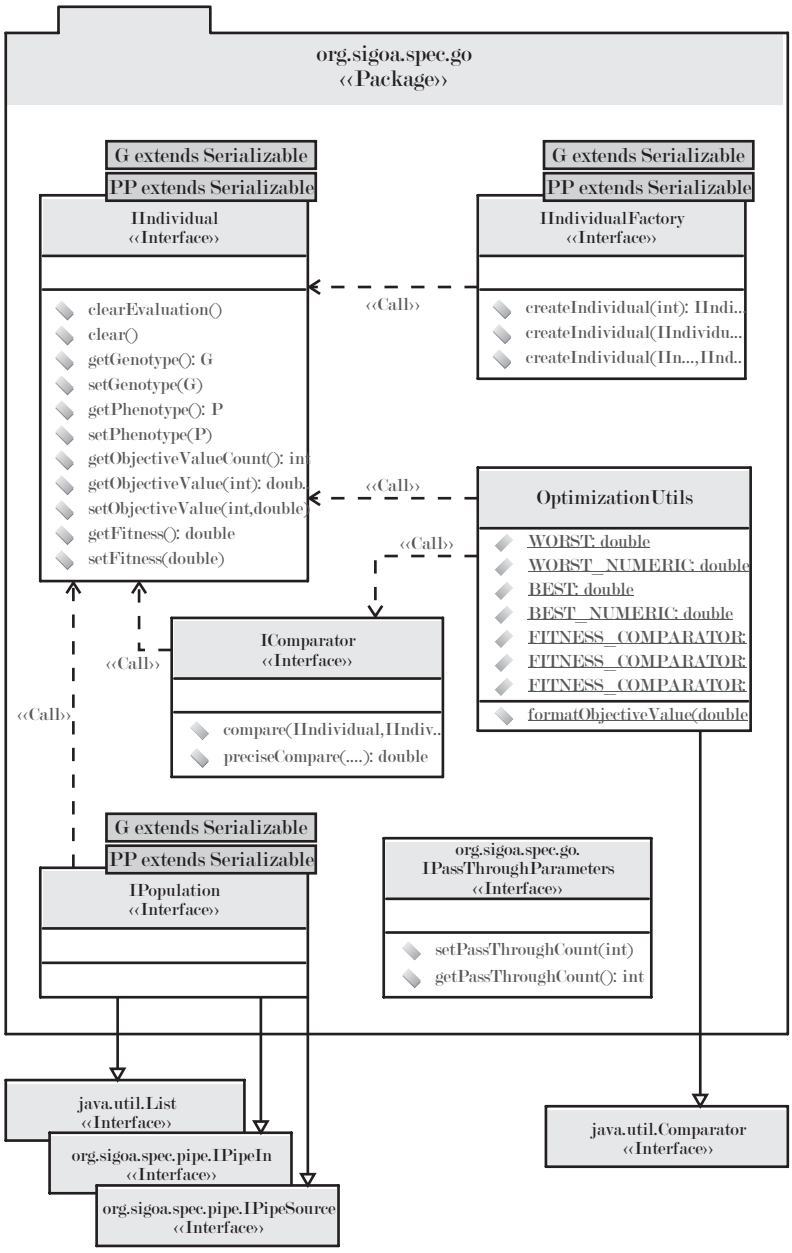


Fig. 31.1: The basic interfaces of the Sigoa global optimization package.

Table 31.1: The properties of `IIndividual`

property	discussed in section	`getter`	`setter`
genotype	Chapter 3 on page 117	`getGenotype`	`setGenotype`
phenotype	Chapter 3 on page 117	`getPhenotype`	`setPhenotype`
objective values	Section 34.6.3 on page 510	`getObjectiveValue`	`setObjectiveValue`
fitness	Section 2.3 on page 65	`getFitness`	`setFitness`

the case for genetic algorithms (introduced in Chapter 3 on page 117), for instance. Many other optimization methods do not need such a distinction - their reproduction methods use the same input data as the objective functions. In such instances, the genotype and the phenotype stored in the individual record will reference the same object.

A solution candidate can be rated with n objective values, where n must be specified at the individual record creation and can be obtained via `getObjectiveValueCount`. The objective values will normally be computed by the objective functions $f \in F, |F| = n$ and subsequently be stored into the individual record.

Instead of using the vector of the objective values directly, optimization algorithms often first compute a fitness value and compare individuals based on this fitness. If needed, an individual record also stores the fitness of the solution candidate.

`IIndividual` is just a container for the solution candidate information. Its data can be read and written by arbitrary modules. It is thus not specified where the values of its properties will come from. It is for example entirely possible not to use objective functions at all and fill in the objective values with numbers obtained according to a whole other method. Also, one could maybe refrain from using objective values at all by computing the fitness directly.

An individual record filled with one solution candidates data can be reused by clearing its fields using `clear` and filling it up with data of another individual. If an optimization process receives solution candidates in the form of individual records from another process, it may want to re-evaluate them. Therefore, it is sufficient to clear only the data that is concerned with the solution candidates evaluation by calling `clearEvaluation`.

An optimization algorithm will use an `IIndividualFactory`[2] in order to create the individual records it needs. The implementation of `IIndividual` can arbitrarily be exchanged every optimizers. The individual factory provides three overloaded[3] functions with the name `createIndividual`. The first one takes the

[2] factory design pattern: `http://en.wikipedia.org/wiki/Factory_object` [accessed 2007-07-03]

[3] overloading functions: `http://en.wikipedia.org/wiki/Function_overloading` [accessed 2007-07-03]

count of objective functions as argument and is thus good for creating new records[4]. One single individual is passed into the second method which creates a new individual record for the same count of objective values as needed in the *mutate* and *duplicate* functions[5]. The last `createIndividual`-method is intended for the use in the *crossover*[6]-operation needs two individual records as parameters and again, creates a new individual record for the same count of objective values.

A default implementation of `IIndividualFactory` would create new, empty individual records in all three methods. Another imaginable idea would be to have individual records that store inheritance information, i. e. which solution candidate is offspring of which other one. This can easily be done by implementing the three methods accordingly.

`IComparator` and Prevalence

In Section 1.3.5 on page 20 we discussed the prevalence optimization in which a comparator function c_F is defined up on the prevalence relation $\succ$. $c_F(x_1, x_2)$ becomes negative if solution candidate $x_1 \in \tilde{X}$ prevails over ("is better than") solution candidate $x_2 \in \tilde{X}$. It is zero if both individuals are equally good and positive if x_1 "is worse than" x_2. In Definition 18 on page 20, c_F inherently makes use of the objective functions $f \in F$. Instead of computing the objective values anew in each comparison, the `IComparator` works on those stored inside an individual record. In the mathematical definition, considerations about complexity were not needed. `IComparator` extends the Java interface `java.util.Comparator` defining an order over a class of objects (in our case individual records). This interface provides the method `compare` which compares two objects `o1` and `o2` (individual records) and returns an `int` value as result. This integer is negative if `o1` comes before `o2` in the defined order, positive if `o1` comes after `o2`, and zero if no precedence is defined between `o1` and `o2`. This conforms perfectly to the definition of c_F. Additional to the inherited method `compare`, `IComparator` specifies the method `preciseCompare` with exactly the same parameters. While codeilicompare returns an `int`, `preciseCompare` returns a `double` precision floating point number. Per definition, both methods must match in their outputs for the same individuals and have to hold the following assumptions:

$$\texttt{compare(o1,o2)} < 0 \Leftrightarrow \texttt{preciseCompare(o1,o2)} < 0 \tag{31.1}$$

$$\texttt{compare(o1,o2)} > 0 \Leftrightarrow \texttt{preciseCompare(o1,o2)} > 0 \tag{31.2}$$

$$\texttt{compare(o1,o2)} = 0 \Leftrightarrow \texttt{preciseCompare(o1,o2)} = 0 \tag{31.3}$$

$$\texttt{compare(o1,o2)} < 0 \Leftrightarrow \texttt{compare(o2,o1)} > 0 \tag{31.4}$$

$$\texttt{compare(o1,o2)} > 0 \Leftrightarrow \texttt{compare(o2,o1)} < 0 \tag{31.5}$$

[4] see Definition 42 on page 99
[5] see Definition 43 on page 99 and Definition 44 on page 100
[6] see Definition 45 on page 101

$$\texttt{compare(o1,o2)} = 0 \Leftrightarrow \texttt{compare(o2,o1)} = 0 \tag{31.6}$$

Furthermore, good a recommendation for their return values is that

$$\texttt{compare(o1,o2)} \approx \texttt{preciseCompare(o2,o1)} \tag{31.7}$$

One important final note: Instances of `IComparator` must only work on objective values.

The Optimization Utils

The class `OptimizationUtils` defines the general conditions for the optimization. Since we want to define reusable comparators (instances of `IComparator` representing c_F functions), we need to specify the *direction* of the optimization. In global optimization, minimization is most often applied. We therefore define here that the values of all objective functions as well as possible assigned fitness values are the better the smaller they are, i. e. are subject to minimization. The best possible objective value is therefore `OptimizationUtils.BEST == Double.NEGATIVE_INFINITY` and the worst one is `OptimizationUtils.WORST == Double.POSITIVE_INFINITY`. The best not-infinite objective value is `OptimizationUtils.BEST_NUMERIC == -Double.MAX_VALUE` and the worst numerical objective value is `OptimizationUtils.WORST_NUMERIC == Double.MAX_VALUE`.

The Populations

The interface `IPopulation` defines the functionality of sets of individuals, so-called populations. It extends the Java collection interface `java.util.List` and the Sigoa pipe interfaces `IPipeIn` and `IPipeSource`. Thus, it can act as a receiving end of a pipeline as well as a source that flushes all the individuals stored to a pipe.

The Pass-Through Parameters

The interface `IPassThroughParameters` in the clustering Chapter 30 on page 437. It is common to all algorithms that have a specific maximum count of solution candidates that may pass them. Later we will discuss that in selection algorithms (instances of `ISelectionAlgorithm`, see Section 31.1.6) for example, this count is enforced by omitting individuals that have not been selected while to few individuals simple are ignored. The `ICreatorPipe` explained in Section 31.1.2 does the opposite: it ignores if too many individuals pass but reacts on too few solution candidates by creating additional ones.

The pass-through count is set with the method `setPassThroughCount` and can be obtained via `getPassThroughCount`.

31.1.2 Reproduction

The Sigoa reproduction facilities are defined in package `org.sigoa.spec.go.reproduction` and depicted in Figure 31.2. There are two types of interfaces

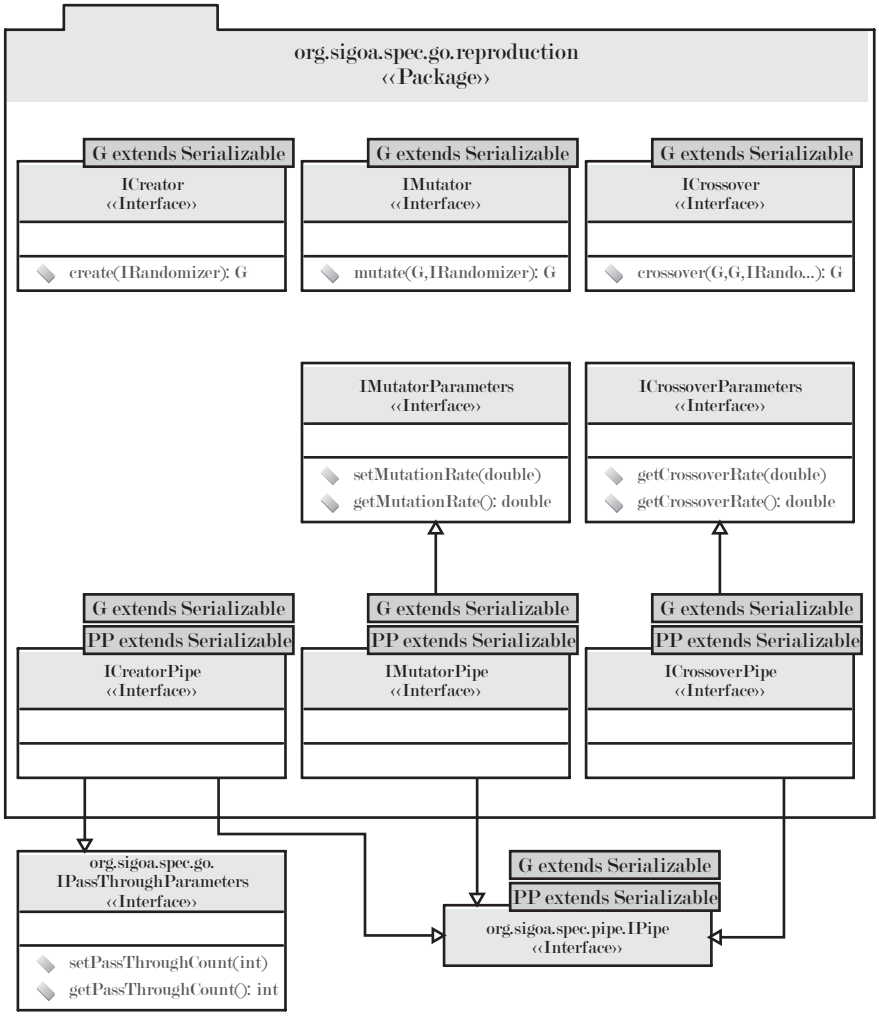


Fig. 31.2: The Sigoa reproduction facilities specifications.

in the `reproduction`-package: those which provide the operations introduced in Section 2.5 on page 99 and those which combine them with the pipe concept of Chapter 29 on page 427.

Interfaces that Provide Reproduction Operations

The reproduction operations are defined similar to those in Section 2.5 but take an additional parameter: By creating a new solutions candidate, things have most often to be carried out in a randomized manner. Therefore, an instance of `IRandomizer` (see Section 26.1.1 on page 399) has to be provided to the reproduction methods. The reproduction interfaces a generically parameterized with the type `G` which represents $\mathbb{G}$, the set of possible genotypes.

- `ICreator` defines the method `create` which corresponds to the operation *create*[7]. It creates a new individual $g \in \mathbb{G}$ with randomized content.
- `IMutator` mutates an existing instance of $\texttt{G} \equiv \mathbb{G}$ in its method `mutate` which obeys the Definition 44 of *mutate* on page 100.
- The method `crossover` of the interface `ICrossover` combines two instances of `G` in order to create a new offspring. It represents the *crossover*-operation[8].

Reproduction Pipes

Following the spirit of the Sigoa pipe architecture, we define special pipe stages for the reproduction operations. Each pipe stage has parameters set which define what it should do. Notice that the pipe stages are defined separately from the operations – the interfaces of the previous paragraph rather specify *how reproduction should be done* and can be used to create compatible implementations than strict rules. Pipe stages for reproduction may also base on different reproduction operation definitions (but should not, for the sake of compatibility).

- The `ICreatorPipe` is a filter that basically counts all solution candidates that pass through it without modifying them. When its `eof`-method is invoked, it compares their number to the predefined pass-through count. If at least that many individuals passed the pipe then everything is ok. Otherwise, the creator pipe should create the remaining count of individuals. These new solution candidates (for instance products of an instance of `ICreator`) are written to the next pipe stage before its `eof`-method is invoked. `ICreatorPipe` is a sub-interface of `IPassThroughParameters` which resides directly in the `go`-package and provides methods to get/set the pass-through count.
- `IMutatorPipe` is a sub-interface of `IMutatorParameters`. `IMutatorParameters` defines a mutation rate with getters and setters. The mutator pipe has to modify *approximately* this fraction of the individuals that are written to it and pass on the modified offspring. If the mutation rate is for example 0.34, then approximately 34% of the individuals should undergo a mutation.

[7] see Definition 42 on page 99
[8] Definition 45 on page 101

This can be done by drawing a uniformly between in $[0, 1)$ distributed random number for each individual that enters the pipe. If it is < 0.34, a mutated copy of the individual is written to the next pipe stage, the original is passed on otherwise.

- `ICrossoverPipe` works very similar: it uses a crossover rate defined by its parent interface `ICrossoverParameters` in order to determine the fraction of incoming solution candidates to be recombined. It may use a storage that can hold one individual. For each solution candidate received, again a random number can be drawn. If it is smaller than the crossover rate, the individual will be recombined. We first check now if the storage is empty. If yes, we store the solution candidate. Otherwise, we create two offspring of it and the stored individual, pass these two on and discard the stored one.

31.1.3 Objective Functions

The default definition of objective functions in Sigoa which can be found in the package `org.sigoa.spec.go.objectives` makes use of the simulations interface for individual evaluation. Its specification is outlined in Figure 31.3. In Sigoa, objective functions are instances of `IObjectiveFunction`. They are not just applied to mathematical optimization – the Artificial Ant example of Section 17.4 on page 284 has already shown that there can be very complex problems to be tackled. Such problems may involve simulating a solution candidate multiple times where simulation takes many steps. It is even possible that the objective function has to inspect the course of the simulations, performing introspective inspections after each nth step.

In Section 28.2.3 on page 422 we gave an example for job systems that utilize multiple processors in order to increase the performance of the optimization algorithms. Therefore, an instance of `IObjectiveFunction` may check multiple solution candidates in parallel. Since synchronizing the evaluation process would make no sense – it is the time consuming operation brings performance gains when parallelized – `IObjectiveFunctions` must not have any member variables. They have to be provided with containers for their state by the underlying system. Such containers can be created for each parallel thread. Now objective functions may be executed in parallel and have their states stored in the container that belongs to their hosting thread. No interference or race-conditions may occur.

The interface `IObjectiveFunction` is generic, it has the parameters

`PP` corresponding to the type of solution candidate evaluated – see $\tilde{X}$ in Definition 1 on page 3.

`ST` The per-evaluation state which extends `IObjectiveState`. The evaluation of an individual's objective values may need multiple rounds (called *evaluations* in the context of Sigoa). After each single evaluation, the objective function has to store the score of the individual into such a record. For each single evaluation, a single instance of `ST` is provided.

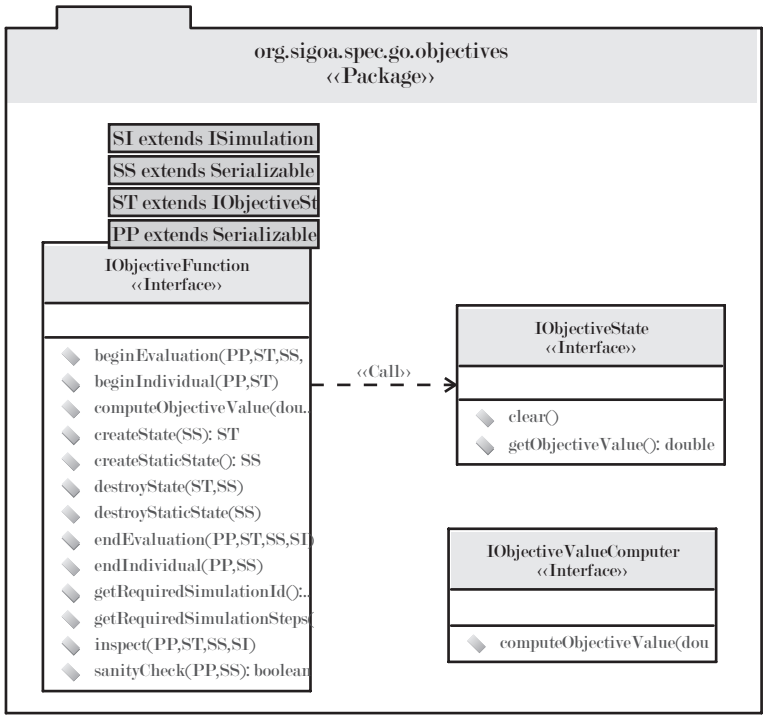


Fig. 31.3: The default specification for objective functions.

`SS` The *static* state per-individual state. One instances of this class is provided for all single evaluations. The objective function may use this record to aggregate values over all single evaluations.

`SI` The simulation type which extends `ISimulation` (specified in Section 27.1 on page 405). If the objective function makes use of a simulation in order to determine a solution candidate's fitness, it needs to specify the `SI`-parameter accordingly. Then, each single evaluation will involve one single simulation, being probably comprised of many simulation steps.

From this, you can see that the objective function itself obeys the factory design pattern[9] for its state and static state containers.

31.1.4 Computing an Objective Value

The computation of an objective value by an `IObjectiveFunction` proceeds as defined in Figure 31.4. In the following text, we define the order of the calls

[9] `http://en.wikipedia.org/wiki/Factory_object` [accessed 2007-07-03]

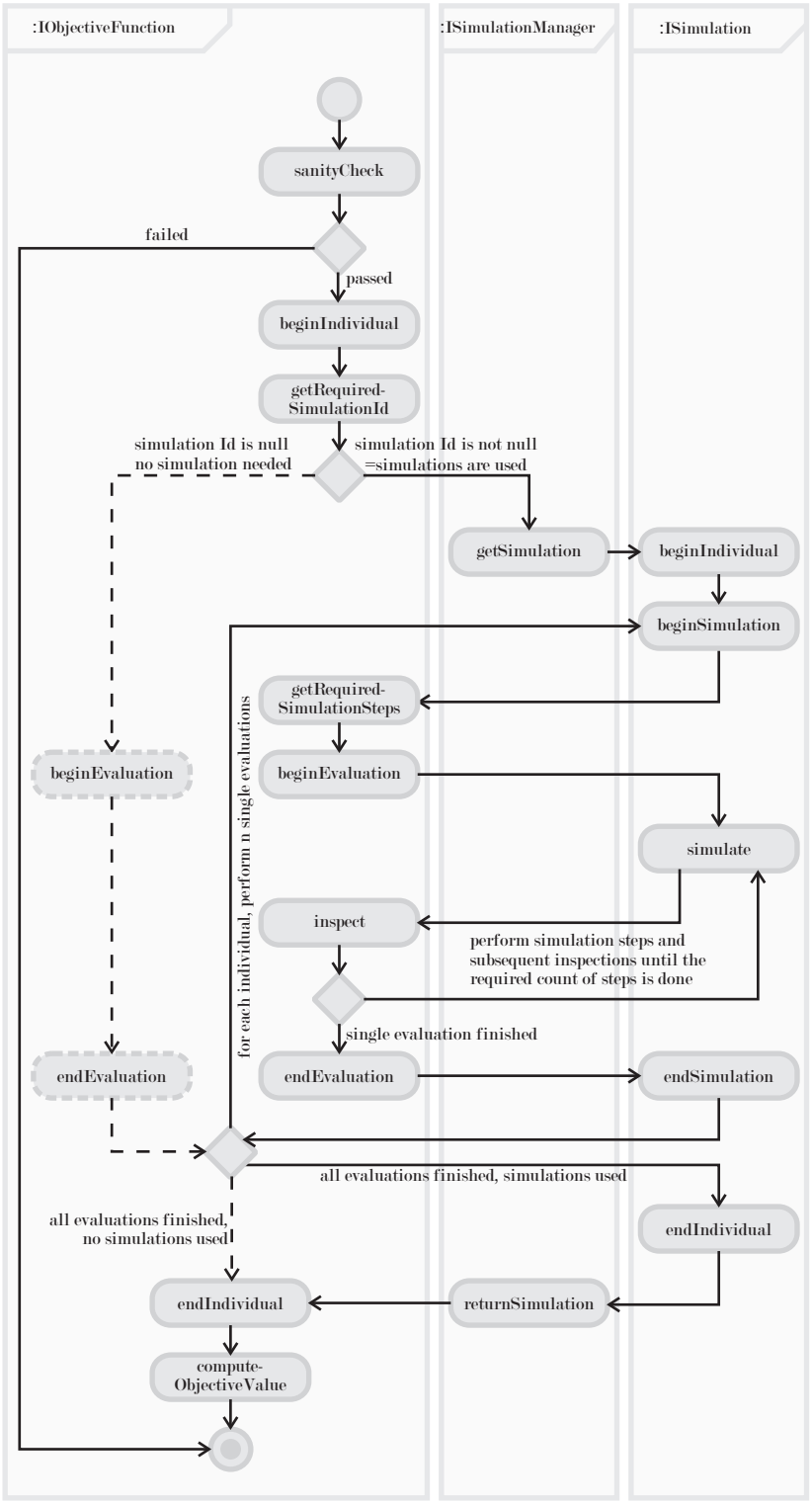


Fig. 31.4: The activity diagram of the evaluation of an individual.

to the methods of `IObjectiveFunction` in order to compute the fitness of a solution candidate.

Whenever the objective value of an individual has to be determined, we first perform a small sanity check with the method `sanityCheck`. No special simulations or evaluations are performed here – we just test if the solution candidate could possible be a valid solution.

If this is not the case, `sanityCheck` returns `false`and the objective value is set to `OptimizationUtils.WORST`. At this stage, you should keep in mind that we most probably perform a multi-objective optimization. A good approach in this case is to call the `sanityCheck`-methods of all objective functions involved before proceeding any further.

Only if all of them return `true`, the evaluation process begins with a call to `beginIndividual`. In this function, some initialization of the static state discussed before may be performed. After this is done, the id of the simulation which is needed for the evaluations is obtained by invoking `getRequiredSimulationId`.

If this id is `null`, no simulation is required. This means that the objective value can be computed directly from the individual data and there is only one single evaluation run needed.

Otherwise, a simulation of the requested Id is queried from the instance of `ISimulationManager` provided by the current host. This simulation is initialized via its method `beginIndividual` which receives the solution candidate as parameter.

Each single simulation run – since simulations may be randomized, multiple runs are possible required – begins with a call to `beginSimulation`. Here the simulation may set up round specific data.

It is now time to request the count of simulation steps that are needed by the objective function with an invocation to `getRequiredSimulationSteps`.

Then the evaluation is initialized by calling `beginEvaluation` which returns the number of simulation steps to be performed until the first inspection or `0` if no inspections are needed. In this case, the whole number of simulation steps specified by `getRequiredSimulationSteps` will be performed at with one single call to the method `simulate` of the `ISimulation`-instance.

Otherwise, `simulate` will only be called with as many steps as returned by `beginSimulation` and then the method `inspect` of the objective function is invoked. `inspect` can take a look into the simulation and maybe store some values in the objective state or the static state object. It returns the number of steps that have to be performed until the next inspection.

The evaluation cycle ends after all simulating is done and the number of steps returned by `getRequiredSimulationSteps` has elapsed. Then, the method `endIndividual` of the objective functions is called which writes the result, the objective value computed in during the simulation round, into the `IObjectiveState`-record passed in.

Afterwards, the simulation is notified that it is done via `endSimulation`. This process of simulation and evaluation may, as already said, be performed multiple times in order to get stable results.

`endIndividual` of the simulation is called so the simulator has a chance to perform some cleanup. Then, the simulation is returned to the `ISimulationManager` with `returnSimulation`. Now `endIndividual` is invoked on the objective function. It also may clean up here its static state.

During each single evaluation, the objective function filled in one objective value into an `IObjectiveState`-record. These values are now put into an array of `double`, are sorted, and are passed to `computeObjectiveValue` which combines them to a single result.

In order to provide some predefined routines for doing so, the interface `IObjectiveValueComputer` is defined. The objective function may use an instance of it to compute the final objective value.

To the scheme defined here, some extensions are possible. It is for example possible to call `getRequiredSimulationId` at any time before the solution candidate evaluations and its result can be stored and reused. In the context of multi-objective optimization, one may as well use one single simulation for all objective functions that require a compatible simulation Id. The reference implementation of Sigoa does exactly this to reduce processing time remarkable.

31.1.5 The Evaluator

Figure 31.5 outlines the package `org.sigoa.spec.go.evaluation` containing the interfaces that can be implemented to perform the aforementioned work. `IEvaluator` is implemented by classes that are capable to evaluate individuals. It therefore provides the method `evaluate` which takes an individual record as parameter. Its purpose is to fill in the objective values. Notice that there is no direct link to the interface `IObjectiveFunction`. Although it is the recommended way that an evaluator uses a set of instances of this interface, it also may apply any other technique to fill in fitness values. One strict requirement is however that the method `getObjectiveValueCount` returns the count of objective values that are specified per individual.

The interface `IEvaluatorPipe` is used to tag pipe stages that fill in objective values into individual records. The general contract of `IEvaluatorPipe` is that all individual records that pass it are evaluated in some way and their objective values are computed and stored. If it uses an instance of `IEvaluator` to do so, it would call its `evaluate` method for each instance of `IIndividual` it receives via `write`.

31.1.6 Embryogeny

We entitle the process of transforming genotypes and phenotypes into each other as embryogeny and provide the package `org.sigoa.spec.go.embryogeny`,

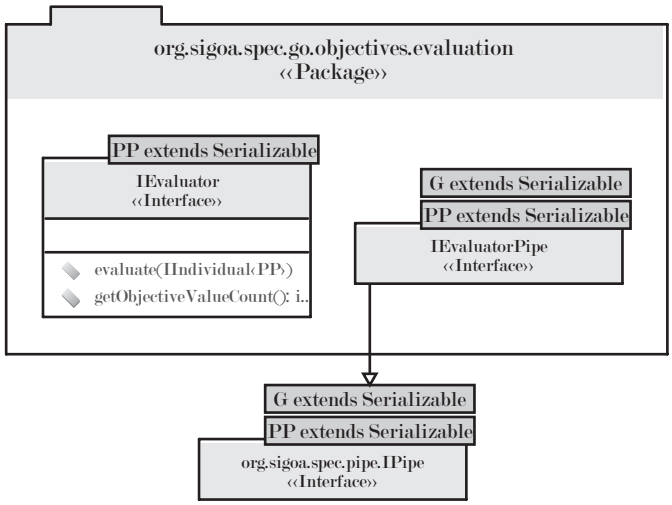


Fig. 31.5: The evaluation interfaces.

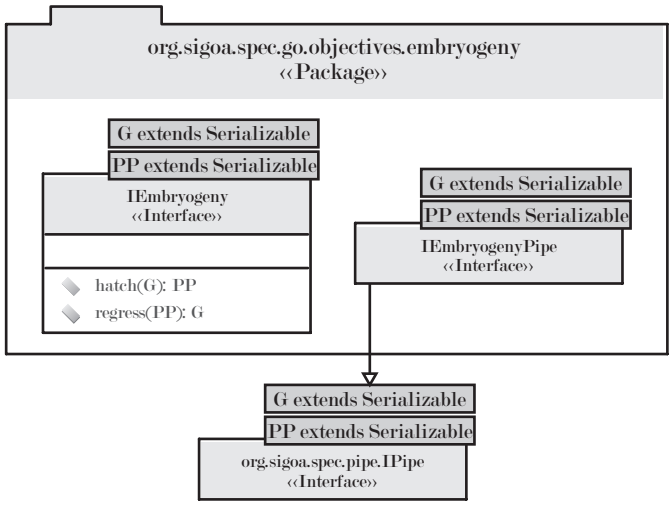


Fig. 31.6: The embryogeny specification of Sigoa.

illustrated in Figure 31.6, which contains according specifications. The interface `IEmbryogeny` has two methods: `hatch` performs the *hatch*-operation specified in Definition 57 on page 128 whereas `regress` represents the operation *regress* (see Definition 58). `hatch` must always be defined properly, but `regress` may return `null` if the hatching cannot be reversed.

The `IEmbryogenyPipe` takes a look into each individual record written to it. If no phenotype is stored (but a genotype is available) it fills in the phenotype that belongs to the genotype found. This is done best by using an instance of `IEmbryogeny`.

31.1.7 Fitness Assignment and Selection

As shown in Figure 31.7, two other predefined pipe stages are available. The fitness assignment processes elaborated on in Section 2.3 on

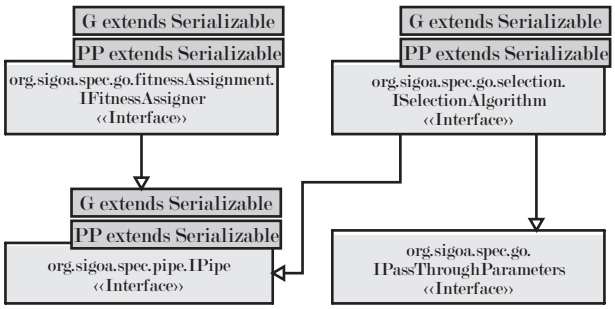


Fig. 31.7: Other predefined pipe stages

page 65 can be encapsulated in instances of `IFitnessAssigner`. Such a pipe stage fills fitness values into individual records. It represents the operation $assignFitness(X_{pop}, X_{arc})$ whereas X_{pop} is implicitly given by the individual records written to the pipe stage between two calls to `eof`. These can be accumulated and later processed as one chunk, if needed.

Remember that, as of Definition 39 on page 65, fitness values must always be in $\mathbb{R}^+$.

Incorporating X_{arc} is more complicated. If an archive is needed for the fitness assignment process, the optimization algorithm itself must maintain it (Where else should it come from?). This archive must then be made available to the fitness assignment process via some additional mechanism. Per default, only the solution candidate stream is used. Most often, the fitness values will be based on the objective values previously determined by an evaluator.

`ISelectionAlgorithm` encapsulates the operation $select(X_{sel}, n)$ introduced in Section 2.4 on page 78. The parameter X_{sel} again is implicitly given

by the individual records written to the pipe stage. n is represented by the pass-through count defined in the `IPassThroughAlgorithm`-interface `ISelectionAlgorithm` inherits from. Selection algorithms may either use fitness values previously computed by a fitness assignment process or directly involve objective values set by an evaluator.

31.1.8 The Optimizer

The optimization system is connected to the job system via two interfaces: `IOptimizer` and `IOptimizationInfo` (see Figure 31.8). The interface

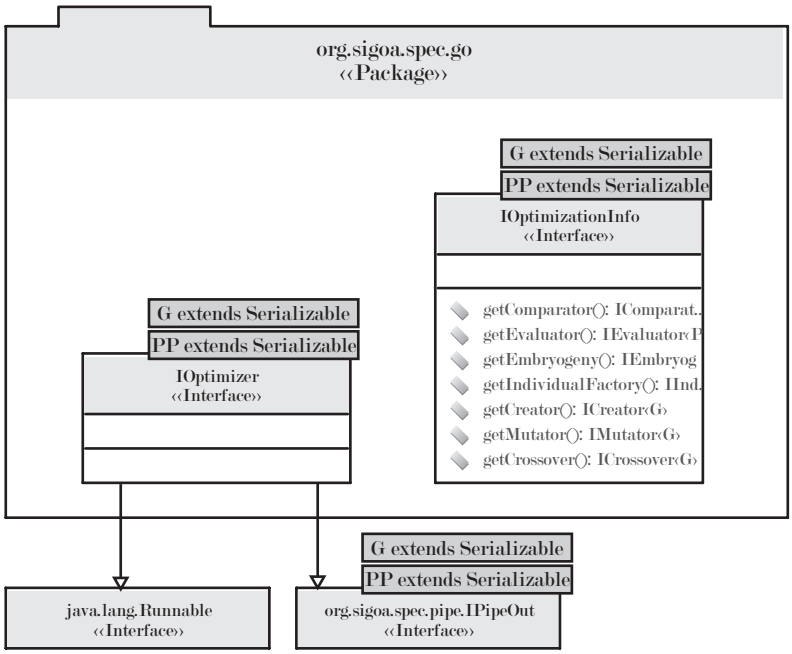


Fig. 31.8: Optimizer and Optimization Info Record

`IOptimizer` represents a global optimization algorithm. It inherits from `java.lang.Runnable` so it can be executed by host threads. Thus, its code resides in an implementation of the method `run`. `IOptimizer` also inherits from `IPipeOut` and thus represents the output end of a pipeline. Per definition it will write out the best solutions found through this interface when the optimization process is finished (or aborted).

An implementation of `IOptimizer` could (but does not necessarily need to) also implement the interface `IActivity`. If doing so, the optimization process could be gracefully aborted if needed. Furthermore, additionally implementing

IPipe would allow individual records to be written to the optimizer which then could integrate them into the optimization process.

31.1.9 The Optimization Info Record

The optimization info record, IOptimizationInfo, guides the optimization process by making functional components public to all its sub-activities, as enumerated in Table 31.2.

Table 31.2: The functional components provided by IOptimizationInfo

component	discussed in section	getter
comparator	Section 31.1.1 on page 448	getComparator
creator	Section 31.1.2 on page 451	getCreator
crossover	Section 31.1.2 on page 451	getCrossover
evaluator	Section 31.1.5 on page 456	getEvaluator
embryogeny	Section 31.1.6 on page 456	getEmbryogeny
individual factory	Section 31.1.1 on page 445	getIndividualFactory
mutator	Section 31.1.2 on page 451	getMutator

The optimization info record is handed over to the job system together with the optimizer at job creation. It is part of the IJobInfo-record discussed in Section 28.1.2 on page 415 which can be accessed through the method getJobInfo method of the host-threads.

31.1.10 Predefined Algorithm Interfaces

In the package org.sigoa.spec.go.algorithms you can find the predefined optimizers illustrated in Figure 31.9. IIterativeAlgorithm is an interface common to all algorithms that proceed iteratively and defines the method getIteration which returns the index of the current iteration. Algorithms that use an archive in order to perform some sort of elitism, the interface IElitistAlgorithm defines the methods getMaxArchiveSize and setMaxArchiveSize that get/set the size of this archive.

Although these two interfaces could be implemented by any sort of algorithm with the specified semantics (not only optimizers), the IEA interface is a good base to define evolutionary algorithms[10] and inherits directly from IOptimizer. It is furthermore derived from IIterativeAlgorithm (since evolutionary algorithms are always iterative) and from IMutationParameters and ICrossoverParameters (since evolutionary algorithms usually support the *mutation* and *crossover* operations and thus need to maintain mutation and crossover rates).

[10] see Chapter 2 on page 47

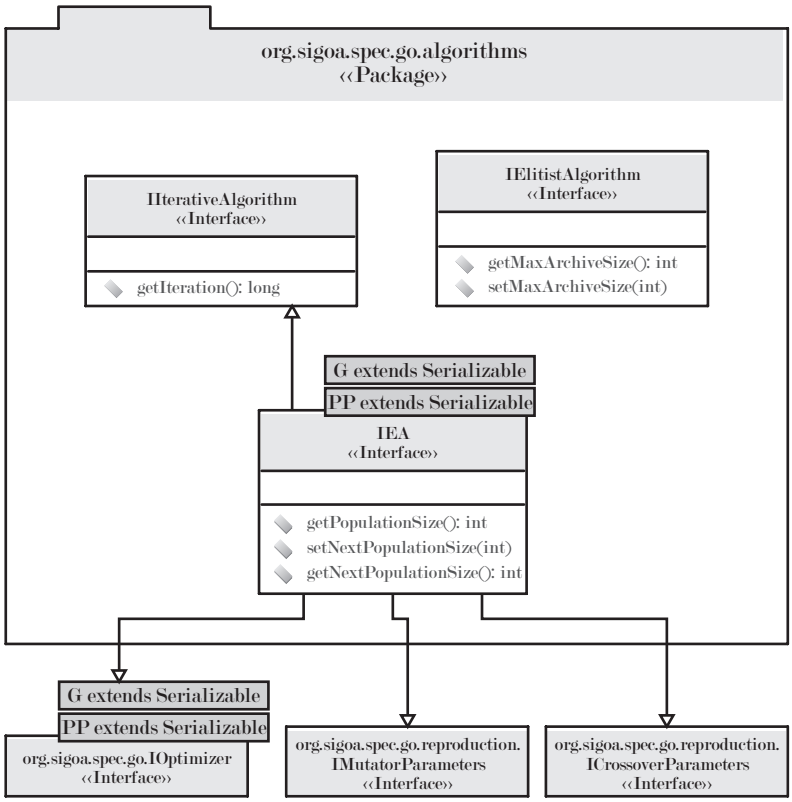


Fig. 31.9: Predefined optimization algorithms.

31.2 Reference Implementation

The reference implementation of the Sigoa global optimization techniques can be found in the package `org.sigoa.refimpl.go`.

31.2.1 Basic Classes

The Implementation Base

As basis for most of the implementations of the global optimization interfaces we use the class `ImplementationBase` illustrated in Figure 31.10. The pipe implementation root class, `PipeOut`, for example is derived from it. `ImplementationBase` provides `protected` methods which grant access to the information provided by the host threads. This way, working with `JobSystemUtils.getCurrentHost` has been replaced by a more elegant, indirect way to access the host data. Testing of algorithms or pipe stages also

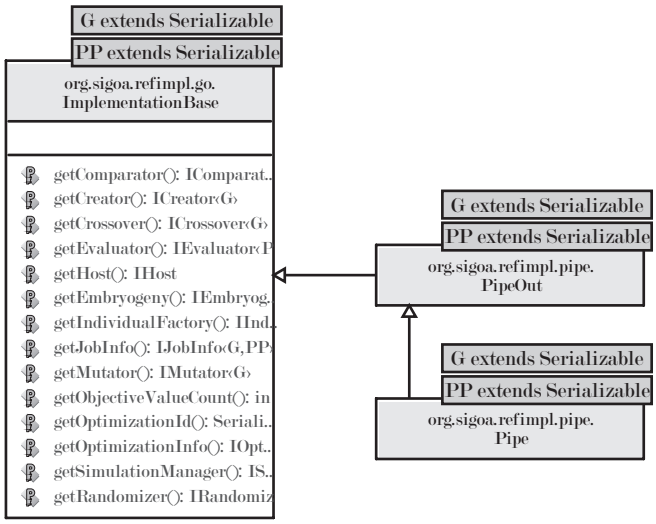


Fig. 31.10: The class `ImplementationBase`.

becomes easier since the methods defined in `ImplementationBase` can be overridden in a way that does not longer require the tested code to run inside a host environment.

The Individual Records

The interfaces `IIndividual` and `IIndividualFactory` are implemented by the classes `Individual` and `IndividualFactory` as you can see in Figure 31.11. An `Individual`-record is created by passing the count of objective functions into the constructor. It overrides the `toString` in order to provide a human readable representation of the individual record. More important, it overrides `equals`, so individual records can be compared for equality. `equals` now compares all objective and fitness values as well as the genotypes and phenotypes for sameness.

The `IndividualFactory` creates instances of `Individual`, using the parameters of its `createIndividual`-methods to obtain the count of objective values needed. An instance of `IndividualFactory` that can globally be shared can be found in the `static final` variable `DEFAULT_INDIVIDUAL_FACTORY`.

The Predefined Comparators

We predefine some prevalence comparator functions in the package `org.sigoa.refimpl.go.comparators` illustrated in Figure 31.12. The utility class `ComparatorUtils` contains useful helper functions. `preciseToNormal` transforms

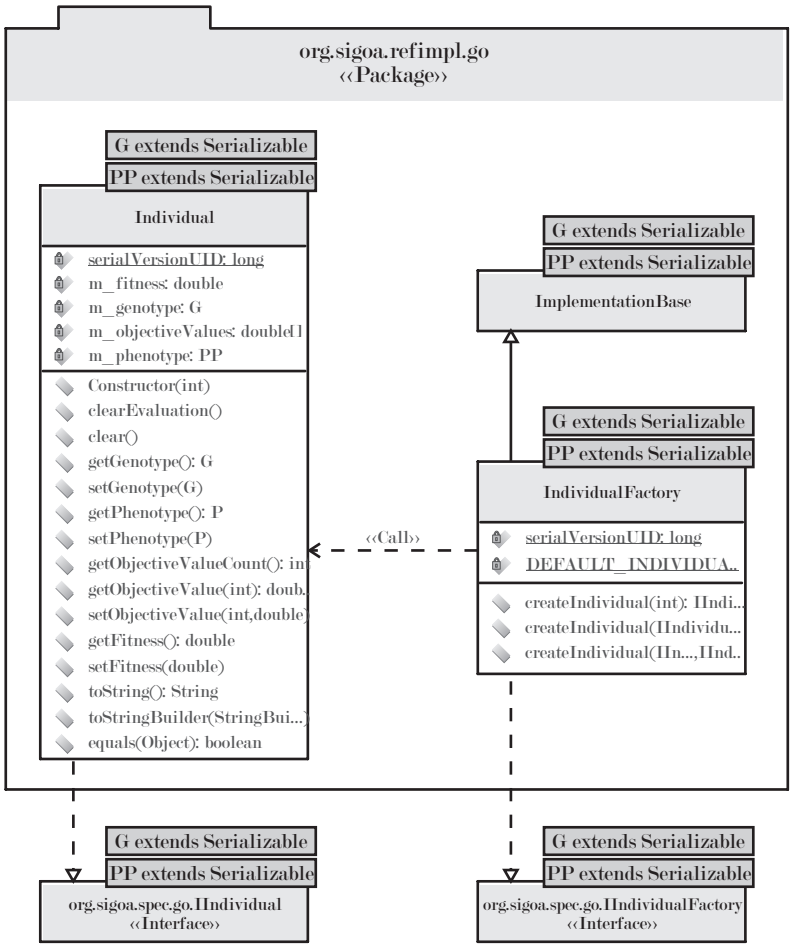


Fig. 31.11: The class `Individual` and `IndividualFactory`.

the results of the function `preciseCompare` (a `double`) to an `int` as returned in `compare`. Some comparators may perform their calculations in `preciseCompare` rather than in `compare` and thus need a way to converse the result. Using `preciseToNormal` for that purpose, the contract of Section 31.1.1 on page 448 is ensured.

`ComparatorUtils` also defines two fallback functions (`compareFallback` and `preciseCompareFallback`) for the case that a comparison fails. A comparison may fail if the resulting value would be `Double.NaN`. Instead of returning that, one should return the values computed by the fallbacks. The fallbacks use the `MajorityComparator` also discussed in the section.

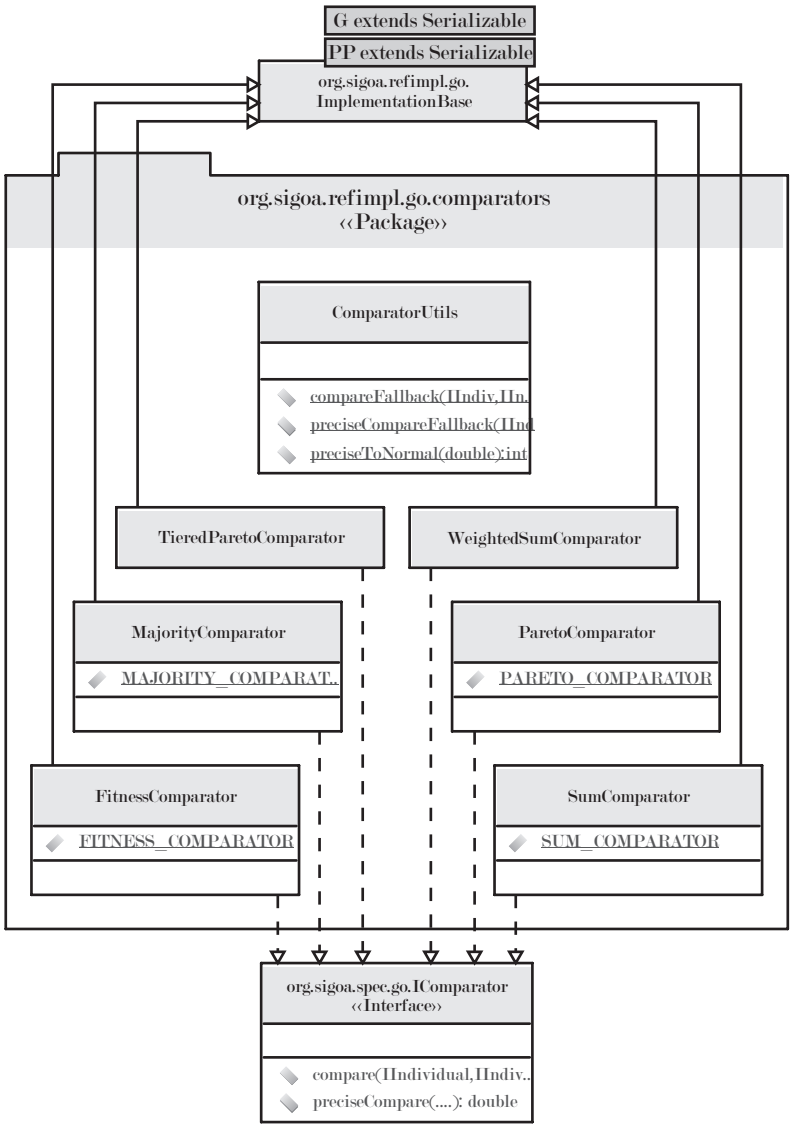


Fig. 31.12: Some predefined comparator functions.

All comparator functions specified here inherit from `ImplementationBase` and realize the interface `IComparator`. Some of the comparators have a behavior that has no parameters and is independent of the count of objective values. For these, we can define globally shared default instances in the form of `public static final`-constants. Some fundamental comparators are

1. `ParetoComparator` which compares the objective values of two solution candidates according to the pareto-principle introduced in Section 1.3.2 on page 14. It thus corresponds to the preto-realization of prevalence defined in Equation 1.16 on page 21. The default instance of this comparator is `PARETO_COMPARATOR`.
2. The `MajorityComparator` takes another approach: the individual that *wins* in most of the objectives also wins the comparison. The result obeys the Equation 31.8. This comparison method does not necessarily yield all possible optimal solutions since it grants a higher weight to individuals that dominate other (maybe also optimal) individuals in more points than vice versa. The default instance of this comparator is `MAJORITY_COMPARATOR`.

$$c_{F,major}(x_1, x_2) = \sum_{i=1}^{n} \begin{cases} -1 \; if \; f_i(x_1) < f_i(x_2) \\ 1 \; if \; f_i(x_1) > f_i(x_2) \\ 0 \; else \end{cases} \tag{31.8}$$

3. `SumComparator` simple adds up all objective values of the individuals and thus returns the result of Equation 31.9. It realizes the most primitive version of weighted sum prevalence with all weights set to 1. The default instance of this comparator is `SUM_COMPARATOR`.

$$c_{F,weightedS}(x_1, x_2) = \sum_{i=1}^{n} (f_i(x_1) - f_i(x_2)) \tag{31.9}$$

   If the sum comparison fails, i.e. the sum is `Double.NaN`, the fall back defined in `ComparatorUtils` is used.
4. Weighted sum prevalence (see Section 1.3.1 on page 13 and Equation 1.15 on page 21) is implemented in the `WeightedSumComparator`. This constructor comparator takes an `d` array of `double` as parameter. This array is copied, and all objective values f_i are weighted with the values `d[i]`. If the weighted sum comparison fails, i. e. the sum is `Double.NaN`, the fall back defined in `ComparatorUtils` is used.
5. The `TieredParetoComparator` is an extension to the standard pareto comparison. Here we define levels of objective functions. The comparison works as follows: first, only the set of objective functions in the lowest level (0) is used. If in this set one individual dominates the other one, it also wins the comparison. If no pareto-domination could be found, then the next tier (1) is taken into consideration and so on. If neither of the two solution candidates dominates the other one in any of the levels, it is a tie. The levels can be configured freely by passing an array of `int` to the constructor which holds their (strictly monotonic increasing) borders.

6. We also define an implementation of `IComparator` which compares individuals according to their fitness as specified in Equation 31.10. Normally, comparators must only use the objective values to compare individuals. The fitness of a solution candidate is determined by a fitness assignment process and will most probably be initially not available. There is however one instance where comparison on the basis of fitness values may be needed: in selection algorithms. A selection algorithm may be dependent on the previous fitness assignment and thus use a `FitnessComparator`. Notice that defining this class, although being a minor breach of the prevalence-paradigma, allows selection algorithms to be defined using comparators instead of strictly adhering to fitness values. A selection scheme that uses instances of `IComparator` does not necessarily need a fitness assignment process but may as well be based on, for example, direct pareto relations. If the fitness comparison fails, i. e. the sum is `Double.NaN`, the fall back defined in `ComparatorUtils` is used. The default instance of this comparator is `FITNESS_COMPARATOR`.

$$c_{F,fitness}(x_1, x_2) = \mathfrak{f}(x_1) - \mathfrak{f}(x_2) \tag{31.10}$$

The Populations

The default implementation of `IPopulation` is the class `Population`. It uses the default Sfc list implementation, `DefaultList` to derive the `java.util.List`-functionality, as outlined in Figure 31.13.

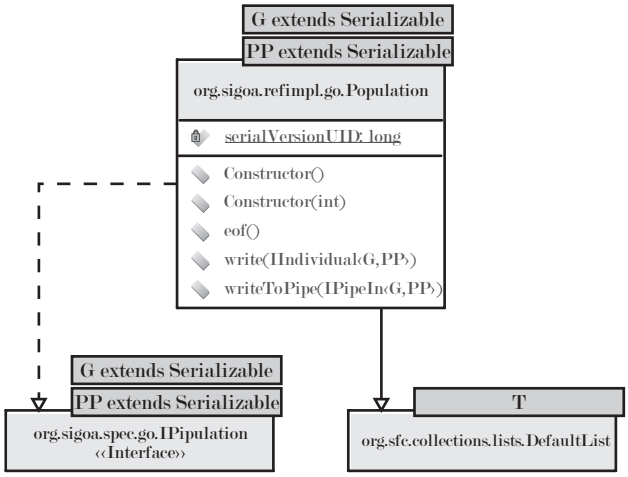


Fig. 31.13: The class `Population`.

`PassThroughPipe` **and** `BufferedPassThroughPipe`

In Figure Figure 31.14, `PassThroughPipe` and `BufferedPassThroughPipe`, the base classes for pass-through algorithms, are depicted. Both classes imple-

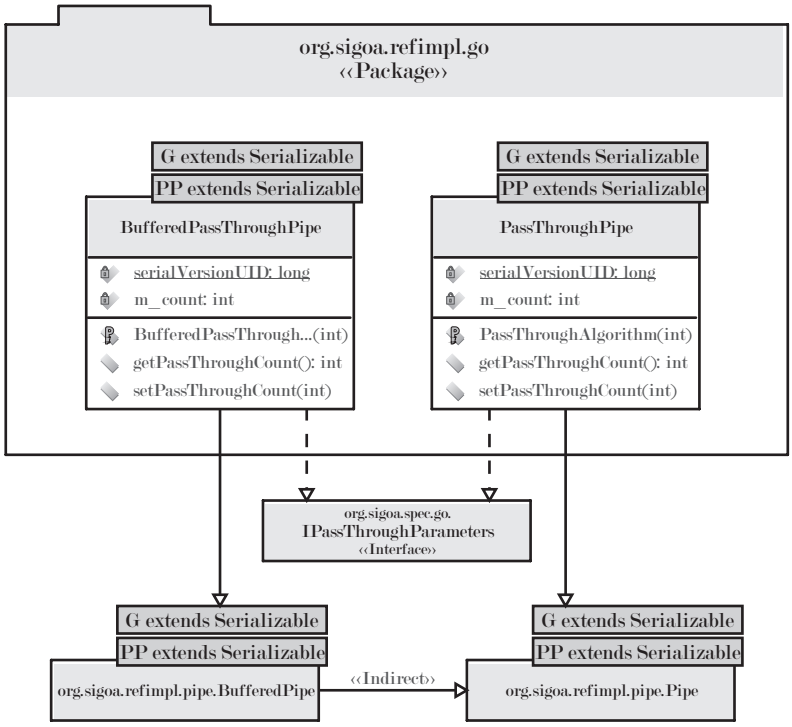


Fig. 31.14: The base classes for pass-through algorithms.

ment the interface `IPassThroughAlgorithm`. `BufferedPassThroughPipe` inherits from `BufferedPipe` and decides which individuals should pass by the means of investigating the whole set of solution candidates at once. Such behavior is ideal for selection algorithms, for example. `PassThroughPipe` on the other hand is a direct descendant of `Pipe` and performs such decisions *on the run*.

31.2.2 Reproduction

The reference implementation of Sigoa reproduction facilities can be found in the package `org.sigoa.refimpl.go.reproduction`.

Classes that Implement the Reproduction Operations

The (partly `abstract`) base classes that implement the reproduction operations discussed in Section 31.1.2 on page 451 are sketched in Figure 31.15. `Creator`, `Mutator` and `Crossover` are such base classes. They implement the

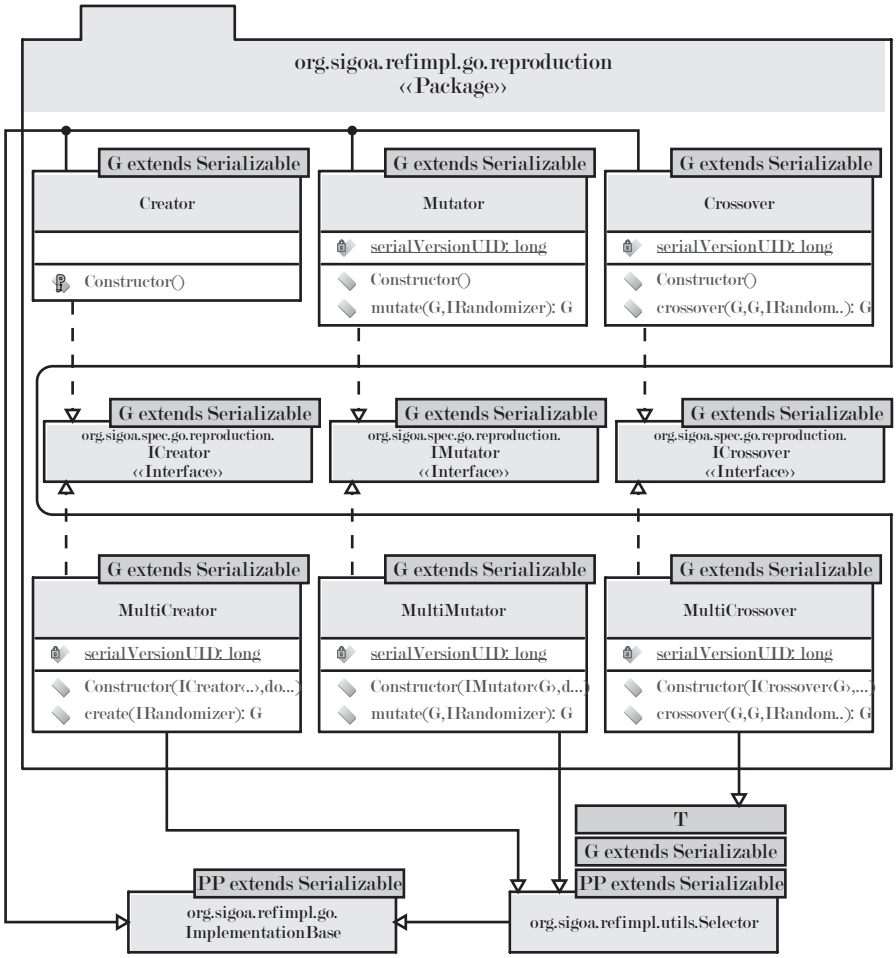


Fig. 31.15: The base classes that implement the reproduction interfaces.

interfaces `ICreator`, `IMutator`, and `ICrossover`, respectively. Of course, since they are generic with the parameter `G` denoting the genotype, their operations do not yet return anything. `Creator` even is `abstract`, and `Mutator` as well as `Crossover` return the input solution candidates as output in their dedicated

operation methods. However, deriving the operations for your genotypes from such classes means using a common foundation providing utility functions (by inheriting from `ImplementationBase`) and ensuring compatibility to future versions of the Sigoa.

Very useful classes are also the operation multiplexers `MultiCreator`, `MultiMutator`, and `MultiCrossover`. The inherite from the class `Selector` discussed in Section 33.1.2 on page 497. Their constructors takes an array of either `ICreator`, `IMutator`, or `ICrossover` plus an array of `double`. While the first array contains the operations to multiplex to, the second one contains their weight. Let us consider the fixed-length string chromosomes introduced in Section 3.4.1 on page 124 for genotypes. We can now implement a one point, a two point, and a n point crossover operation. By passing these operators in an array to the constructor of `MultiCrossover` along with `{1,2,3}`, the resulting multiplexing crossover operation would defer in one of six cases to the single point, in two of three cases to the double point and in half of the invocations to the n point crossover operators. The three multiplexers make it thus very easy to combine different reproduction operators.

Reproduction Pipes

The reference implementation classes of the reproduction pipe specifications from Section 31.1.2 on page 451 are outlined in Figure 31.16. `CreatorPipe` is a specialization of the `PassThroughPipe` and implements `ICreatorPipe`. It counts how many individuals are written to it while passing them straight on to the next pipe stage. If `eof` is invoked, it checks whether at least the pass-through count of solution candidates passed it. If not, it uses the default creator specified in the optimization information record (see Section 31.1.8 on page 459, obtained via the method `getCreator` inherited from `ImplementationBase`) to create the missing number.

`MutatorPipe` derives from `Pipe` and implements the interface `IMutatorPipe`. It stores the mutation rate and whenever a solution candidate passes it, it generates a random number uniformly distributed in $[0, 1)$. If this number is lower than the mutation rate, the individual will be mutated – otherwise it is just passed on. Mutation is done by querying the active mutator, the individual factory, as well as the randomizer from the host thread via the inherited methods from `ImplementationBase`. These are then used to create a randomized offspring of the solution candidate and to package it into a new individual record.

`CrossoverPipe` also inherits from `Pipe` and realizes the interface `ICrossoverPipe`. It works very much like the `MutatorPipe` with one exception: it needs two individuals to perform its operation. Thus, instead of creating a new offspring each time an individual is passed in and a random number below the crossover rate is generated, it creates two new *children* of two *parent* solution candidates. It therefore has a buffer with space for a single individual record into which the first of the two parents is stored. Whenever the

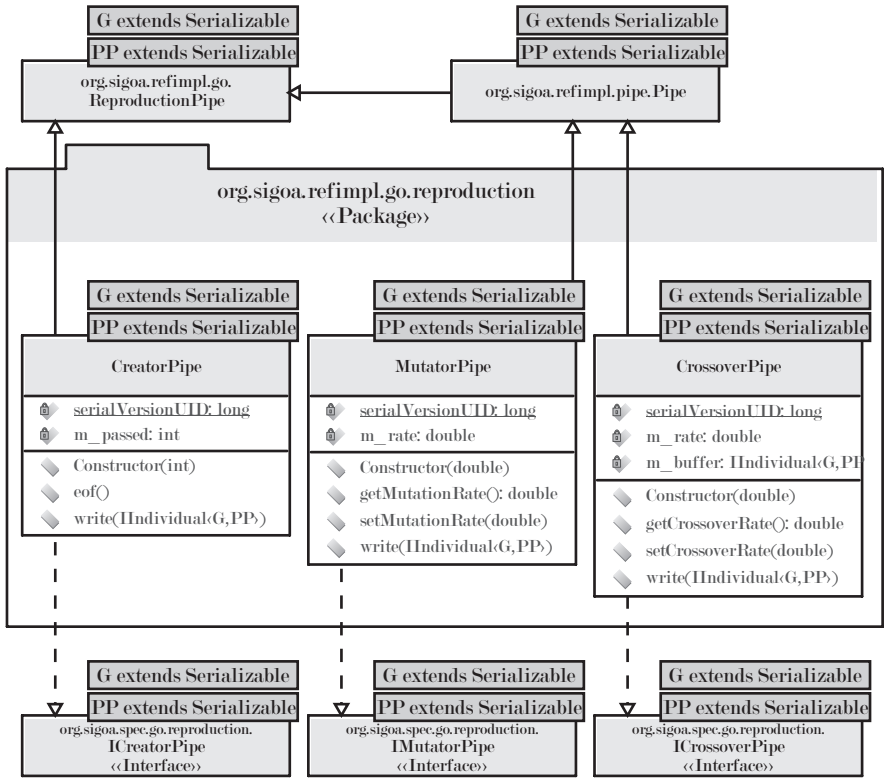


Fig. 31.16: The reproduction pipes.

second solution candidate is selected for crossover, it is combined twice with the buffered one, creating two offspring. Afterwards, the buffer is cleared. If `eof` is called while an individual is still in the buffer, it is passed on before propagating the call.

31.2.3 Objective Functions

In this section we discuss the reference implementation of the Sigoa concept of objective functions discussed in Section 31.2.3. Figure 31.17 illustrates the hierarchy of the package `org.sigoa.refimpl.go.objectives` which contains this reference implementation. `ObjectiveState` realizes the single-evaluation state of an objective function specified in the interface `IObjectiveState`. Basically, it is just a container for one `double` value denoting the objective value. If needed, classes with more specific fields should be derived from it.

The default implementation of the interface `IObjectiveFunction` is the class `ObjectiveFunction`. It implements all the necessary methods with-

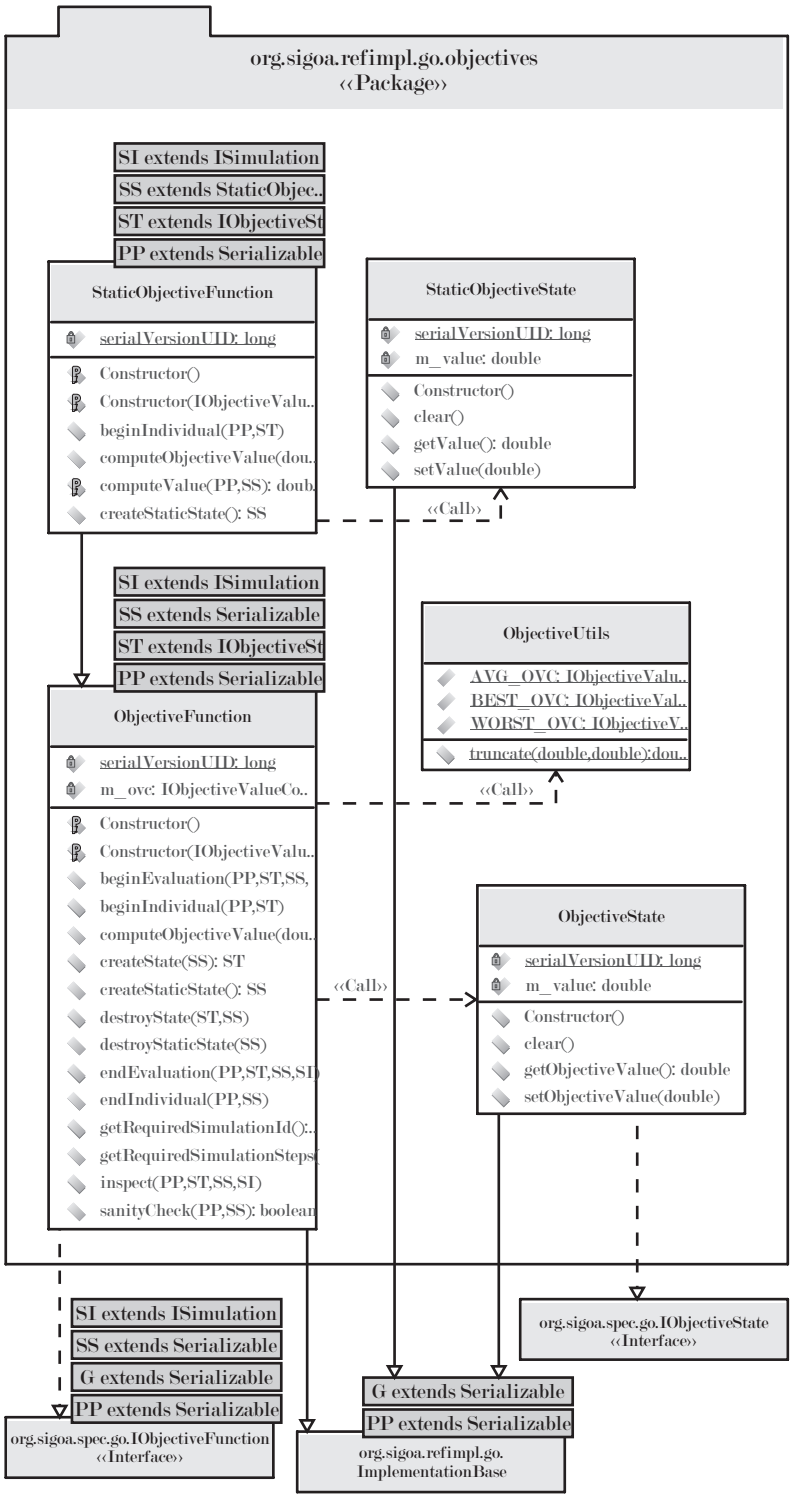


Fig. 31.17: The reference implementation of the objective functions.

out performing any specific functionality. Its method `createState` returns a new instance of the aforementioned class `ObjectiveState`. The method `getRequiredSimulationId` as well as `createStaticState` both return `null` and must be overridden if a simulation or a static state is needed. `sanityCheck` always returns `true`and `getRequiredSimulationSteps` returns `0`. The constructor of `ObjectiveFunction` optionally takes an instance of `IObjectiveValueComputer` as argument which is used `computeObjectiveValue`. This method is always called after all (single) evaluations of a solution candidate have been performed in order to determine the final objective value. This could be the average, worst or best of all single runs. The behavior of doing so is outsourced to the instance of `IObjectiveValueComputer`. If no such instance is specified, the `AVG_OVC` constant specified in `ObjectiveUtils` is used.

`ObjectiveUtils` provides useful utilities for the objective functions, like some default objective value computers:

1. `AVG_OVC` returns the arithmetic mean (see Definition 146 on page 521) of all objective values that are the results of single evaluations.
2. `WORST_OVC` returns the worst of all the objective values – the highest of them.
3. `BEST_OVC` returns the best of all the objective values – the smallest of them.

Furthermore, it provides the `static` function `truncate` which realizes a simple rounding mechanism. If evaluations/simulations are randomized, their results may contain noise. `truncate` helps to reduce the bandwidth of a floating point variable. Obeying the Equation 31.13, it takes two `double`s as arguments, the `value` (a in the equation) and the `precision` (p in the equation) it should be limited to. With this equation, we allow only a given count of powers of e for the contents of return value. It is very much like expressing a natural number with, let's say, four bits. If the number is $70'00$, for example, the closest we can get is to say $70'000 \approx 73'728 = 1*2^{16}+0*2^{15}+0*2^{14}+1*2^{13}$. This means that at least all number ins $[70'000, 73'728]$ are all mapped to a single one, removing possible noise of a magnitude of ca. $3'000$. The function `truncate` does exactly the same, but uses powers of e instead of powers of 2 (since we have only a built-in ln function in Java).

$$\forall a \geq 0, a \in \mathbb{R} \Rightarrow round(a) = \begin{cases} \lfloor a \rfloor & a - \lfloor if\ a \rfloor < 0.5 \\ \lfloor a \rfloor + 1 & else \end{cases} \tag{31.11}$$

$$\forall a < 0, a \in \mathbb{R} \Rightarrow round(a) = -round(-a) \tag{31.12}$$

$$truncate(a, p) = \begin{cases} -truncate(-a, p) & if\ a < 0 \\ 0 & if\ a = 0 \\ \frac{round\left(a * e^{p - round(\ln a)}\right)}{e^{p - round(\ln a)}} & else \end{cases} \tag{31.13}$$

A special derivate of `ObjectiveFunction` is `StaticObjectiveFunction`. This special function should be used to describe objective values that do not depend on simulations and are not randomized in any way. If deriving a pro-

gram for the artificial ant as described in Section 17.4 on page 284, the code size would be such an objective value. While this is a very simple example, a more complicated one would be the percentage of code that is actually reachable. This value can be computed by analyzing the program. It makes no sense to do so for every single evaluation – performing this calculation only once for each individual is enough. Therefore we introduce the class `StaticObjectiveState` which, very similar to `ObjectiveState`, holds exactly one `double` and is filled with the result of the method `computeValue` of `StaticObjectiveFunction`. This function is called exactly once per individual. The overridden version of `computeObjectiveValue` then returns this value and the overridden version of `createStaticState` returns an instance of `StaticObjectiveState`. Hence, we save performing the same computation over and over again.

31.2.4 The Evaluator

The reference implementation of the evaluator interfaces specified in Section 31.1.5 on page 456 can be found in the package `org.sigoa.refimpl.go.evaluation` depicted in Figure 31.18. `Evaluator` is the standard realization

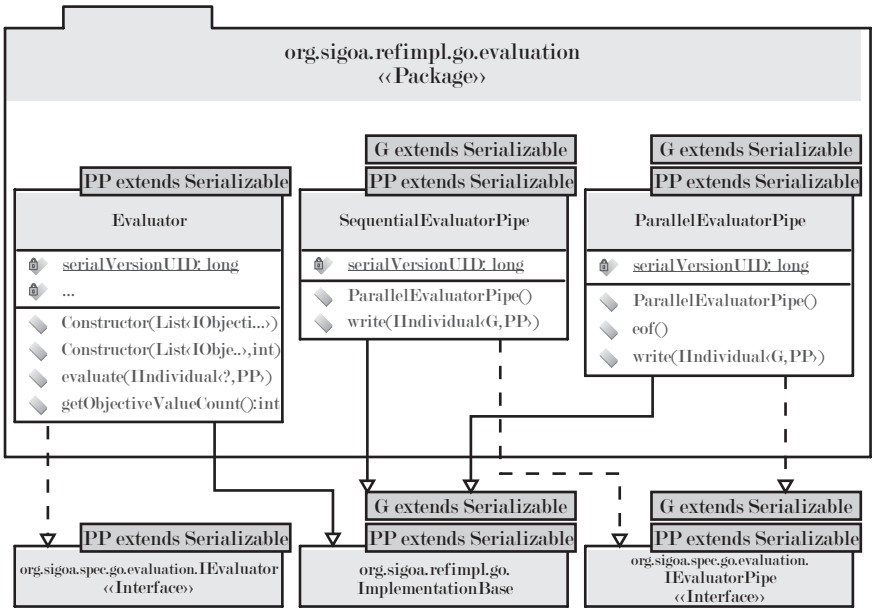


Fig. 31.18: The reference implementation of the evaluation interfaces.

of the interface `IEvaluator`. It is constructed with a list of objective functions

to work on along with an optional integer denoting the count of single evaluation runs to be performed per solution candidate. If this `int` is not provided, a single evaluation run is performed.

When created, the `Evaluator` divides the objective functions into groups, for each compatible simulation id. The simulation id compatibility is checked according to the elaborations in Section 27.2.3 on page 410: If simulation ids are classes, we can check their inheritance and find compatible ids not only by equal ids but also by the generalization/specialization in the class hierarchy. Each group of objective functions is dealt with separately when evaluating a solution candidate.

To allow for parallelization, the `Evaluator` builds containers which hold the objective function states and the static states. These containers are stored in a linked list. Whenever an evaluation is to be performed, the first container in the list is used. If none is available, a new one is created. After the evaluation, the containers again are inserted in to the list. The removing and adding of containers to the list are small critical sections which are synchronized. With this technique, the state objects of the objective functions are effectively reused.

We furthermore define two default evaluator pipes: `SequentialEvaluatorPipe` and `ParallelEvaluatorPipe`, both implementing the interface `IEvaluatorPipe`. While `SequentialEvaluatorPipe` evaluates each individual written to it directly in its `write`-method, `ParallelEvaluatorPipe` instead creates a new `java.lang.Runnable` which is inserted into the host's job queue via `executeJob` (see Section 28.1.3 on page 416). In its `eof`-method, it just has to use `flush` in order to ensure that all jobs are performed before propagating the `eof`-call. Each such job evaluates the solution candidate and subsequently writes the individual record to the next pipe stage.

31.2.5 Embryogeny

In Section 31.1.6 on page 456 we discussed the specification of the embryogenic operations – the transformation of genotypes into phenotypes. Like in the basic reproduction classes, we cannot specify any concrete functionality, since it will depend on the genotypes/phenotypes subsequently chosen/implemented. Figure 31.19 outlines the package `org.sigoa.refimpl.go.embryogeny` containing the basic embryogenic classes. The `EmbryogenyPipe` realizes the interface `IEmbryogenyPipe`. Each individual record to it is checked if its phenotype is `null`. If so, and if a non-`null` genotype is found, we use the method `hatch` of current embryogeny (obtained from host) to compute and set the phenotype to the individual record. The same is done vice versa for the genotype using the method `regress`, if needed.

Although implementing `IEmbryogeny`, `Embryogeny` does not provide real hatching/regressing functionality. Both methods simple assume that the genotype and phenotype are equal ($\tilde{X} = \mathbb{G}$) and return their parameter. For different genotypes/phenotypes, both have to be overridden.

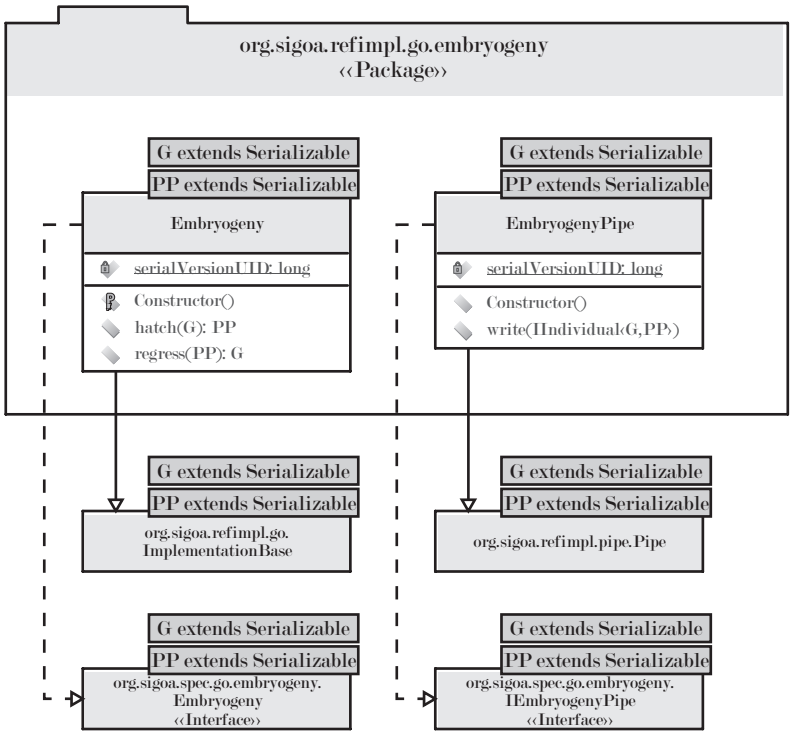


Fig. 31.19: The embryogeny classes.

31.2.6 Fitness Assignment

Figure 31.20 illustrates the contents of the package `org.sigoa.refimpl.go.fitnessAssignment` – some default fitness assignment algorithms. Fitness assignment algorithms have been introduced in Section 2.3 on page 65 and their functionality from the Sigoa point of view is specified in Section 31.1.7 on page 458. A fitness assigner is basically a pipe stage that sets the field fitness of the individual records. Most often an algorithm needs access to the whole chunk of solution candidates and would thus be implemented as `BufferedPipe`. For this case, we specify a default base class to derive from: `FitnessAssigner` which both inherits from `BufferedPipe` and implements the interface `IFitnessAssigner`. In the other instances, where the global view onto the solution candidates is not required, the normal `Pipe`-class can be extended. In Table 31.3 you can find the fitness assignment algorithms predefined in the package `org.sigoa.refimpl.go.fitnessAssignment` along with references to the algorithms they realize. The `SumFitnessAssigner` is a special case of weighted sum fitness assignment where all weights equal 1 ($w_i = 1 \forall i \in [1, |F|]$).

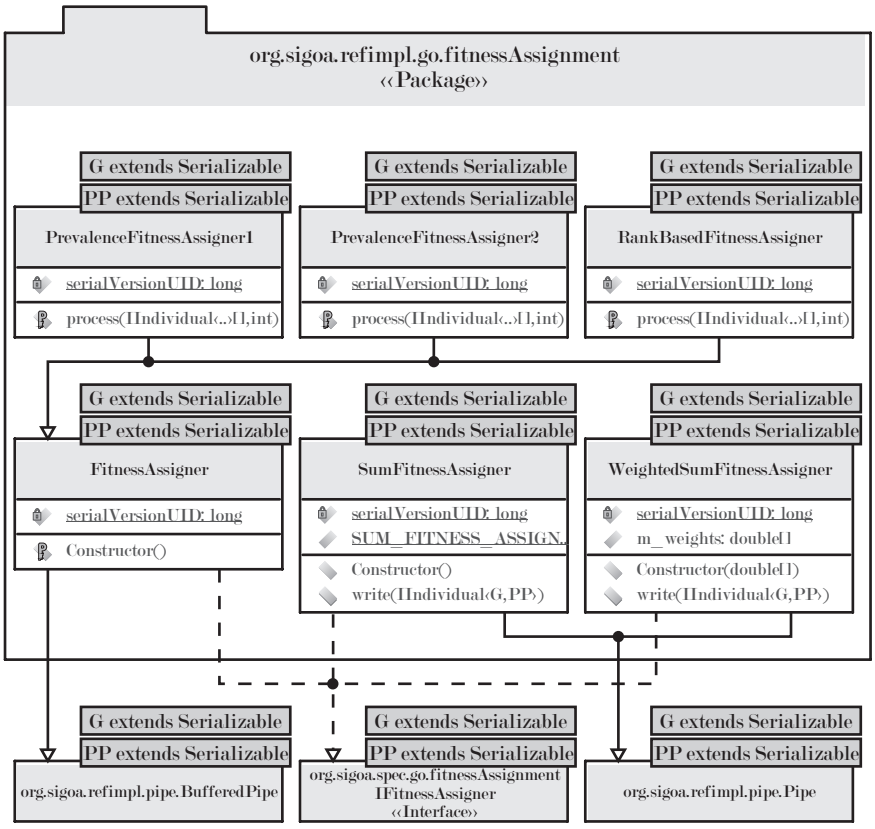


Fig. 31.20: Some default fitness assigners.

Table 31.3: The predefined fitness assigners.

class	realized algorithm	definition
`WeightedSumFitnessAssigner`	$weightedSumFitnessAssign$	Section 2.3.1 on page 66
`SumFitnessAssigner`	$weightedSumFitnessAssign$	Section 2.3.1 on page 66
`PrevalenceFitnessAssigner1`	$prevalenceFitnessAssign_1$	Section 2.3.2 on page 66
`PrevalenceFitnessAssigner2`	$prevalenceFitnessAssign_2$	Section 2.3.2 on page 66
`RankBasedFitnessAssigner1`	$rankBasedFitnessAssign$	Section 2.3.3 on page 67

31.2.7 Selection

In Section 31.1.7 on page 458 we have specified the interface `ISelectionAlgorithm` which is common to all pipe stages that perform a selection. Their reference realization is found in the package `org.sigoa.refimpl.go.selection` shown in Figure 31.21. Since `ISelectionAlgorithm` inher-

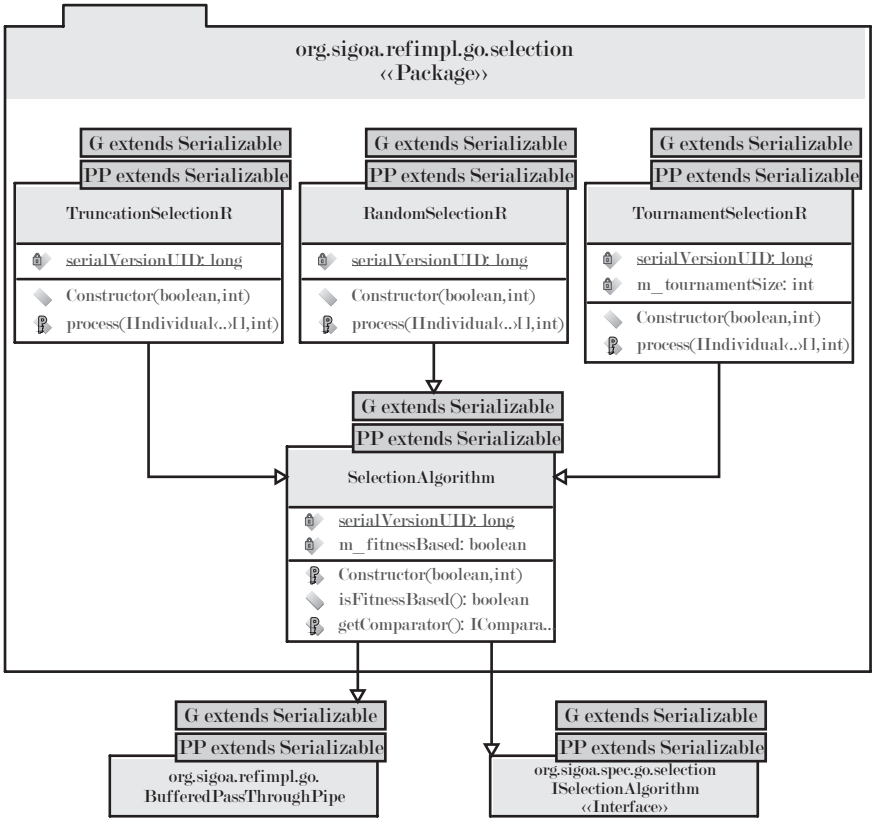


Fig. 31.21: The predefined selection algorithms package

its from `IPipe` and `IPassThroughParameters`, the base class of the selection algorithms of the reference implementation, `SelectionAlgorithm`, inherits from `BufferedPassThroughPipe`, a pipe that provides the pass-through parameters. Selection algorithms may either be based on prevalence comparison or on a preceding fitness assignment process. The constructor of `SelectionAlgorithm` takes therefore a `boolean` parameter `fitnessBased` which determines which of the both possibilities is valid. The method `getComparator` inherited from `ImplementationBase` is overridden accordingly: If `fitnessBased`

is `true`, the `getComparator` will return `org.sigoa.refimpl.go.comparators.FitnessComparator.FITNESS_COMPARATOR` (discussed in Section 31.2.1 on page 462) and the current comparator otherwise.

Table 31.4: The predefined selection algorithms.

class	realized algorithm	definition
`TruncationSelectionR`	$truncationSelect_r$	Section 2.4.1 on page 80
`RandomSelectionR`	$rndSelect_r$	Section 2.4.2 on page 80
`TournamentSelectionR`	$tournamentSelect_{r,k}$	Section 2.4.3 on page 81

31.2.8 The Optimizer

In Section 31.1.8 we have specified the optimization interfaces. Their reference implementations can be found in the root package of the global optimization system, `org.sigoa.refimpl.go`, as illustrated in Figure 31.22. The class `Optimizer` is the foundation of all implementations of global optimization algorithms (at least, of those relying on the Sigoa reference implementation).

A realization of `IOptimizer` is always also a realization of `java.lang.Runnable` and `IPipeOut`, since `IOptimizer` inherits from them. `Optimizer` furthermore additionally implements `IAdaptable` (see Section 23.1 on page 387), `IActivity` (see Section 28.1.1 on page 413), and, by extending `Pipe`, also `IPipe` (see Chapter 29 on page 427).

The work of an optimization algorithm is always performed in the method `doRun` which is called by `run` (of `Runnable`). This indirection is needed in order to support the `IActivity` operations. Since `Optimizer` already inherits from `Pipe`, it cannot also extend `Activity`. Anyhow, in order to provide the correct semantics and functionality of its methods, we use an internal class that derives from `Activity`. The calls to `abort`, `isRunning`, `isFinal`, `isTerminated`, `waitFor`, and `finished` are then deferred to an instance of this class. In turn, the methods `doStart` and `doAbort` of the `Optimizer`. The `Optimizer`-instance calls `start` of its member variable at first in its `run` method before invoking `doRun` and `finished` afterwards. This more or less complex approach helps us to achieve two goals: we can provide `doStart`, `abort`, `doAbort`, `finished`, `isRunning`, `isFinal`, `isTerminated`, and `waitFor` with exactly the same semantics as in the class `Activity` introduced in Section 28.2.1 on page 420 but do not need to re-implement them and thus ensure their consistent behavior. They can be used or overridden in order to introduce new functionality in exactly the same way as in `Activity`. The only difference is that there is no method `start` since an `Optimizer` is not a independent activity with own threads but run by a job system. The functionality of `start` is thus part of the `run`-method, as already stated.

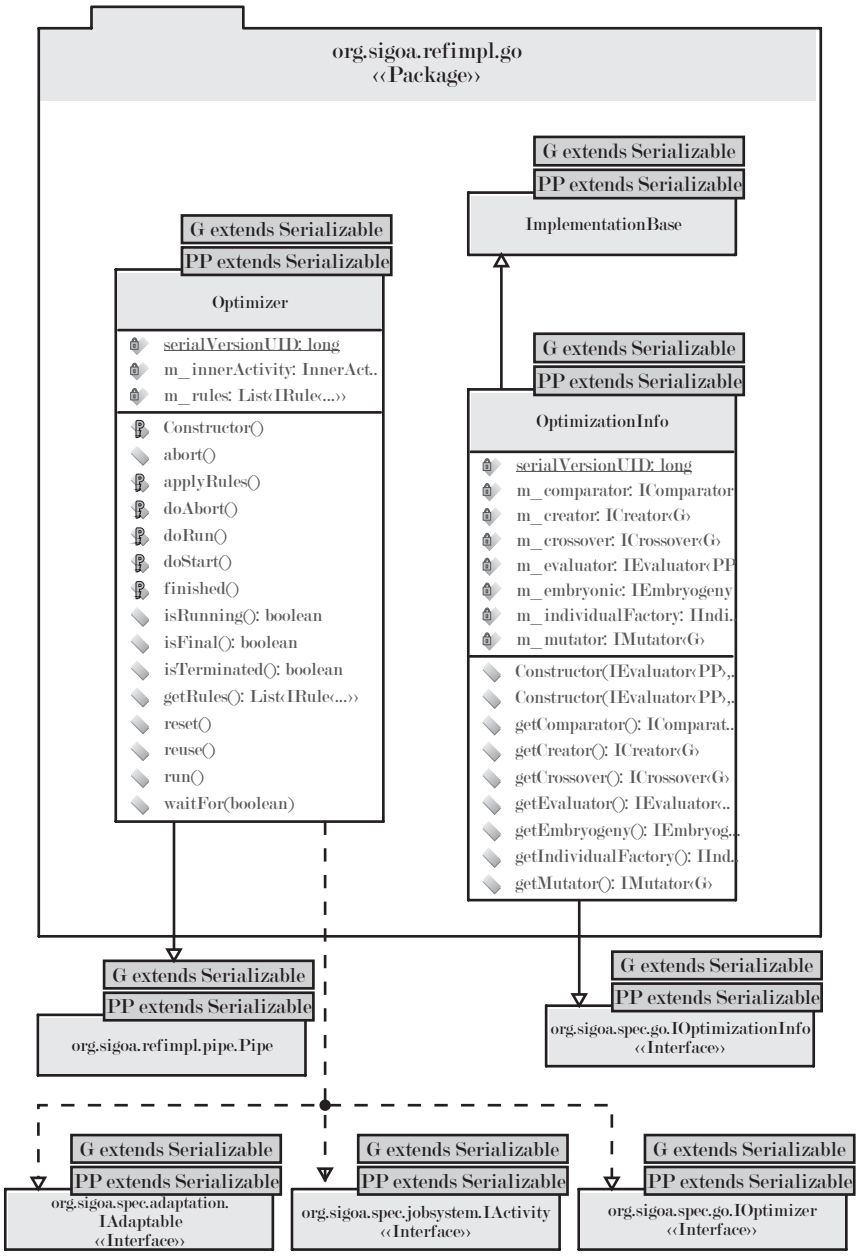


Fig. 31.22: The classes `Optimizer` and `OptimizationInfo`.

The second advantage is that we can mage `Optimizer`s reusable. If a normal activity reaches the state `TERMINATED`, it also reaches the end of its lifecycle. `Optimizer` however comes with the method `reuse` which simple replaces the internal activity by a new one – and thus brings the optimization algorithm back into the state before its `run`-method was executed. Of course, doing so is only possible if being in the state `TERMINATED`, reusing an activity that is currently running or terminating will lead to a `java.lang.IllegalStateException` being thrown. The method `reuse` needs to be overridden if internal data structures need to be reset when putting the algorithm back into its initial state.

The semantics of `reset` are a little bit different from `reuse`. `reuse` will put the whole `Optimizer` into its initial state, including the rules, metadata like iteration counters, all statistic information and so on. `reset` only clears its current state. Imagine the following scenario: All `Optimizer`s implement the interface `IPipeOut` in order to output their solutions when done. An optimizer may also be used to improve and refine a single solution candidate – there is no rule against that. So basically, one may want to chain optimizers together to form hybrid algorithms. An evolutionary algorithm may use a hill climber in order to refine its solutions. The hill climber would then be executed as a sub-job of the evolutionary algorithm once for each individual and could for example be limited in the count of iterations to perform. It is no problem to write a special pipe for that. In this case, one would need to `reuse` the hill climber multiple times. On the other hand, an evolutionary algorithm or a hill climber may reach a dead end, where no further improvements are possible. A rule could detect that no progress is made and `reset` the algorithm. `reset` could still preserve the best individuals and it could leave some counters, statistics, and such and such untouched. This is not what we want when reusing it. It is however imaginable to combine both scenarios.

`reuse` always needs to clear at least the same fields and data structures as `reset`, thus it invokes `reset` per default.

Since `Optimizer` implements `IAdaptable`, it also manages a set of rules. This set can be accessed by the method `getRules`, which returns an instance of `java.util.List` containing them. If the `protected` method `applyRules` is invoked, all the rules in the list are applied to the `Optimizer` as defined in Chapter 23. It should periodically be invoked in the method `doRun` of the `Optimizer`.

An `Optimizer` is a special `Pipe`. When it is done with its work, the best solution candidates found will be written to its output. This must be done by its subclasses, and the best point to do it is by overriding the method `finished`.

But a pipe has also an input end. An `Optimizer` may not support receiving new solution candidates as input in order to incorporate them into its course. This can be very useful for a number of reasons:

- It would allow optimization algorithms to be effectively chained, like in the scenario discussed before. An `Optimizer` used by another receives the solution candidates it should optimize through its input. It will then be ex-

ecuted as a sub job, writing its results to its output when done. This could be realized now very simple as a pipe stage. By this approach, arbitrary optimizers could be concatenated in arbitrary levels.

- When distributing optimizers over a network, many population based algorithms may run in parallel. These parallel instances could exchange individuals using for example the island model paradigm (see Section 16.2.2 on page 267). From time to time, they would select some solution candidates and send them to another algorithm instance on another machine. This other instance could incorporate them asynchronously by simple writing them to the input of the optimization algorithm.
- Integrating individuals received over the network into an optimization process is just a special case, if viewed more closely. By providing the `IPipeIn`-capabilities, any source of solution candidates can be utilized. It becomes for example also possible to store individual records to a file and later read them again directly into the optimizer, as maybe useful for snap-shooting.

31.2.9 The Optimization Info Record

The class `OptimizationInfo` is the reference implementation of the optimization information record specified in Section 31.1.9 on page 460. It stores all properties which can be accessed through getters in member variables which are initialized by the constructors. Basically, all of them are `final` and hold parameters passed into the constructors. The constructors thus require a creation, a mutation, and a crossover operation, an evaluator, and an embryogeny to be passed to them. The difference between the two constructors is that one also takes an individual factory and a comparator as argument, whereas the other one simple uses the default individual factory (see Section 31.2.1 on page 462) and the pareto-comparator (specified in Section 31.2.1 on page 462).

31.3 Predefined Algorithms

In this section we discuss some of the algorithms provided which are based on the predefined algorithm interfaces defined in Section 31.1.10 on page 460. As indicated by Figure 31.23, the package `org.sigoa.refimpl.go.algorithms` contains the class `IterativeOptimizer` realizing the interface `IIterativeAlgorithm`. The method `doRun` of `IterativeOptimizer` now performs a loop. In each step, the methods `applyRules`, `beforeIteration`, `iteration`, and `afterIteration` are called in exactly that order. `applyRules` already has been discussed in Section 31.2.8 on the facing page. `beforeIteration`, `iteration`, and `afterIteration` per default do nothing – they are to be overridden by derived classes in order to implement some useful behavior. They all receive the index of the current iteration as parameter.

At the end of each iteration step, this current iteration index as well as the count of total iterations is incremented by one. They are both zero-based

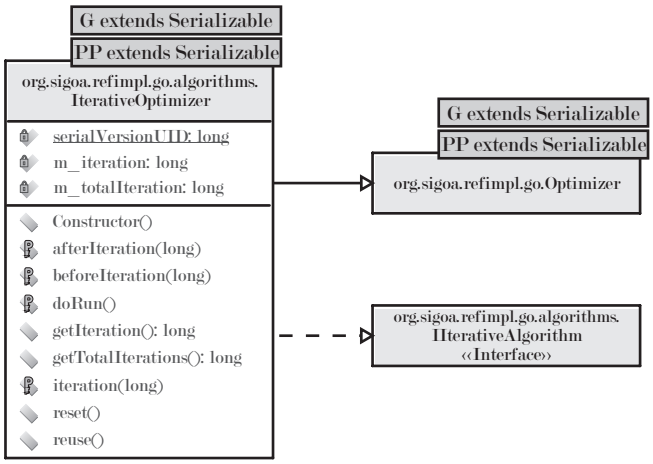


Fig. 31.23: The classes `IterativeOptimizer`.

(meaning that first iteration has the index zero) and can be obtained by the methods `getIteration` and `getTotalIterations` respectively. The difference between the two counters is that the current iteration index will be set to zero if by `reset` whereas the total iteration counter remains untouched. This allows rules to reset optimization algorithms any number of times while still preserving knowledge of the total count of iterations completed. This total iteration count is, of course, also set to zero again by `reuse`.

31.3.1 Implementing Evolutionary Algorithms

The package `org.sigoa.refimpl.go.algorithms.ea` contains implementations of evolutionary algorithms. The two classes `EA` and `ElitistEA`, sketched in Figure 31.24, are the Sigoa foundation for evolutionary algorithms.

`EA` - the Evolutionary Algorithm Implementation

Figure 31.25 shows that both are basically concatenations of pipes. The internal pipeline of the `EA` objects can be accessed by calling the protected method `getPipeline` and is built in the method `createPipeline`, which in turn uses the following methods:

1. `createFitnessAssigner` creates the fitness assignment algorithm to be used. It returns per default an instance of `PrevalenceFitnessAssigner2` (see Table 31.3 on page 476).
2. `createSelectionAlgorithm` returns the selection algorithm to be used – a binary tournament selection with replacement (see `TournamentSelectionR` in Table 31.4 on page 478) is created per default.

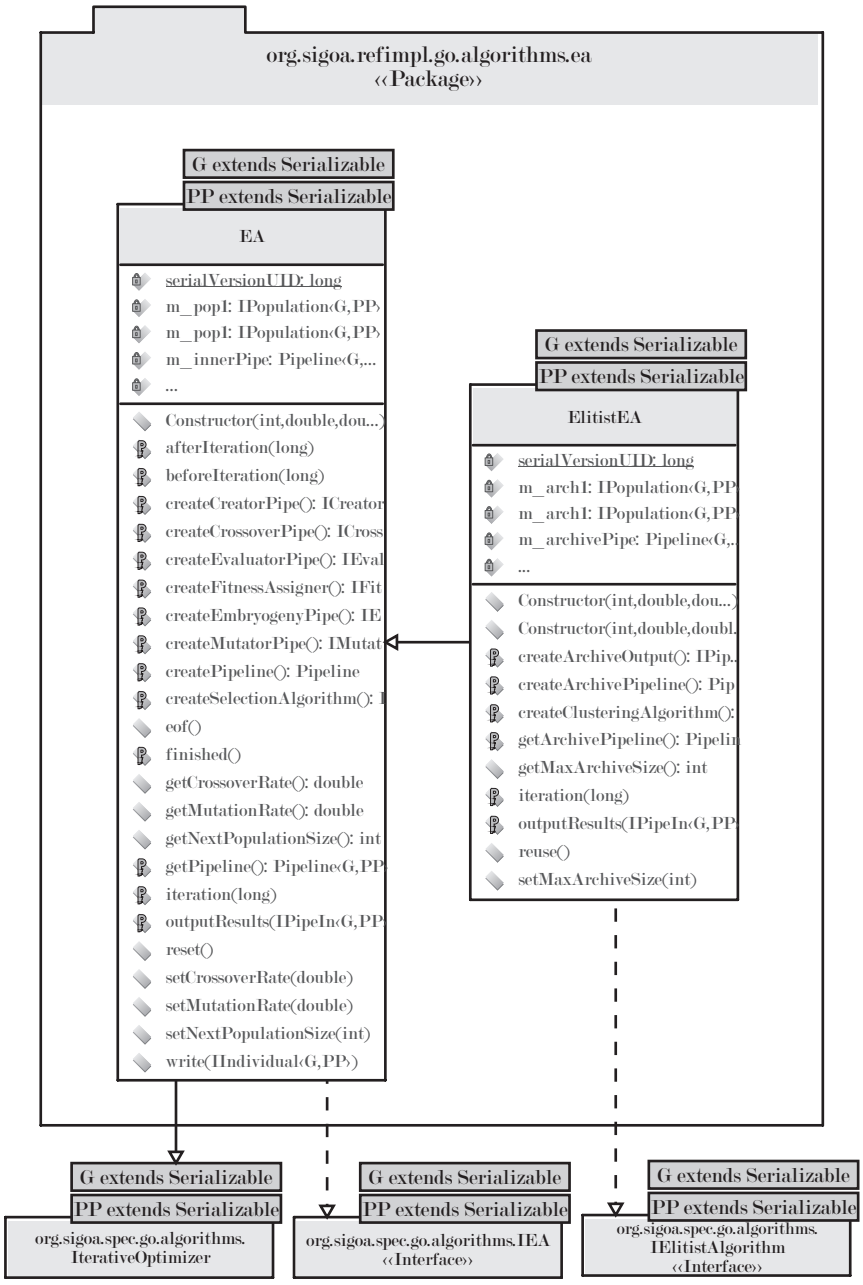


Fig. 31.24: The default evolutionary algorithm implementations of Sigoa.

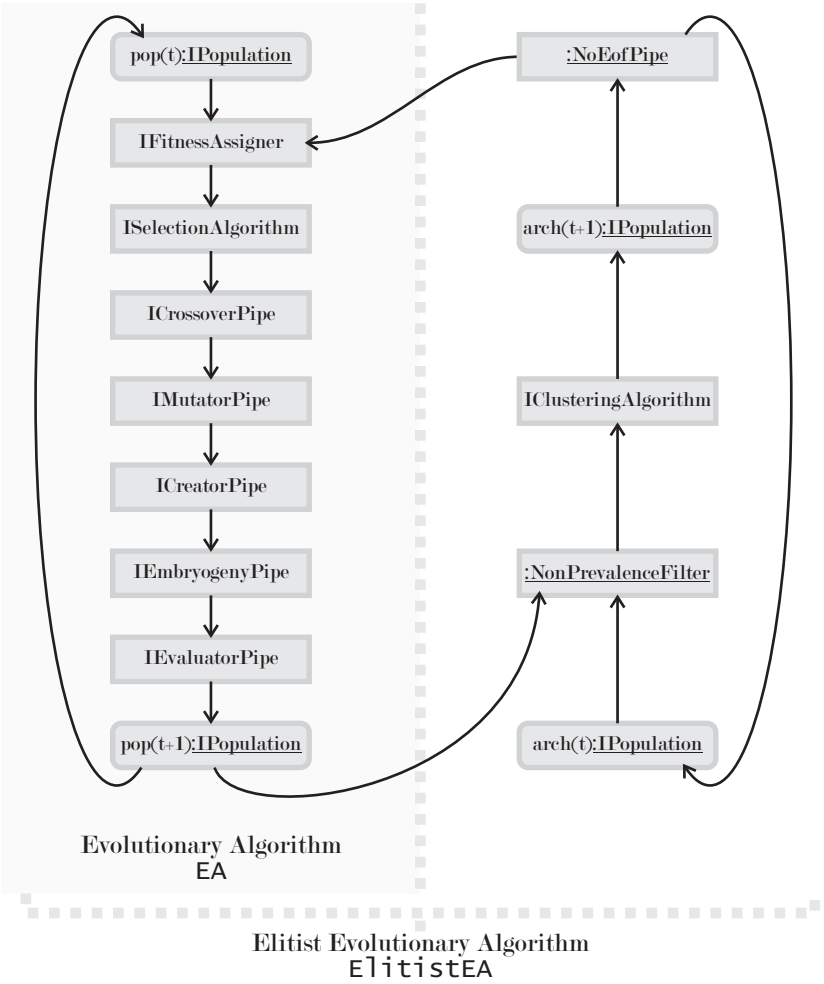


Fig. 31.25: The individual flow through a default EA pipe

3. `createCrossoverPipe` is used to obtain an instance of `ICrossoverPipe` (see Section 31.1.2). The standard implementation of this returns an `CrossoverPipe`.
4. `createMutatorPipe` returns a new `IMutatorPipe` (see Section 31.1.2), an instance of `MutatorPipe` per default,
5. `createCreatorPipe` provides instances of `ICreatorPipe` (again, see Section 31.1.2). In its original form, this method returns a brent new `CreatorPipe`.

6. `createEmbryogenyPipe` returns the `IEmbryogenyPipe` to be used by the evolutionary algorithm. It returns an `Embryogeny`-object per default (see Section 31.1.6 on page 456).
7. `createEvaluatorPipe` creates an instance of `IEvaluatorPipe`. The standard implementation returns an `SequentialEvaluatorPipe`-object – the standard evolutionary algorithms make no use of parallelism or multiple processors. If a parallel evolutionary algorithm is required, this method should be overridden in order to return an instance of `ParallelEvaluatorPipe` (see Section 31.2.4 on page 474) instead.

Figure 31.25 on the preceding page shows how the individuals flow through the pipe construction built by these methods. One may be surprised that the fitness assignment is the first stage in the pipe. In the first iteration, the population $pop(0) = \emptyset$ is empty, and thus, no individuals pass the stages up to the creator pipe. This stage creates the count of individuals needed to fill up to the wanted population size and passes them on to the embryogeny and the evaluator. In the second iteration step, the individuals now enter the fitness assignment and will subsequently take part in a selection. Thus, constructing the pipeline this way is no error but correct and efficient.

Although we work with populations called $pop(t)$ and $pop(t + 1)$ in Figure 31.25, in fact only two instances of `IPopulation` are needed: One that is written to the input end of the pipe and one to receive its outputs. These two are swapped after each iteration step.

In order to perform the work of the evolutionary algorithm, the methods `afterIteration`, `beforeIteration` and `iteration` are overridden. `afterIteration` calls `flushJobs` of its host, so all pending jobs in the queue will be finished by this method.

We implement the `IEA` interface in `EA` and thus need to specify methods to get and set mutation and crossover rates as well as the next population size are required. This is done be first holding these in internal variables initialized first in the constructor. While the getters just return the values of these variables, the setter methods have to browse the pipeline for stages that need these values. The population size setter, `setNextPopulationSize`, passes the new value to all instances of `IPassThroughParameters` (that are, for example selection algorithms and creator pipe stages). Notice the current population size returned by `getPopulationSize` may differ from the values set using `setNextPopulationSize` since these become operative not before the next iteration – and even if so, the real population size also depends on the construction of the pipeline. `setMutationRate` and `setCrossoverRate` propagate the new values to `IMutatorParameters` and `ICrossoverParameter` instances respectively.

When an evolutionary algorithm has finished, its method codeilifinished is invoked. Now it is time to find the best solutions created so far and write the to the output of the optimizer. `finished` therefore creates a `NonPrevalenceFiler` attached to a `NoEofPipe`. The output of this structure is than attached to

the output of the output of the evolutionary algorithm while its input end is passed to `outputResults`. The method `outputResults` is now to write all the individual records it knows to the `IPipeIn` instance it receives as parameter. In standard `EA` instances, these are the contents of the two, internally used populations.

`ElitistEA` - the Elitist Evolutionary Algorithm Implementation

`ElitistEA` extends `EA` by an additional internal pipeline, the archive pipeline. The archive pipeline can be accessed via `getArchivePipeline` and is built using the method `createArchivePipeline`. `createArchivePipeline` simple attaches the result of `createClusteringAlgorithm`, an instance of `IClusterAlgorithm` (an instance of `NNearestNeighborClustering` with $n = \sqrt{|archive|}$ per default, see Section 30.2.1 on page 439) to the output of a `NonPrevalenceFilter` (since archives normally contain only non-prevailed individuals).

Over a `CopyPipe`, the output of the archive is copied to an `NoEofPipe` which writes it to fitness assigner inside the main population pipeline. Both, the individuals from this main pipe as well as those residing in the archive of the last iteration step enter the archive pipe in its input end. Only the non-prevailed are will pass the first filter. If too many individuals remain, the clustering algorithm reduces them to a smaller but still significant set according to the archive size set and passes them to the next level. In the standard elitist EA, this is the end of the archive pipeline, attached to the archive of the current time step.

Like in `EA` objects, we only need two instances of `IPopulation` (which are switched in each iteration step) in order to represent the archives.

The size of the archive (initially $\sqrt{|population|}$) can be set using `setMaxArchiveSize` and is available via `getMaxArchiveSize`. The real size of the archive, obtained by using `getArchiveSize`, may be lower – it can only contain as much individuals as are non-prevailed at most.

32

Genotypes

Figure 32.1 gives an overview about the package `org.sigoa.refimpl.genotypes` containing the standard genotypes and phenotypes provided by the Sigoa framework. In this chapter, we want to elaborate on these default genotypes,

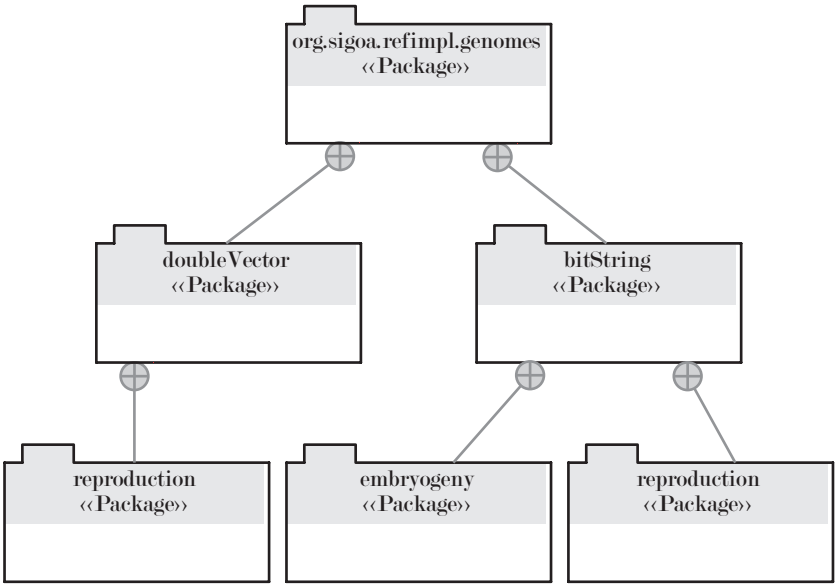


Fig. 32.1: The utility classes of the Sigoa reference implementation.

phenotypes, and the operations defined on them.

32.1 Vectors of Real Numbers

The package `org.sigoa.refimpl.genomes.doubleVector` provides vectors of real numbers, in other words, arrays of `double`, as genotype and phenotype. In Figure 32.2 we illustrate the classes that concentrate on the phenotypic aspects of this `doubleArray` genome. If the phenotype is the set of real numbers, i. e.

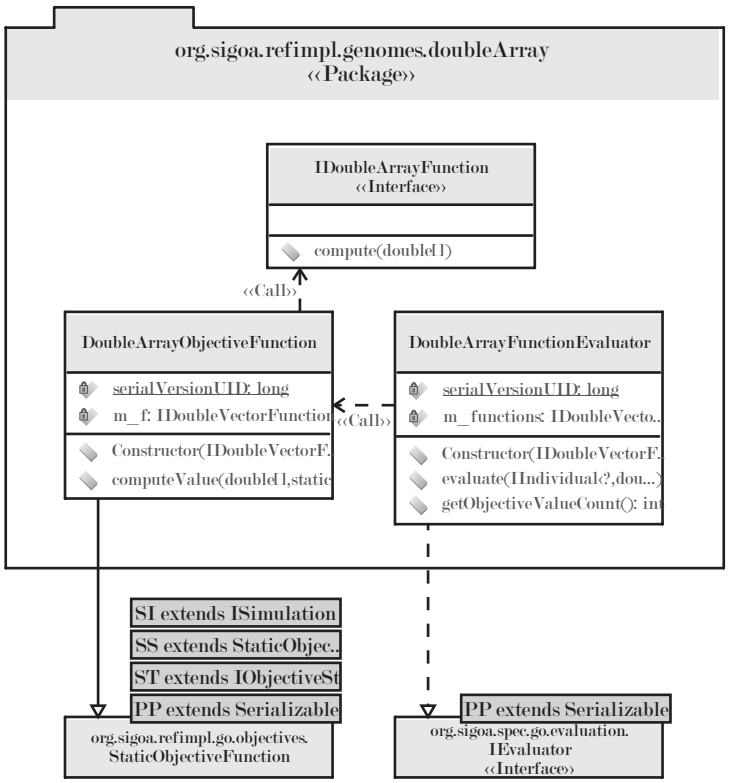


Fig. 32.2: The phenotypic aspects of the double array genome.

$\tilde{X} = \mathbb{R}^n$, the objective functions are often mathematical functions. In many of these cases the optimization algorithm is just used to minimize/maximize them [1110, 1111, 1112, 226, 1113].

32.1.1 The Evaluation Scheme for Functions of Real Vectors

For this reason, we provide an interface which encapsulates such mathematical functions $y \in \mathbb{R} = f(x) : x \in \mathbb{R}^n$. The method `compute` of the interface

`IDoubleArrayFunction` accepts an array of `double`, the closest we can get to $\mathbb{R}^n$ in Java, and returns again a `double`.

An interesting fact is that in the case of sole function optimization, the `Evaluator` and the whole concept of objective functions and simulations as illustrated in Figure 31.4 on page 454 would be a total overkill. A simplified evaluation process is therefore provided by the class `DoubleArrayFunctionEvaluator`. Instead of performing stepwise evaluation and using `IObjectiveFunction`s (see Section 31.2.3 on page 470), it simple owns a list of `IDoubleArrayFunction`s. The length of this list is also the count of objective values. Whenever an individual (in this case, a `double[]`) is evaluated, these functions are computed. The result of each function is stored as an objective value in the individual record. This is, of course, much, much faster than the approach applied normally. Notice however that the length of the vectors of real numbers is not correlated in any way with the count of objective values – we may have five objective functions $f_1 \ldots f_5(x) : x \in \mathbb{R}^2$ where each one takes a real vector of the length 2 as argument.

There may be, however, occasions, where the computation of a function of vectors of real numbers is only part of the evaluation of an individual. The real vector could be a set of coordinates describing a certain structural component. An instance of `IDoubleArrayFunction` could compute its volume and thus material cost. Another objective function may however simulate the component and returns e.g. the lifetime expectancy. In this case, we cannot use the `DoubleArrayFunctionEvaluator` but have to stick with the good old `Evaluator` (see Section 31.1.5 on page 456). In order to make `IDoubleArrayFunction`s usable with the `Evaluator`, we define the class `DoubleArrayObjectiveFunction`. This is a descendent of `StaticObjectiveFunction` and its `computeValue` method is overridden in a way that it returns the result of the meth `compute` of the `IDoubleArrayFunction` it contains. Now we can combine the evaluation of instances `IDoubleArrayFunction` with normal objective functions easily.

32.1.2 Reproduction Operators for Real Vectors

In Figure 32.3 we outline the package `org.sigoa.refimpl.genomes.doubleArray.reproduction` containing the reproduction operators for real number vector genomes. An internal base class, `DoubleArrayReproducer`, which extends `ImplementationBase` takes two arrays of `double` as parameters in its constructor. Each array is as long as the real vectors in the genome. The first array represents the lower border of the values of the vector and the second one the upper border. All operators that inherit from `DoubleArrayReproducer` can thus use this range information and confine the values in the produced genotype instances accordingly. Let us assume that we have the set of real vectors $X = \mathbb{R}^3$. We want to investigate a subset of this three dimensional space by defining the genome $\tilde{X} \subset X$ that consists of the vectors $x = (x_1, x_2, x_3) \in \tilde{X} \Leftrightarrow x_1 \in [0, 1] \wedge x_2 \in [-100, 100] \wedge x_3 \in [4, 5]$. Hence, we will provide the constructor of the reproduction operator with the two arrays

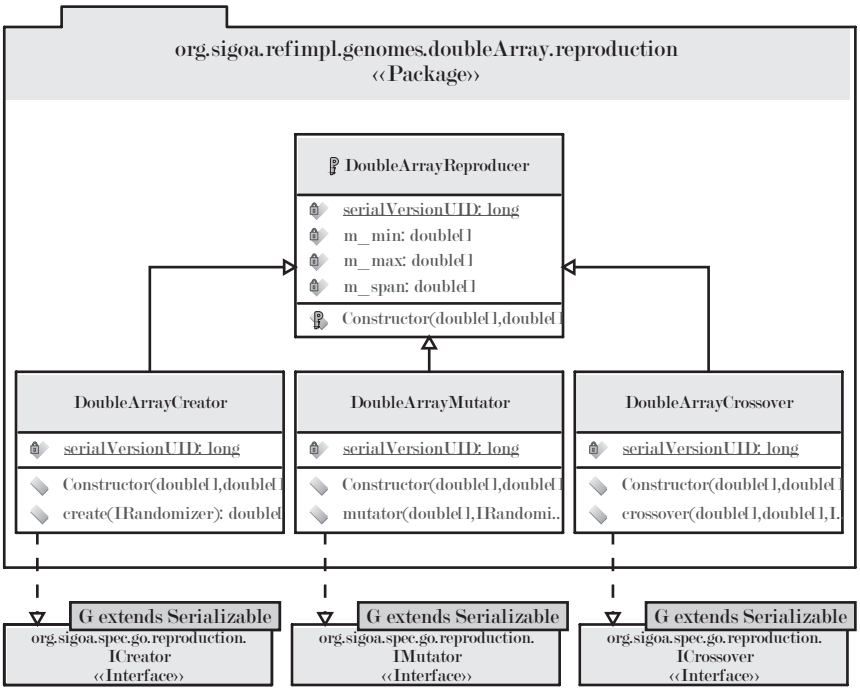


Fig. 32.3: The reproduction operators for the double array genome.

`{0,-100,4}` and `{1,100,5}`. The semantics of the operator guarantee that all vectors created will be in $\tilde{X}$.

The class `DoubleArrayCreator` is the first of such reproduction operators. The real vectors it creates are uniformly distributed over the confined space $\tilde{X}$.

The `DoubleArrayMutator` selects one place of the `double` array passed in. This place will undergo a mutation by computing a random number that is normally distributed (see Section 35.4.2 on page 537). The normal distribution has two parameters, μ, denoting its center, and σ, the standard deviation, describing how much the distribution function is stretched. If we set μ to the original value of the vector place that we want to mutate, the `IRandomizer` will us return a value that is spread around this value. σ is set using a bit more complicated strategy: we define some standard σ-values according to the possible range of the vector place. From these, we chose randomly.

The crossover operator for `double` vectors, `DoubleArrayCrossover`, utilizes two different approaches. It creates a new real vector $x_n \in \tilde{X}$ from two existing ones $x_1, x_2 \in \tilde{X}$. For each place i in the vector, it decides whether $x_{n,i}$ should be $x_{1,i}$, $x_{2,i}$, or $(\gamma x_{1,i}) + ((1-\gamma)x_{2,i}$ randomly. In the last case, $\gamma \in [0,1)$ is

a uniformly distributed random number $\gamma = random_u()$ and $x_{n,i}$ would thus be assigned to a weighted mean of $x_{1,i}$ and $x_{2,i}$.

32.2 Bit String Genomes

Bit string genomes are a special case of the string chromosomes discussed in Section 3.4 on page 124. Inside of the Java environment, we can define bit strings as arrays of `byte`, where each `byte` holds eight bits.

Other than real vectors, bit strings rarely are used as phenotypes and represent in most cases the genotypes only. Thus, embryongenesis, the encoding and decoding of information into and from the genome, is an important aspect here.

32.2.1 Encoding and Decoding Data in Bit String Genomes

In the package `org.sigoa.refimpl.genomes.bitString`, sketched in Figure 32.4, therefore contains versatile helper classes. Instances of the class `BitStringInputStream` can be initialized with an array of `byte` via the two `init`-methods. This `byte` array is then treated as a string of consecutive bits that can be read from. The same instance of `BitStringInputStream` can be reused and initialized multiple times. `BitStringInputStream` implements all the methods of the interface `java.io.DataInput` that read `byte`s, `short`s, `int`s, `long`s, `float`s, `double`s, `char`acters, character `String`s, and `boolean` values from the bit string. In `BitStringInputStream`, all these method use the routine `readBits`. `readBits` can be called with an `int`eger parameter with a value of $1 \ldots 32$, specifying how many bits should actually be read. It returns an `int` which is filled with bits read and increments the internal position counter accordingly. Analogously to the method `available`, which returns the count of remaining `byte`s in the stream, `availableBits` returns the remaining count of bits. To sum it up, with `BitInputStream`, all the primitive types of Java can be decoded from a bit string in any combination and order.

While `BitStringInputStream` allows the dencoding of all primitive types from a bit string, with `BitStringOutputStream` they can be encoded into one. By implementing `java.io.DataOutput`, `BitStringOutputStream` is, like `BitStringInputStream`, compatible to the Java I/O[1] and provides the methods needed to write data to a bit string. These methods all delegate to `writeBits` which takes an `int` containing the binary data to be written and another one holding the count of bits that should be used from that data. After writing all the information to be encoded to the `BitStringOutputStream`, the array of `byte` holding their binary representation can be obtained using the method `getOutput`. Notice that `getOutput` returns an array of `byte` and one thus cannot deduce the number of bits actually written from this return value

[1] `http://java.sun.com/javase/6/docs/technotes/guides/io/`

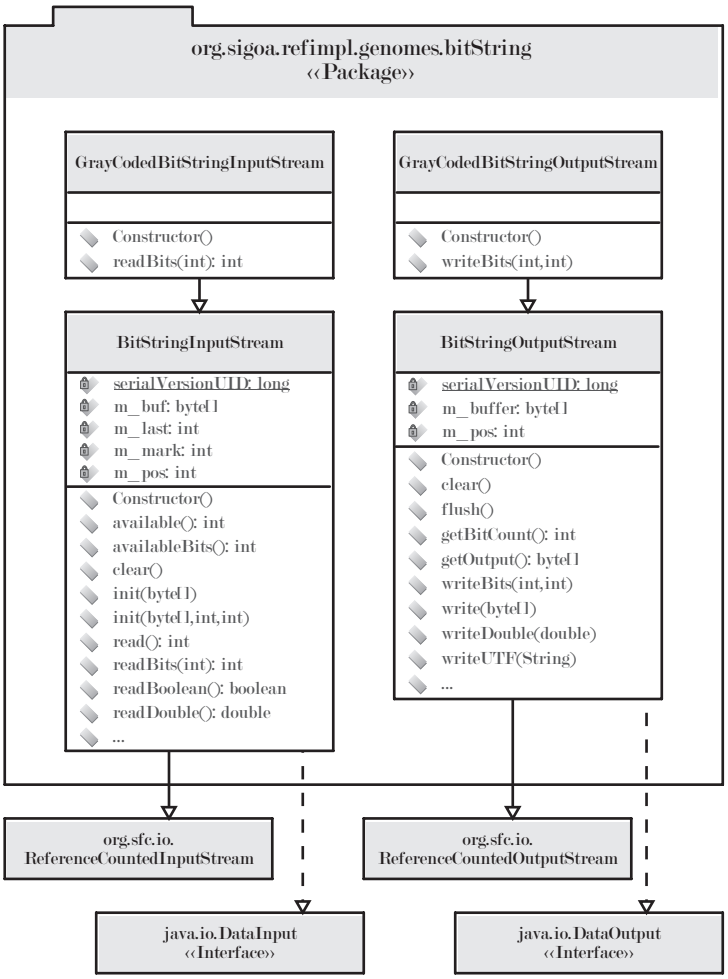


Fig. 32.4: Bit string encoding and decoding classes.

(since one `byte`) holds eight bits). Therefore, the method `getBitCount` is provided which returns this number. An instance of `BitStringOutputStream` can also be reused by simply deleting all the data written to it by invoking the method `clear`. The classes `BitStringInputStream` and `BitStringOutputStream` are compatible, meaning that any data encoded in a bit string using `BitStringInputStream` can be decoded correctly using `BitStringOutputStream`. Also, data decoded from a bit string with `BitStringInputStream` and re-encoded with `BitStringOutputStream` will produce exactly the same bit string again, with some minor exceptions (e.g. the `NaN` values of `double`).

Sometimes the usage of Gray-code for bit string genomes is of advantage, as discussed in Section 3.4 on page 124. The class `GrayCodedBitStringInputStream` extends `BitStringInputStream` for this purpose. In the overridden method `readBits`, it applies a Gray code to binary conversation after calling its inherited pendant and returns the result. `GrayCodedBitStringOutputStream` does the same with `BitStringOutputStream` – it encodes all bits written to it into Gray code before passing them on to the inherited `writeBits` method. Again, both classes are compatible and data written with one can correctly read by the other and vice versa.

32.2.2 Embryogeny of Bit String Genomes

In Figure 32.5, the `BitStringToDoubleArrayEmbryogeny` is introduced. It

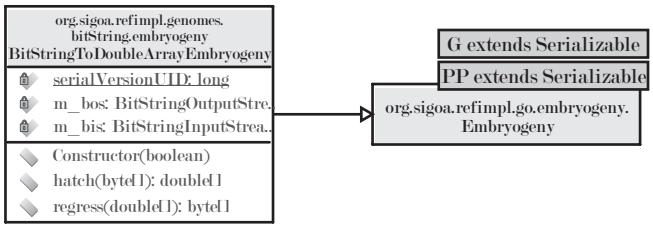


Fig. 32.5: The definition of `BitStringToDoubleArrayEmbryogeny`

builds the bridge of bit string genomes to the real vector phenotypes introduced in Section 32.1. This special embryogeny uses the aforementioned `BitStringInputStream` and `BitStringOutputStream` classes for the transformation. Its constructor can be supplied with a `boolean` value indicating if Gray code should be used or not.

32.2.3 Reproducing Bit Strings

In the package `org.sigoa.refimpl.genomes.bitString.reproduction` we specify some basic reproduction operations of bit strings. Figure 32.6 also includes the packages `fixedLength` and `variableLength` which contain specialized operators for bit strings of fixed and variable length.

Both sorts of bit strings however can be mutated by toggling one to n bits, as discussed in Section 3.4.1 on page 124. For this purpose, three basic mutation operators are provided:

1. The `BitStringToggleOneBitMutator` simple toggles a randomly picked bit in the genotype (by *xor*ing it with 1). The default, globally shared instance of this operator is the constant `BIT_STRING_TOGGLE_ONE_BIT_MUTATOR`.

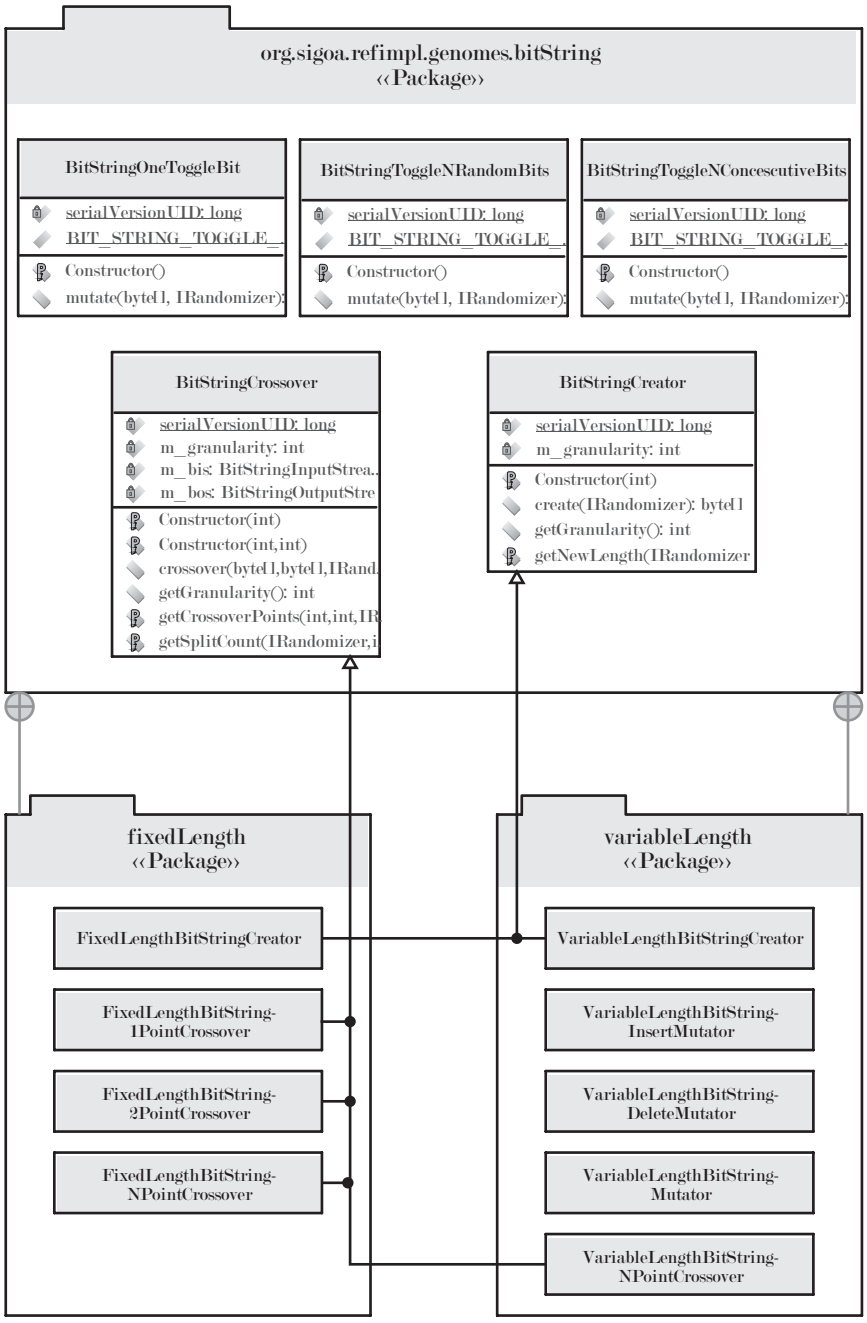


Fig. 32.6: The reproduction facilities for bit strings.

2. `BitStringToggleNRandomBits` repeats this operation n times, where n is picked randomly. It draws n locations in the string and toggles the bits at these locations. The default, globally shared instance of this operator is the constant `BIT_STRING_TOGGLE_N_RANDOM_BIT_MUTATOR`
3. `BitStringToggleNConsecutiveBits` toggles all bits in a consecutive group of the length n, where both n and the location of the group are picked randomly. The default, globally shared instance of this operator is the constant `BIT_STRING_TOGGLE_N_CONSECTIVE_BIT_MUTATOR`

The operations creation and crossover however differ for fixed and variable length strings. The base class `BitStringCreator` provides the means to create a randomly initialized bit string. Its method `getNewLength` has to be overridden in order to provide the length of the string to be created. Its descendant `FixedLengthBitStringCreator` here always returns the same number (specified in its constructor), whereas `VariableLengthBitStringCreator` returns a random value which is at least as big as a minimal size (also specified in its constructor). `BitStringCreator` has another important feature: it supports a granularity value which can be obtained via the `getGranularity` method. The granularity value (1 per default) is the measurement unit of the length of the bit strings. In other words, the string lengths will always be multiples of the granularity. This is realized by multiplying the values returned by `getNewLength` with the granularity before actually creating the new genotype.

Similar to `BitStringCreator`, the base class for bit string crossover supports a granularity value. Here, all crossover points will be at bit indexes that are multiples of this value. `BitStringCreator` specifies two `protected` methods that have to be overridden to define the functionality of the crossover operation: `getSplitCount` returns the number of crossover points and `getCrossoverPoints` fills their locations into arrays of `int` (one per parent). For fixed-length bit string genomes we base on the foundation of this class

1. single-point crossover operation in `FixedLengthBitString1PointCrossover`
2. two-point crossover operation in `FixedLengthBitString2PointCrossover`
3. n-point crossover operation in `FixedLengthBitStringNPointCrossover`

For variable-length bit strings we also define an n-point crossover operator in the class `VariableLengthBitStringNPointCrossover`. In the fixed-length forms, the crossover points are the same for both parents and thus, the child genome has the same length as the parent genomes. For variable-length genomes the crossover points may differ and so will the length of the child genome from the length of the parent genome.

If the length of the chromosomes is not fixed, two additional mutation operators become available as mentioned in Section 3.4.2 on page 126: the insertion and the deletion of bits. The first one is done in the `VariableLengthBitStringInsertMutator` and the later in the `VariableLengthBitStringDeleteMutator`. Again, both apply a granularity value which is used as a multiplier for the count of bits to insert/delete as well as for the insertion/deletion point.

To ease working with mutation of variable-length genomes, we specify the class `VariableLengthBitStringMutator` which is a specialization of `MultiplexingMutator` elaborated on in Section 31.2.2 on page 468. It utilizes all the operators defined here with a default probability distribution. An instance of this class can be created by specifying a granularity value which will be used to initialize the mutators internally used.

33

Utility Classes

In this chapter we want to have a closer look on the utility classes used by the Sigoa system.

33.1 The Utility Classes of the Reference Implementation

The utility classes of the reference implementation of the Sigoa reside in the package `org.sigoa.refimpl.utils` depictured in Figure 33.1.

33.1.1 The Default Thread Class

`DefaultThread` is a thread equipped with the `IActivity2`-interface (see Section 28.1.1). The worker threads of the job system for example are derived from this class, as described in Section 28.2.3. `DefaultThread` inherits from `org.sfc.parallel.SfcThread` which in turn is conform to the Sigoa activity model discussed in Section 28.1.1. Thus, we can provide the the methods `start`, `abort`, `waitFor`, `isRunning`, `isTerminated` and `isFinished` of `IActivity2` as well as the `protected doAbort` and `doStart` routines with the same semantics as in the class `Activity` of the job system. This is achieved already in `SfcThread` by utilizing the same technique as in the `Optimizer` of Section 31.2.8 on page 478 – an internal activity instance to provide their functionality. Additionally, the method `run` delegates now to the new `abstract` method `doRun`. This call is however encapsulated in a `try...catch`-clause. If an error is caught, it is passed to the method `onError`, which may then for example create an `ErrorEvent` or such and such, but per default only checks if the error had probably something to do with memory shortage, and, if so, invokes the garbage collector.

33.1.2 The Selector

The class `Selector` is used by the multiplexing reproduction operations in Section 31.2.2. A `Selector` is parameterized with the type `T` and takes an

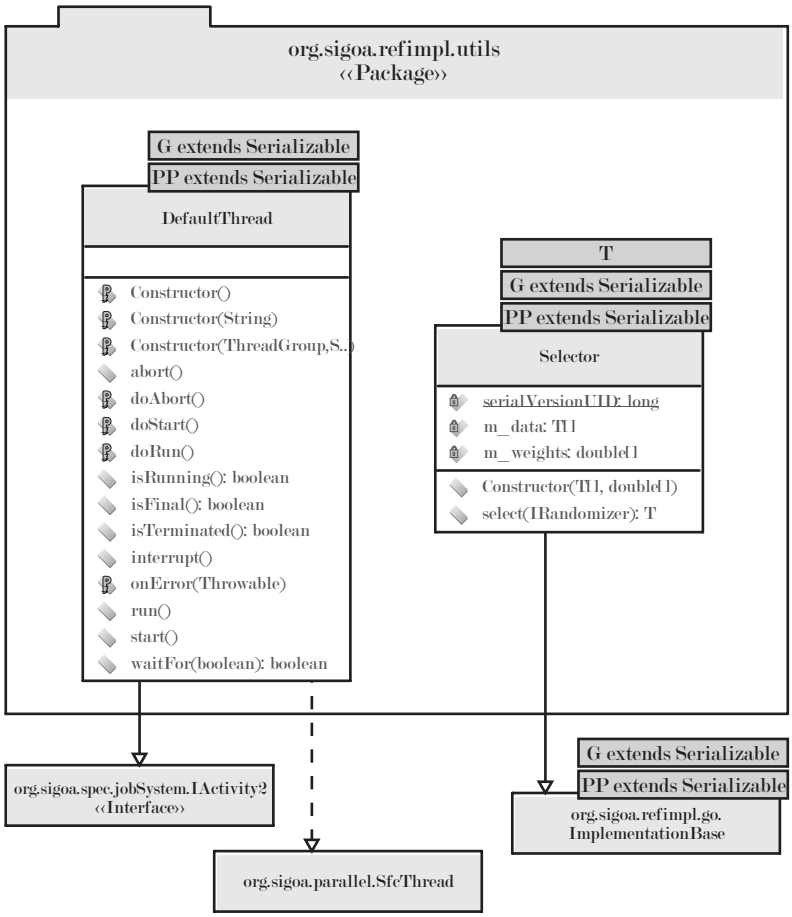


Fig. 33.1: The utility classes of the Sigoa reference implementation.

array of instances of `T` along with a same-length array of `double` (the weights) when being constructed. Each of this instances of `T` is assigned the `double` at the same position in the second array as weight. Whenever the method `select` is invoked, it randomly picks one of the `T`s with a probability proportional to its weight and returns it. Let us for example assume that `T` was `String`, the first array was "`A`","`B`","`C`", and that the second array was `1, 2, 3`. In one out of six calls to `select`, it would thus *probably* return "`A`", in two it would return "`B`" and in the rest of the cases, "`C`" would returned.

Part IV

Background

34

Set Theory

Set theory[1] [1114, 1115, 1116] is an important part of the mathematical theory. Numerous other disciplines like algebra, analysis and topology are based up on it. Set theory can be divided into naïve set theory[2] and axiomatic set theory[3]. The first form, the naïve set theory, is inconsistent and therefore not regarded in this book.

Definition 86 (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.

34.1 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. A set can contain an element only once. There are three common forms to define sets:

- With their elements in braces: $A = \{1, 2, 3\}$ defines a set A containing the three elements 1, 2, and 3.
- The same set can be specified using logical operators to describe its elements: $\forall b \in \mathbb{N} : (b \geq 1) \wedge (b < 4) \Leftrightarrow b \in B$.
- 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.

[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]

34.2 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 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.

$$A = B : x \in A \Leftrightarrow x \in B \tag{34.1}$$
$$A \neq B : (\exists x : x \in A \wedge x \notin B) \vee (\exists y : y \in B \wedge y \notin A) \tag{34.2}$$
$$A \subseteq B : x \in A \Rightarrow x \in B \tag{34.3}$$
$$A \subset B : A \subseteq B \wedge \exists y : y \in B \wedge y \notin A \tag{34.4}$$
$$A \not\subseteq B : \exists x : x \in A \wedge x \notin B \tag{34.5}$$
$$A \not\subset B : (A = B) \vee (\exists x : x \in A \wedge x \notin B) \tag{34.6}$$

34.3 Special Sets

Special sets used in the context of this book are

- The empty set $\emptyset = \{\}$ contains no elements ($|\emptyset| = 0$).
- The natural numbers $\mathbb{N}$ include all whole numbers bigger than 0. ($\mathbb{N} = \{1, 2, 3, ...\}$)
- The natural numbers including 0 ($\mathbb{N}_0$) include all whole numbers bigger than or equal to 0. ($\mathbb{N}_0 = \{1, 2, 3, ...\}$)
- $\mathbb{Z}$ is the set of all integers, positive and negative. ($\mathbb{Z} = \{..., -2, -1, 0, 1, 2, ...\}$)
- The rational numbers $\mathbb{Q}$ are defined as $\mathbb{Q} = \left\{ \frac{a}{b} : a, b \in \mathbb{Z}, b \neq 0 \right\}$.
- All real numbers are members of $\mathbb{R}$.
- $\mathbb{R}^+$ denotes the positive real numbers including 0 ($\mathbb{R}^+ = [0, \infty)$).

$$\mathbb{N} \subset \mathbb{N}_0 \subset \mathbb{Z} \subset \mathbb{Q} \subset \mathbb{R} \tag{34.7}$$
$$\mathbb{N} \subset \mathbb{N}_0 \subset \mathbb{R}^+ \subset \mathbb{R} \tag{34.8}$$

For these numerical sets, special subsets, so called intervals, can be specified. $[1, 5)$ is a set containing all the numbers starting from (including) 1 up to (exclusive) 5. $(1, 5]$ on the other hand contains all numbers bigger than 1 inclusive 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.

34.4 Operations on Sets

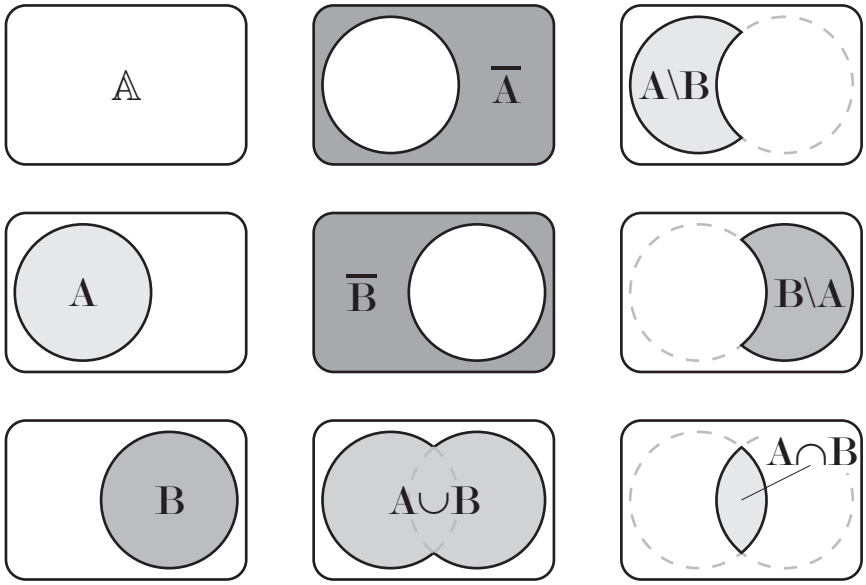


Fig. 34.1: Set operations performed on sets A and B inside a set $\mathbb{A}$

In this section we define the possible unary and binary operations on sets, some of which are illustrated in Figure 34.1.

Definition 87 (Set Union). The union[5] 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.

$$C = A \cup B \Leftrightarrow ((c \in A) \vee (c \in B) \Leftrightarrow (c \in C)) \tag{34.9}$$
$$A \cup B = B \cup A \tag{34.10}$$
$$A \cup \emptyset = A \tag{34.11}$$
$$A \cup A = A \tag{34.12}$$
$$A \subseteq A \cup B \tag{34.13}$$

Definition 88 (Set Intersection). The intersection[6] D of two sets A and B, denoted by $A \cap B$, contains all the objects that are elements of both of the

[5] http://en.wikipedia.org/wiki/Union_%28set_theory%29 [accessed 2007-07-03]
[6] http://en.wikipedia.org/wiki/Intersection_%28set_theory%29 [accessed 2007-07-03]

sets. If $A \cap B = \emptyset$, meaning that A and B have no elements in common, they are called *disjoint*.

$$D = A \cap B \Leftrightarrow ((d \in A) \wedge (d \in B) \Leftrightarrow (d \in D)) \tag{34.14}$$
$$A \cap B = B \cap A \tag{34.15}$$
$$A \cap \emptyset = \emptyset \tag{34.16}$$
$$A \cap A = A \tag{34.17}$$
$$A \cap B \subseteq A \tag{34.18}$$

Definition 89 (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.

$$E = A \setminus B \Leftrightarrow ((e \in A) \wedge (e \notin B) \Leftrightarrow (e \in E)) \tag{34.19}$$
$$A \setminus \emptyset = A \tag{34.20}$$
$$\emptyset \setminus A = \emptyset \tag{34.21}$$
$$A \setminus A = \emptyset \tag{34.22}$$
$$A \setminus B \subseteq A \tag{34.23}$$

Definition 90 (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:

$$A \subseteq \mathbb{A} \Rightarrow \overline{A} = \mathbb{A} \setminus A \tag{34.24}$$

Definition 91 (Cartesian Product). The Cartesian product[7] 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.

$$P = A \times B \Leftrightarrow P = \{(a, b) : a \in A, b \in B\} \tag{34.25}$$

Definition 92 (Countable Set). A set S is called countable[8] if there exists an injective function[9] $\exists f : S \Rightarrow \mathbb{N}$.

Definition 93 (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 94 (Power Set). The power set[10] $\mathcal{P}(A)$ is the set of all subsets of A.

$$\forall p \in \mathcal{P}(A) \Leftrightarrow p \subseteq A \tag{34.26}$$

[7] `http://en.wikipedia.org/wiki/Cartesian_product` [accessed 2007-07-03]
[8] `http://en.wikipedia.org/wiki/Countable_set` [accessed 2007-07-03]
[9] see definition of function on page 510
[10] `http://en.wikipedia.org/wiki/Axiom_of_power_set` [accessed 2007-07-03]

34.5 Tuples and Lists

A tuple[11] is an ordered, finite sequence of elements, each of a special type. Other than sets, tuples may contain the same element twice. We define tuples with parenthesis ((x)) whereas we define sets with braces ($\{x\}$). Each item of a tuple may have another type, $(Monday, 23, \{a, b, c\})$ for example is a valid tuple.

Definition 95 (Tuple Type). To formalize this relation, we define the tuple type T. T defines the basic sets for the elements of its tuples and we write $t \in T$ if a tuple t meets the constraints imposed to its values by T.

$$\begin{aligned} T &= \langle T_1, T_2, \dots T_n \rangle, n \in \mathbb{N} \\ t = (t_1, t_2, \dots t_n) \in T &\Leftrightarrow t_i \in T_i \forall 0 < i \leq n \end{aligned} \tag{34.27}$$

[List] Lists[12] are abstract data types which can bee regarded as special tuples or sets. They are sequences where every item is of the same type. 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. 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: $n = (a, b, c) = |(a, b, c)| = 3$).

Definition 96 ($createList$). The $l = createList(n, q)$ method creates a new list l of the length n filled with the item q. If a list of the length 0 is created, the parameter q may be omitted. Such a creation of an empty list could be abbreviated like $l = createList(0, 0) \equiv ()$.

$$l = createList(n, q) \Leftrightarrow |l| = n \wedge \forall 0 \leq i < n \Rightarrow l[i] = q \tag{34.28}$$

Definition 97 ($insertListItem$). The function $m = insertListItem(l, i, q)$ creates a new list m by inserting one element q in a list l at the index $0 \leq i \leq |l|$ shifting all the elements already in the list from this index on forwards.

$$\begin{aligned} m = insertListItem(l, i, q) \Leftrightarrow\ & |m| = |l| + 1 \wedge m[i] = q \wedge \\ & \forall j : 0 \leq j < i \Rightarrow m[j] = l[j] \\ & \forall j : i \leq j < |l| \Rightarrow m[j + 1] = l[j] \end{aligned} \tag{34.29}$$

Definition 98 ($addListItem$). The $addListItem$ method is a shortcut for inserting one item at the end of a list:

$$addListItem(l, q) \equiv insertList(l, |l|, q) \tag{34.30}$$

[11] `http://en.wikipedia.org/wiki/Tuple` [accessed 2007-07-03]
[12] `http://en.wikipedia.org/wiki/List_%28computing%29` [accessed 2007-07-03]

Definition 99 (*appendList*). The $appendList(l_1, l_2)$ method is a shortcut for adding all the elements of a list l_2 to a list l_1. We define it recursively as:

$$appendList(l_1, l_2) \equiv \begin{cases} l_1 & if\ |l_2| = 0 \\ appendList(addListItem(l_1, l_2[0]), & \\ \qquad deleteListItem(l_2, 0)) & otherwise \end{cases} \tag{34.31}$$

Definition 100 (*deleteListItem*). The method $deleteListItem(l, i)$ creates a new list m by removing the element at index $0 \leq i < |l|$ from the list l ($|l| \geq i + 1$).

$$\begin{aligned} m = deleteListItem(l, i) \Leftrightarrow\ & |m| = |l| - 1 \wedge \\ & \forall j : 0 \leq j < i \Rightarrow m[j] = l[j] \\ & \forall j : i < j < |l| \Rightarrow m[j-1] = l[j] \end{aligned} \tag{34.32}$$

Definition 101 (*deleteListRange*). The method $m = deleteListRange(l, i, c)$ creates a new list m by removing c elements beginning at index $0 \leq i < |l|$ from the list l ($|l| \geq i + c$).

$$\begin{aligned} m = deleteListRange(\ & l, i, c) \Leftrightarrow |m| = |l| - c \wedge \\ & \forall j : 0 \leq j < i \Rightarrow m[j] = l[j] \\ & \forall j : i + c \leq j < |l| \Rightarrow m[j-c] = l[j] \end{aligned} \tag{34.33}$$

Definition 102 (*countItemOccurrences*). The function $countItemOcurences(x, l)$ returns the number of occurrences of the element x in the list l.

$$countItemOcurences(x, l) = |\{i \in 0 \ldots |l| - 1 : l[i] = x\}| \tag{34.34}$$

Definition 103 (*subList*). The method $subListRange(l, i, c)$ extracts c elements from the list l beginning at index i and returns them as a new list.

$$subList(l, i, s) \equiv deleteListRange(deleteListRange(l, 0, i), c, |l| - i - c) \tag{34.35}$$

Definition 104 (Sorting). It is often useful to have sorted lists[13]. Thus we define the functions $S = sort_a(U, s)$ and $S = sort_d(U, s)$ which sort a list U in ascending and descending order using a comparison function $s(u_1, u_2)$ which 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. Sorting is done in $\mathcal{O}(n \log n)$ time. For concrete algorithm examples, see [1117, 957, 1118].

$$S = sort_a(U, s) \tag{34.36}$$

$$\forall u \in U\ \exists i \in [0, |U| - 1] : S[i] = u \tag{34.37}$$

$$|S| = |U| \tag{34.38}$$

$$\forall 0 \leq i < |U| \Rightarrow s(S[i], S[i+1]) \leq 0 \tag{34.39}$$

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

For $sort_d$, only Equation 34.39 changes, the rest stays valid:

$$S = sort_d(U, s) \tag{34.40}$$

$$\forall 0 \leq i < |U| \Rightarrow s(S[i], S[i+1]) \geq 0 \tag{34.41}$$

Definition 105 (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.

$$search_u(u, U) = \begin{cases} i : U[i] = u \; if \; u \in U \\ -1 \; otherwise \end{cases} \tag{34.42}$$

Definition 106 (Searching in Sorted Lists). Searching an element s in sorted list S means to perform a binary search[14] 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 $search_{as}$ searches in an ascending sorted list, $search_{ds}$ searches in a descending sorted list. Searching in a sorted list is done in $\mathcal{O}(logn)$ time. For concrete algorithm examples, again see [1117, 957, 1118].

$$search_{as}(s, S) = \begin{cases} i : S[i] = s \; if \; s \in S \\ (-i-1) : (\forall j \geq 0, j < i \Rightarrow S[j] \leq s) \wedge \\ \qquad (\forall j < |S|, j \geq i \Rightarrow S[j] > s) \;\; otherwise \end{cases} \tag{34.43}$$

$$search_{ds}(s, S) = \begin{cases} i : S[i] = s \; if \; s \in S \\ (-i-1) : (\forall j \geq 0, j < i \Rightarrow S[j] \geq s) \wedge \\ \qquad (\forall j < |S|, j \geq i \Rightarrow S[j] < s) \;\; otherwise \end{cases} \tag{34.44}$$

Definition 107 ($removeListItem$). The method $removeListItem$ 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).

$$m = removeListItem(l, q) \Leftrightarrow \begin{cases} l \; if \; search(q, l) < 0 \\ deleteListItem(l, search(q, l)) \; otherwise \end{cases} \tag{34.45}$$

We define transformation functions for sets and lists:

$$Y = setToList(set\ X) \Rightarrow \; \forall x \in X \; \exists i : Y[i] = x \wedge \\ \forall i \in [0, |Y| - 1] \Rightarrow Y[i] \in X \tag{34.46}$$

$$|tuple(X)| = |X| \tag{34.47}$$

$$X = listToSet(tuple\ Y) \Rightarrow \; \forall i \in [0, |Y| - 1] \Rightarrow Y[i] \in X \wedge \\ \forall x \in X \; \exists i : Y[i] = x \tag{34.48}$$

$$|set(Y)| \leq |Y| \tag{34.49}$$

[14] `http://en.wikipedia.org/wiki/Binary_search` [accessed 2007-07-03]

34.6 Binary Relations

Definition 108 (Binary Relation). A binary[15] relation[16] R is defined as an ordered triple (X, Y, G) where X and Y are arbitrary sets, and G is a subset of the Cartesian product $X \times Y$ (see Equation 34.25). The sets X and Y are called the domain and codomain, respectively, of the relation, and G is called its graph. The statement $(x, y) \in G$ is read "x is R-related to y" and is denoted by $R(x, y)$. The order of the elements in each pair of G is important: if $a \neq b$, then $R(a, b)$ and $R(a, b)$ can be `true`or `false`, independently of each other.

Some types and possible properties of binary relations are listed below and illustrated in Figure 34.2. A binary relation can be

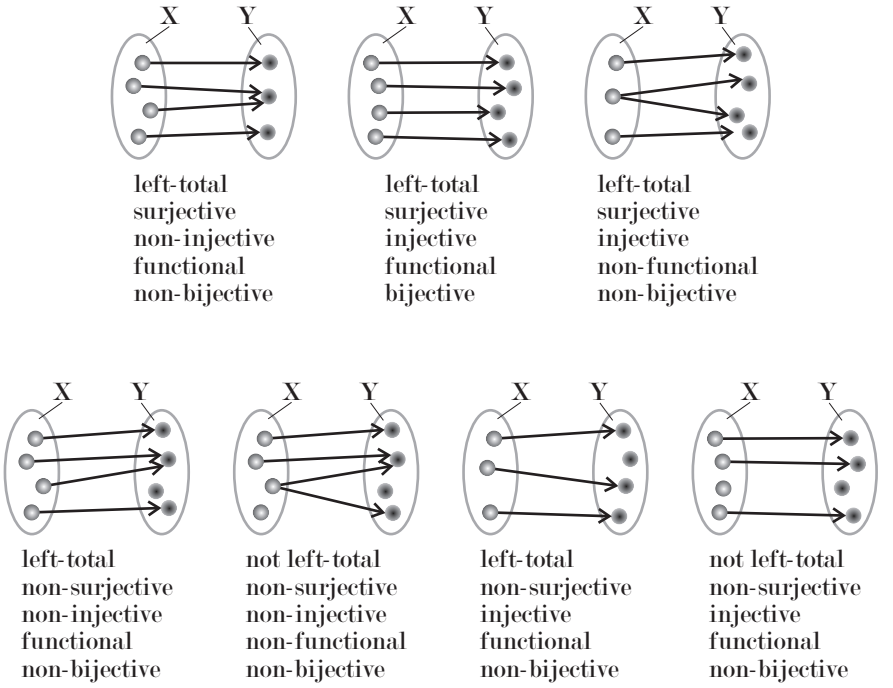


Fig. 34.2: Properties of a binary relation $R \in X \times Y$.

- left-total if $\forall x \in X \; \exists y \in Y : R(x, y)$.

[15] `http://en.wikipedia.org/wiki/Binary_relation` [accessed 2007-07-03]
[16] `http://en.wikipedia.org/wiki/Relation_%28mathematics%29` [accessed 2007-07-03]

- surjective[17] or right-total if $\forall y \in Y \; \exists x \in X : R(x, y)$.
- injective[18] if $\forall x, z \in X, y \in Y : R(x, y) \wedge R(z, y) \Rightarrow x = z$.
- functional if $\forall x \in X, y, z \in Y : R(x, y) \wedge R(x, z) \Rightarrow y = z$.
- bijective[19] if it is left-total, right-total and functional.
- transsititve[20] if $\forall x, y \in X, \; y, z \in Y \; : \; R(x, y) \wedge R(y, z) \Rightarrow R(x, z)$. [1119]

34.6.1 Order relations

Besides functions, which are discussed in the next section, there is another important group of relations – order relations[21].

Definition 109 (Partial Order). On the set X the binary relation R defines a (*non-strict*, *reflexive*) partial order if and only if it is

1. reflexive: $R(x, x) \; \forall \; x \in X$ (34.50)
2. antisymmetric: $R(x, y) \wedge R(y, x) \Rightarrow x = y \; \forall \; x, y \in X$ (34.51)
3. transitive: $R(x, y) \wedge R(y, z) \Rightarrow R(x, z) \; \forall \; x, y, z \in X$. (34.52)

The relation R thus plays the role of the $\leq$-operator, i. e. $R(x, y) \equiv x \leq y$.

The definition above can be compared with the $\leq$ operator. In some contexts, partial orders are used more in the sense of $<$. Such partial orders are called *strict*. The Pareto dominance relation (see Definition 16 on page 15) is an example for such a strict partial order.

Definition 110 (Strict Partial Order). A relation R defined on the set X is a *strict* (or *irreflexive*) partial order relation R if it is

1. irreflexive: $\nexists x \in X : R(x, x)$
2. asymmetric: $R(x, y) \Rightarrow \neg R(y, x) \; \forall x, y$
3. transitive: (see definition of reflexive partial order)

Definition 111 (Total Order). A total order[22] (or linear order, simple order) R on the set X is a partial order which is complete/total.

$$R(x, y) \vee R(y, x) \; \forall \; x, y \in X \tag{34.53}$$

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).

[17] http://en.wikipedia.org/wiki/Surjective [accessed 2007-07-03]
[18] http://en.wikipedia.org/wiki/Injective [accessed 2007-07-03]
[19] http://en.wikipedia.org/wiki/Bijective [accessed 2007-07-03]
[20] http://en.wikipedia.org/wiki/Transitive_relation [accessed 2007-07-03]
[21] http://en.wikipedia.org/wiki/Order_relation [accessed 2007-07-03]
[22] http://en.wikipedia.org/wiki/Total_order [accessed 2007-07-03]

34.6.2 Equivalence Relations

Another important class of relations are equivalence relations[23] [1120, 1121] which are often abbreviated with $\equiv$ or $\sim$, i. e. $x \equiv y$ and $x \sim y$ mean $R(x, y)$ for the equivalence relation R on X and $x, y \in X$.

Definition 112 (Equivalence Relation). On the set X the binary relation R defines an equivalence relation if and only if it is

1. reflexive: $R(x, x)\ \forall\ x \in X$ (34.54)
2. symmetric: $R(x, y) \Rightarrow R(x, y)\ \forall\ x, y \in X$ (34.55)
3. transitive: $R(x, y) \wedge R(y, z) \Rightarrow R(x, z)\ \forall\ x, y, z \in X$. (34.56)

Definition 113 (Equivalence Class). If an equivalence relation R is defined on a set X, the subset $A \subseteq X$ of X is an equivalence class[24] if and only if $\forall\ a, b \in A \Rightarrow R(a, b)\ (= a \sim b)$.

34.6.3 Functions

Definition 114 (Function). A function f is a binary relation with the property that for an element x of the domain X there is no more than one element y of 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.

$$\forall x \in X, y, z \in Y : f(x, y) \wedge f(x, z) \Rightarrow y = z$$

The set of inputs X of a function f is called its domain[25]. While the codomain Y is the set of the possible output values of f, the set of all actual outputs $\{f(x) : x \in X\}$ is called range. A function maps one element of Y to each element of X. The function $f = \frac{1}{x}$ has the domain $x = \mathbb{R} \setminus \{0\}$ instead of $\mathbb{R}$ since it is undefined at $x = 0$.

Monotonicity

Real functions are monotone, i. e. have the property of monotonicity[26], if they preserve a given order.

Definition 115 (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 34.57 holds.

$$\forall\ x < y \in X \Rightarrow f(x) \leq f(y) \tag{34.57}$$

[23] http://en.wikipedia.org/wiki/Equivalence_relation [accessed 2007-07-28]
[24] http://en.wikipedia.org/wiki/Equivalence_class [accessed 2007-07-28]
[25] http://en.wikipedia.org/wiki/Domain_%28mathematics%29 [accessed 2007-07-03]
[26] http://en.wikipedia.org/wiki/Monotonic_function [accessed 2007-08-08]

Definition 116 (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 34.58 holds.

$$\forall\, x < y \in X \Rightarrow f(x) \geq f(y) \tag{34.58}$$

35

Stochastic Theory

The stochastic[1] theory includes the probability[2] theory[3] which is is the mathematical study of phenomena characterized by randomness or uncertainty as well as statistics[4] dealing with the collection, analysis, interpretation, and presentation of data [1122, 1123, 1124, 1125].

35.1 Probability

Probability theory is used to determine the likeliness of the occurrence of an event under ideal mathematical conditions. [1126, 1127]

Definition 117 (Random Experiment). Random experiments can be repeated arbitrary often, their results cannot be predicted.

Definition 118 (Elementary Event). The possible outcomes of random situations are called elementary events or samples ω.

Definition 119 (Sample Space). The set of all possible outcomes (elementary events, samples) of a random situation is the sample space $\Omega = \{\omega_i : i \in 1..N\}$. When throwing dice[5], for example, Ω 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 120 (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.

Definition 121 (Certain Event). The certain event is the random event will occur in each repetition of a random situation, it is defined as $A = \Omega$.

[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]
[5] Throwing a dice is discussed as example for stochastic extensively in Section 35.5 on page 548.

Definition 122 (Impossible Event). The impossible event will never occur any repetition of a random situation, it is defined as $A = \emptyset$.

Definition 123 (Absolute Frequency). If repeating a random experiment, the number an event A occurred in this repetitions is its absolute frequency[6].

Definition 124 (Conflicting Events). Two conflicting events A_1 and A_2 can never occur together in a random situation. Therefore, $A_1 \cap A_2 = \emptyset$.

35.1.1 Probabily as defined by Bernoulli (1713)

Under the assumption that no previous knowledge exists it can also be assumed that all elementary events have the same probability. All elementary events of a sample space are equally probable if $P(\omega) = \frac{1}{N} \forall \omega \in \Omega$ holds (Laplace assumption, [1128]). Under this circumstances, the probability of an event A can be defined as:

$$P(A) = \frac{\textit{number for event in favour for } A}{\textit{number of possible events}} = \frac{n_A}{n} = h(A, n) \tag{35.1}$$

For some of the random experiments of this type, we can use combinatorics[7] in order to determine the number of possible outcomes. Therefore we introduce the factorial and the combinations as follows:

Definition 125 (Factorial). The factorial[8] $n!$ of $n \in \mathbb{N}$ is the product of n and all natural numbers smaller then n:

$$n! = \prod_{i=1}^{n} i \tag{35.2}$$

$$0! = 1 \tag{35.3}$$

See also Gamma Function in Section 35.9.1 on page 570.

Definition 126 (Combinations). The number of possible combinations[9] of $n \in \mathbb{N}$ elements out of a set Ω with $M = |\Omega| \geq n$ is

$$C(M, n) = \binom{M}{n} = \frac{M!}{n!(n-r)!} \tag{35.4}$$

$$C(M+1, n) = C(M, n) + C(M, n-1) = \tag{35.5}$$

$$\binom{M+1}{n} = \binom{M}{n} + \binom{M}{n-1} \tag{35.6}$$

[6] http://en.wikipedia.org/wiki/Frequency_%28statistics%29 [accessed 2007-07-03]
[7] http://en.wikipedia.org/wiki/Combinatorics [accessed 2007-07-03]
[8] http://en.wikipedia.org/wiki/Factorial [accessed 2007-07-03]
[9] http://en.wikipedia.org/wiki/Combinations_and_permutations [accessed 2007-07-03]

- Permutation with repetition: when order matters and an element $\omega \in \Omega$ can be chosen more than once, the number of different permutations is M^n.
- Permutation without repetition: when the order matters and an element $\omega \in \Omega$ can be chosen more than once, the number of different permutations is $\frac{M!}{(M-n)!}$.
- Combination without repetition: when order does not matter and each element $\omega \in \Omega$ can be chosen exactly once, the number of different combinations is $C(M,n) = \binom{M}{n}$.
- Combination with repetition: when order does not matter and each element $\omega \in \Omega$ can be chosen more than once, the number of different combinations is $\frac{(M+n-1)!}{n!(M-1)!} = \binom{M+n-1}{n} = \binom{M+n-1}{M-1}$

35.1.2 The Metrical Method of Van Mises (1919)

Definition 127 (Relative Frequency). The relative frequency of an event A is its absolute frequency normalized to the number of total events. The relative frequency has the following properties:

$$h(A,n) = \frac{n_A}{n} \tag{35.7}$$

$$0 \leq h(A,n) \leq 1 \tag{35.8}$$

$$h(\Omega,n) = 1 \tag{35.9}$$

$$A \cap B = \emptyset \Rightarrow h(A \cup B,n) = \frac{n_A + n_B}{n} = h(A,n) + h(B,n) \tag{35.10}$$

The (statistical) probability $P(A)$ as result of a process of metering is the limit of the relative frequency h of the event A. This is the limit of the quotient of the number of elementary events favouring A and the number of all possible elementary events for infinite many repetitions. [1129]

$$P(A) = \lim_{n\to\infty} h(A,n) = \lim_{n\to\infty} \frac{n_A}{n} \tag{35.11}$$

35.1.3 The Axioms of Kolmogorov

Definition 128 (σ-algebra). A subset S of the power set $\mathcal{P}(\Omega)$ is called σ-algebra[10], if it holds the following axioms:

$$\Omega \in S \tag{35.12}$$

$$\emptyset \in S \tag{35.13}$$

$$A \in S \Leftrightarrow \overline{A} \in S \tag{35.14}$$

$$A \in S \wedge B \in S \Rightarrow (A \cup B) \in S \tag{35.15}$$

[10] `http://en.wikipedia.org/wiki/Sigma-algebra` [accessed 2007-07-03]

From this axioms others can be deduced, for example:

$$\begin{aligned} A \in S \wedge B \in S &\Rightarrow \overline{A} \in S \wedge \overline{B} \in S && (35.16)\\ &\Rightarrow \overline{A} \cup \overline{B} \in S \\ &\Rightarrow \overline{\overline{A} \cup \overline{B}} \in S && (35.17)\\ &\Rightarrow A \cap B \in S \\ A \in S \wedge B \in S &\Rightarrow (A \cap B) \in S && (35.18) \end{aligned}$$

Definition 129 (Probability Space). A probability space (or random experiment) is defined by the triple (Ω, S, P) whereas

- Ω is a set of events,
- S is a σ-algebra defined on Ω, and
- $P(\omega)$ defines a probability measure[11] that determines an occurrence probability for each event $\omega \in \Omega$. (Kolmogorov axioms[12] [1130])

Definition 130 (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:

$$\forall A \in S \Rightarrow 0 \leq P(A) \leq 1 \quad (35.19)$$

$$P(\Omega) = 1 \quad (35.20)$$

$$\forall disjoint A_i \in S \Rightarrow P(A) = P\left(\bigcup_{\forall i} A_i\right) = \sum_{\forall i} P(A_i) \quad (35.21)$$

From this axioms can be deduced:

$$P(\emptyset) = 0 \quad (35.22)$$

$$P(A) = 1 - P(\overline{A}) \quad (35.23)$$

$$P(A \cap \overline{B}) = P(A) - P(A \cap B) \quad (35.24)$$

$$P(A \cup A) = P(A) + P(B) - P(A \cap B) \quad (35.25)$$

35.1.4 Conditional Probability

Definition 131 (Conditional Probability). Conditional probability[13] is the probability of some event A, given the occurrence of some other event B. Conditional probability is written $P(A|B)$, and is read "the probability of A, given B".

[11] http://en.wikipedia.org/wiki/Probability_measure [accessed 2007-07-03]
[12] http://en.wikipedia.org/wiki/Kolmogorov_axioms [accessed 2007-07-03]
[13] http://en.wikipedia.org/wiki/Conditional_probability [accessed 2007-07-03]

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \tag{35.26}$$

$$P(A \cap B) = P(A|B)P(B) \tag{35.27}$$

Definition 132 (Statistical Independence). Two events A and B are (statistical) independent if and only if $P(A \cap B) = P(A)P(B)$ holds. From this, we can deduce:

$$P(A \cap B) = P(A)P(B) \tag{35.28}$$

$$P(A|B) = P(A) \tag{35.29}$$

$$P(B|A) = P(B) \tag{35.30}$$

35.1.5 Random Variable

Definition 133 (Random Variable). The function X which relates the sample space Ω to the real numbers $\mathbb{R}$ is called random variable[14] in the probability space (Ω, S, P).

$$X : \Omega \to \mathbb{R} \tag{35.31}$$

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 by an 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 134 (Probability Space of a Random Variable). Is $X : \Omega \mapsto \mathbb{R}$ a random variable, then probability space of this random variable is defined as the triplet

$$(\Omega_X = \mathbb{R}, S_X, P_X) \tag{35.32}$$

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:

$$P_X(X \leq x) = P\left(\{\omega : \omega \in \Omega \wedge X(\omega) \leq x\}\right) \tag{35.33}$$

35.1.6 Cumulative Distribution Function

Definition 135 (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 $F_X : \mathbb{R} \to [0, 1]$ with

[14] http://en.wikipedia.org/wiki/Random_variable [accessed 2007-07-03]

$$F_X := \underbrace{P_X(X \leq x)}_{\textit{definition by rnd. var.}} \equiv \underbrace{P\left(\{\omega : \omega \in \Omega \wedge X(\omega) \leq x\}\right)}_{\textit{definition by prob. space}} \tag{35.34}$$

the (cumulative) distribution function[15] (CDF) of the random variable X.

A cumulative distribution function has the following properties:

- $F_X(X)$ is normalized:

$$\underbrace{\lim_{x \to -\infty} F_X(x) = 0}_{\textit{impossible event}}, \quad \underbrace{\lim_{x \to +\infty} F_X(x) = 1}_{\textit{certain event}} \tag{35.35}$$

- $F_X(X)$ is monotonously[16] growing:

$$F_X(x_1) \leq F_X(x_2) \; \forall x_1 \leq x_2 \tag{35.36}$$

- $F_X(X)$ is (right-sided) continuous[17]:

$$\lim_{h \to 0} F_X(x + h) = F_X(x) \tag{35.37}$$

- 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:

$$P(x_0 \leq X \leq x_1) = F_X(x_1) - F_X(x_0) \tag{35.38}$$

- The probability that the random variable X takes on the value of a single random number x:

$$P(X = x) = F_X(x) - \lim_{h \to 0} F_X(x - h) \tag{35.39}$$

We further distinguish between discrete[18] and continuous[19] random variables.

Definition 136 (Discrete Random Variable). A random variable X (and its probability measure P_X respectively) is called discrete if it takes on at most countable infinite many values and the cumulative distribution function $F_X(X)$ therefore has the shape of a stairway.

Definition 137. A random variable X (and its probability measure P_X respectively) is called continuous if it takes on uncountable infinite many values and the cumulative distribution function $F_X(X)$ is also continuous.

[15] `http://en.wikipedia.org/wiki/Cumulative_distribution_function` [accessed 2007-07-03]

[16] `http://en.wikipedia.org/wiki/Monotonicity` [accessed 2007-07-03]

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

[18] `http://en.wikipedia.org/wiki/Discrete_random_variable` [accessed 2007-07-03]

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

35.1.7 Probability Mass Function

The probability mass function[20] (PMF) exists for discrete distributions only. It assigns a probability to each value the random variable X can take on.

Definition 138 (Probability Mass Function). If a random variable X takes on only discrete values, its probability mass function f_X is defined as

$$f_X : \mathbb{Z} \to [0,1] \ : \ f_X(x) := P_X(X = x) \tag{35.40}$$

Therefore, we can specify the relation between the PMF and its according (discrete) CDF as done in Equation 35.41 and Equation 35.41. We can further define the probability of an event A in Equation 35.43.

$$P(X \leq x) = F_X(x) = \sum_{i=-\infty}^{x} f_X(x) \tag{35.41}$$

$$P(X = x) = f_X(x) = F_X(x) - F_X(x-1) \tag{35.42}$$

$$P_X(A) = \sum_{\forall x \in A} f_X(x) \tag{35.43}$$

35.1.8 Probability Density Function

The probability density function[21] (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 has the probability 0.

Definition 139 (Probability Density Function). If a random variable X is continuous, its probability density function f_X is defined as

$$f_X : \mathbb{R} \to [0,\infty) \ : \ F_X(x) = \int_{-\infty}^{+\infty} f_X(\xi) d\xi \ \forall x \in \mathbb{R} \tag{35.44}$$

35.2 Properties of Distributions and Statistics

Each random variable X obeying a probability distribution may or may not have certain properties such as a maximum or a minimum value, a mean or value which will most often be taken on by X. If the distribution of X is known, these values can most often be computed directly from its parameters.

[20] `http://en.wikipedia.org/wiki/Probability_mass_function` [accessed 2007-07-03]

[21] `http://en.wikipedia.org/wiki/Probability_density_function` [accessed 2007-07-03]

On the other hand, it is possible that one only knows some values $a \in A$ which X took on in the past. From this sample A, we can approximate the properties of the underlying (most often unknown) distribution of X using statistical methods. Statistics[22] is the mathematical science of collecting, analyzing, interpreting, explaining, and presenting of data.

In the following we will elaborate on the properties random variable $X \in \mathbb{R}$ both, from the standpoint 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 $n = |A|$ and the elements $a_i = A[i] \forall i \in [0, n-1]$.

35.2.1 Count, Min, Max and Range

Definition 140 (Count). $n = |A|$ is called the item count.

It does only exist in statistics and for samples, because random variables represent experiments which can infinitely be repeated and thus always produce infinitely many values. This should not be mixed up with the possible count of different values the random variable may take on which may limited. A however can contain the same value b multiple times. If throwing a dice seven times, one may throw $A = [1, 4, 3, 3, 2, 6, 1]$, for example[23].

Definition 141 (Minimum). There exists no smaller element α in A than the minimum (or the minima if the minimum element is included multiple times) $\check{a} \equiv min(A)$. This definition is identical with the definition of the global minimum, Definition 10 on page 9. If the random variable X has a smallest value $\check{x}$ it can take on, its minimum equals this value (Equation 35.47), otherwise its minimum is infinitely for in the negative (Equation 35.47). (see also Definition 7 on page 9 and Definition 10 on page 9)

$$min(A) \equiv \check{a} \in A \ : \ \forall b \in A \wedge \alpha \neq \check{a} \Rightarrow \check{a} < \alpha \tag{35.45}$$

$$\exists \ \check{x} = min(X) \Leftrightarrow f_X(\check{x}) > 0 \wedge f_X(y) = 0 \ \forall y < \check{x} \tag{35.46}$$

$$\nexists \ \check{x} \Leftrightarrow min(X) = -\infty \tag{35.47}$$

Definition 142 (Maximum). There exists no bigger element α in A than the maximum (or the maxima if the maximum element is included multiple times) $\hat{a} \equiv min(A)$. This definition is identical with the definition of the global maximum, Definition 9 on page 9. If the random variable X has a largest value $\hat{x}$ it can take on, its maximum equals this value (Equation 35.50), otherwise its minimum is infinitely large (Equation 35.50). (see also Definition 6 on page 8 and Definition 9 on page 9)

[22] `http://en.wikipedia.org/wiki/Statistics` [accessed 2007-07-03]

[23] Throwing a dice is discussed as example for stochastic extensively in Section 35.5 on page 548.

$$max(A) \equiv \hat{a} \in A : \forall \alpha \in A \wedge \alpha \neq \hat{a} \Rightarrow \hat{a} > \alpha \tag{35.48}$$

$$\exists\, \hat{x} = max(X) \Leftrightarrow f_X(\hat{x}) > 0 \wedge f_X(y) = 0\ \forall y > \hat{x} \tag{35.49}$$

$$\nexists\, \hat{x} \Leftrightarrow max(X) = \infty \tag{35.50}$$

Definition 143 (Range).
The range $range(A)$ of the data set A is the difference of the maximum and the minimum of a data set and therefore represents the width of the span covered with data. If a random variable X is limited in both directions, it has a finite range, otherwise its range is infinite.

$$range(A) = \hat{a} - \check{a} = max(A) - min(A) \tag{35.51}$$

$$range(X) = \hat{x} - \check{x} = max(X) - min(X) \tag{35.52}$$

35.2.2 Expected Value and Arithmetic Mean

Definition 144 (Expected Value). The expected value[24] of a random variable X the sum of the probability of each possible outcome of the experiment multiplied by the outcome value. It is abbreviated by EX or μ and is often also called the mean or arithmetic mean value (see Definition 146). For discrete distributions it can be computed using Equation 35.53 and for continuous ones Equation 35.54 holds.

$$EX = \sum_{i=-\infty}^{\infty} i f_X(i) \tag{35.53}$$

$$EX = \int_{-\infty}^{\infty} x f_X(x) dx \tag{35.54}$$

For the expected value EX of a random variable X, the following statements are valid:

$$Y = a + X \Rightarrow EY = a + EX \tag{35.55}$$

$$Z = bX \Rightarrow EZ = bEX \tag{35.56}$$

Definition 145 (Sum). The $sum(A)$ represents the sum of all elements in A. This value does, of course, not exist for random variables.

$$sum(A) = \sum_{i=0}^{n-1} a_i \tag{35.57}$$

Definition 146 (Arithmetic Mean). The arithmetic mean[25] $\overline{a}$ is the sum of all elements in A divided by their count. It corresponds to the expected

[24] http://en.wikipedia.org/wiki/Expected_value [accessed 2007-07-03]
[25] http://en.wikipedia.org/wiki/Arithmetic_mean [accessed 2007-07-03]

value $\overline{a} \equiv EX$. The arithmetic mean of a sample data set approximates the expected value of the random variable that produced the sample.

$$\overline{a} = \frac{sum(A)}{n} = \frac{1}{n}\sum_{i=0}^{n-1} a_i \tag{35.58}$$

35.2.3 Variance and Standard Deviation

Definition 147 (Variance). The variance[26] $D^2X \equiv var(X)$ is a measure of statistical dispersion. It illustrates how close the elements $a \in A$ are to their arithmetical mean $\overline{a}$. The variance is defined for both, random variables (where it is often abbreviated with σ^2 and samples (where it is often abbreviated with s^2).

$$var(X) = D^2X = var(X) = E((X - EX)^2) = EX^2 - (EX)^2 \tag{35.59}$$

The variance of a discrete random variable X can be computed using Equation 35.60 and the one of a continuous distribution will obey Equation 35.61.

$$D^2X = \sum_{-\infty}^{\infty} f_X(i)(i - EX)^2 \tag{35.60}$$

$$\begin{aligned} D^2X &= \int_{-\infty}^{\infty} x^2 f_X(x)dx - \left[\int_{-\infty}^{\infty} x f_X(x)dx\right]^2 \\ &= \int_{-\infty}^{\infty} x^2 f_X(x)dx - (EX)^2 \end{aligned} \tag{35.61}$$

For the variance D^2X of a random variable X, the following statements are valid:

$$Y = a + X \Rightarrow D^2Y = D^2X \tag{35.62}$$

$$Z = bX \Rightarrow D^2Z = b^2D^2X \tag{35.63}$$

Definition 148 (Sum of Squares). The $sumSqr(A)$ represents the sum of the squares all elements in A. This property does not exist for random variables.

$$sumSqr(A) = \sum_{i=1}^{n-1} a_i^2 \tag{35.64}$$

We define the (unbiased) estimator[27] s^2 of the variance of the random variable which produced the sample values $a \in A$ according to Equation 35.65. The variance is zero for all samples with $n \leq 1$.

[26] `http://en.wikipedia.org/wiki/Variance` [accessed 2007-07-03]
[27] see Definition 167 on page 551

$$\begin{aligned} s^2 &= \frac{1}{n-1}\sum_{i=0}^{n-1}(a_i - \overline{a})^2 \\ &= \frac{1}{n-1}\left(sumSqr(A) - \frac{(sum(A))^2}{n}\right) \end{aligned} \tag{35.65}$$

Definition 149 (Standard Deviation). The standard deviation[28] DX is the square root of the variance. It is also often referred to as σ (random variables) or s (sample data sets).

$$DX = \sqrt{D^2X} \tag{35.66}$$

The standard deviation is zero for all samples with $n \leq 1$.

Definition 150 (Coefficient of Variation). The coefficient of variation[29] c_v is the ratio of the standard deviation by the arithmetic mean. For sample sets A, c_v can be computed as specified in Equation 35.68.

$$c_v = \frac{DX}{EX} \equiv \frac{\sigma}{\mu} \tag{35.67}$$

$$c_v = \frac{n}{sum(A)}\sqrt{\frac{sumSqr(A) - \frac{(sum(A))^2}{n}}{n-1}} \tag{35.68}$$

35.2.4 Moments

Definition 151 (Statistical Moment). The k^{th} moment is the expected value raised to the k^{th} power.

$$\mu'_k = E\left[x^k\right] \tag{35.69}$$

Definition 152 (Central Moment). The k^{th} moment about the mean (or central moment)[30] is the expected value of the difference between elements and their expected value raised to the k^{th} power.

$$\mu_k = E\left[(X - EX)^k\right] \tag{35.70}$$

Definition 153 (Standardized Moment). The k^{th} standardized moment is written the quotient of the k^{th} central moment by the standard deviation raised to the k^{th} power.

$$\frac{\mu_k}{\sigma^k} \tag{35.71}$$

[28] `http://en.wikipedia.org/wiki/Standard_deviation` [accessed 2007-07-03]
[29] `http://en.wikipedia.org/wiki/Coefficient_of_variation` [accessed 2007-07-03]
[30] `http://en.wikipedia.org/wiki/Moment_about_the_mean` [accessed 2007-07-03]

35.2.5 Skewness and Kurtosis

Definition 154 (Skewness). The skewness[31] γ_1 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.

$$\gamma_1 = \frac{\mu_3}{\sigma^3} \tag{35.72}$$

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 with at least three elements.

$$G_1 = \frac{n}{(n-1)(n-2)} \sum_{i=0}^{n-1} \left(\frac{a_i - \overline{a}}{s} \right)^3 \tag{35.73}$$

Definition 155 (Kurtosis). The excess kurtosis 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 35.4.2) has a zero kurtosis.

$$\gamma_2 = \frac{\mu_4}{\sigma^3} - 3 \tag{35.74}$$

For sample data A represents only a sample of a greater dataset, 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.

$$G_2 = \left\{ \frac{n(n+1)}{(n-1)(n-2)(n-3)} \sum_{i=0}^{n-1} \left(\frac{a_i - \overline{a}}{s} \right)^4 \right\} - \frac{3(n-1)^2}{(n-2)(n-3)} \tag{35.75}$$

35.2.6 Median, Quantiles, and Mode

Definition 156 (Median). The median $m = med(X)$ 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 then $med(X)$ is equal to the probability of drawing an element larger than m.

$$P(X \leq m) \geq \frac{1}{2} \wedge P(X \geq m) \geq \frac{1}{2} \wedge P(X \leq m) \leq P(X \geq m) \tag{35.76}$$

[31] `http://en.wikipedia.org/wiki/Skewness` [accessed 2007-07-03]

We can solve Equation 35.77 for continuous distributions and Equation 35.78 for discrete distributions in order to obtain the median m.

$$\frac{1}{2} = \int_{-\infty}^{m} f_X(x)dx \tag{35.77}$$

$$\sum_{i=-\infty}^{m-1} f_X(x) \leq \frac{1}{2} \leq \sum_{i=-\infty}^{m} f_X(x) \tag{35.78}$$

$$\tag{35.79}$$

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 representing the sample better. The median of a sample can be computed as:

$$A_s \equiv sort(A) \tag{35.80}$$

$$med(A) = \begin{cases} A_s\left[\lfloor\frac{n}{2}\rfloor\right] & if\ n\ is\ odd \\ \frac{1}{2}\left(A_s\left[\frac{n}{2}\right] + A_s\left[\frac{n}{2} - 1\right]\right) & otherwise \end{cases} \tag{35.81}$$

Definition 157 (Quantile). Quantiles[32] are points taken at regular intervals from a sorted dataset (or a cumulative distribution function). q-quantile divide a distribution/sample A into q parts A_i with equal probability. They can be regarded as the generalized median, or, in other words, the median is the 2-quantile.

$$\forall\, a \in \mathbb{R}, i \in [0, q-1] \Rightarrow \frac{1}{q} \leq P(a \in A_i) \tag{35.82}$$

A sorted data sample is divided into q subsets of equal length by the q-quantiles. The cumulative distribution function of a random variable A is divided by the q-quantiles into q subsets of equal area. The quantiles are the boundaries between the subsets. Therefore, the k^{th} 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 less than or equal to ζ is at least $\frac{k}{q}$. There exist $q-1$ q-quantiles (k spans from 1 to $q-1$).

The k^{th} q-quantile $quantile_q^k(A)$ of a dataset A can be computed as:

$$A_s \equiv sort(A) \tag{35.83}$$

$$quantile_q^k(A)) = A_s\left[\lfloor\frac{k * n}{q}\rfloor\right] \tag{35.84}$$

[32] http://en.wikipedia.org/wiki/Quantiles [accessed 2007-07-03]

Table 35.1: Special Quantiles

q	name
100	percentiles
10	deciles
9	noniles
5	quintiles
4	quartiles
2	median

Definition 158 (Interquartile Range). The interquartile range[33] is the range between the first and the third quartile and defined as $quantile_4^3(X) - quantile_4^1(X)$.

Definition 159 (Mode). The mode[34] 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 [1131, 1132] you can find further information of the relation between the mode, the mean and the skewness.

35.2.7 Entropy

Definition 160 (Entropy). The information entropy[35] $H(X)$, first defined by Shannon [1133], is often referred as measure of uncertainty.

The entropy of a discrete distribution with a finite number of possible values is given by Equation 35.85 whereas Equation 35.86 defines the entropy of a continuous distribution function.

$$H(X) = \sum_{i=1}^{n} f_X(x_i) \log_2 \left(\frac{1}{f_X(x_i} \right) = -\sum_{i=1}^{n} f_X(x_1) \log_2 f_X(x_i) \quad (35.85)$$

$$H(X) = -\int_{-\infty}^{\infty} f_X(x) \ln f_X(x) dx \quad (35.86)$$

[33] http://en.wikipedia.org/wiki/Inter-quartile_range [accessed 2007-07-03]
[34] http://en.wikipedia.org/wiki/Mode_%28statistics%29 [accessed 2007-07-03]
[35] http://en.wikipedia.org/wiki/Information_entropy [accessed 2007-07-03]

35.2.8 The Law of Large Numbers

The law of large numbers combines statistic and probability by stating 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 127) converges to its probability p if n becomes larger.

The weak law of large numbers states that for each positive real number $\varepsilon > 0, \varepsilon \in \mathbb{R}$ the mean $\overline{X}$ of an infinite sequence of independent random numbers X_i with all the same expected value EX and variance converges to EX,

$$\overline{X} = \frac{X_1 + \ldots + X_n}{n}, EX_1 = \ldots = EX_n = EX, D^2X_1 = \ldots = D^2X_n$$
$$\Rightarrow \lim_{n\to\infty} P(|\overline{X} - \mu| < \varepsilon) = 1 \tag{35.87}$$

The mean $\overline{X}$ of a sequence $X_1, \ldots, X_n$ of equally distributed and pairwise independent random variables converges to their expected value $EX = EX_1 = \ldots = EX_n$ for infinite large n according to the strong law of large numbers.

$$P(\lim_{n\to\infty} \overline{X} = \mu) = 1 \tag{35.88}$$

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.

35.3 Some Discrete Distributions

In this section we will introduce some common discrete distributions. Continuous 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 [2].

35.3.1 Discrete Uniform Distribution

The uniform distribution exists in a discrete[36] 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 35.3.1.

All possible outcomes $\omega \in \Omega$ of a 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

[36] http://en.wikipedia.org/wiki/Uniform_distribution_%28discrete%29 [accessed 2007-07-03]

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 an item out of n possible ones are other instances of the uniform distribution. Table 35.2 contains the characteristics of the discrete uniform distribution. In Figure 35.1 you can find some example uniform probability density functions and in Figure 35.2 the according cumulative distribution functions.

Table 35.2: Parameters of the discrete uniform distribution.

parameter	definition	
parameters	$a, b \in \mathbb{Z}, a \geq b$	(35.89)
$\|\Omega\|$	$\|\Omega\| = r = range = b - a + 1$	(35.90)
PMF	$P(X = x) = f_X(x) = \begin{cases} \frac{1}{r} & \forall a \leq x \leq b, x \in \mathbb{N} \\ 0 & otherwise \end{cases}$	(35.91)
CDF	$P(X \leq x) = F_X(x) = \lfloor \frac{x-a+1}{r} \rfloor$	(35.92)
mean	$EX = \frac{a+b}{2}$	(35.93)
median	$med = \frac{a+b}{2}$	(35.94)
mode	$mod = \emptyset$	(35.95)
variance	$D^2X = \frac{r^2-1}{12}$	(35.96)
skewness	$\gamma_1 = 0$	(35.97)
kurtosis	$\gamma_2 = -\frac{6(r^2+1)}{5(r^2-1)}$	(35.98)
entropy	$H(X) = \ln r$	(35.99)
mgf	$M_X(t) = \frac{e^{at} - e^{(b+1)t}}{r(1-e^t)}$	(35.100)
char. func.	$\varphi_X(t) = \frac{e^{iat} - e^{i(b+1)t}}{r(1-e^{it})}$	(35.101)

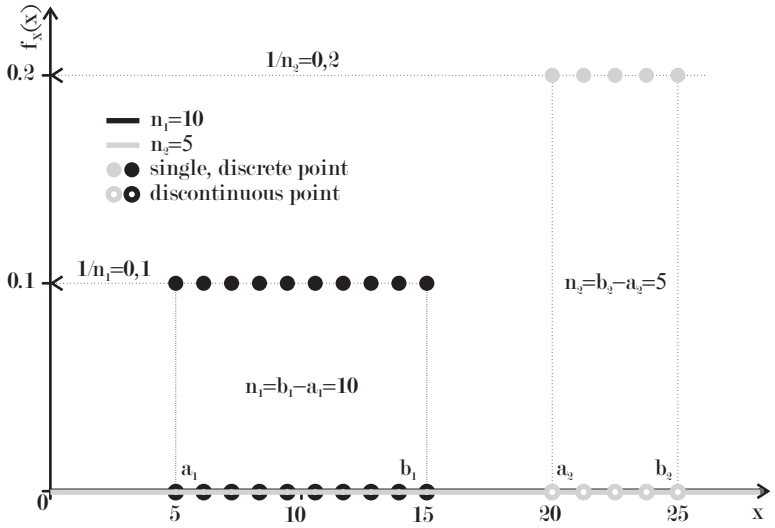


Fig. 35.1: The PMFs of some discrete uniform distributions

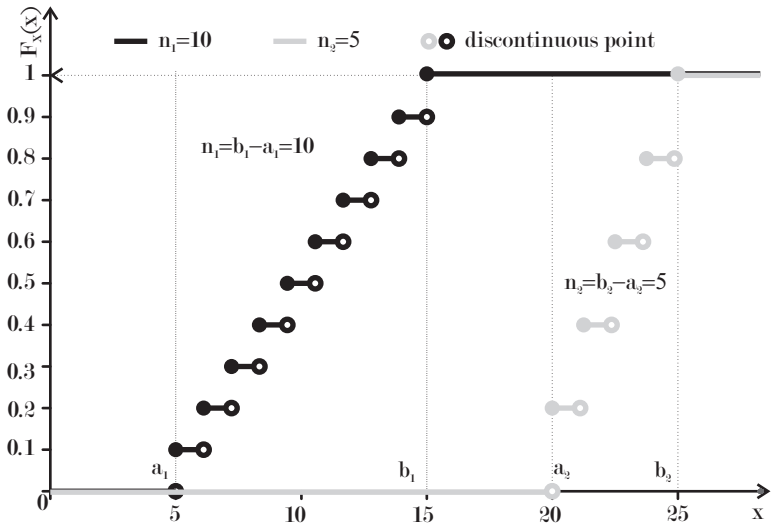


Fig. 35.2: The CDFs of some discrete uniform distributions

35.3.2 Poisson Distribution π_λ

The Poisson distribution[37] π_λ [1134] applies to 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 its duration and not of its position (prehistory). Events do not have any aftermath and thus, there is no mutual influence of non-overlapping time intervals (homogenity). Events have no duration and in infinite short time intervals no event occurs. The features of the Poisson distribution are listed in Table 35.3[38] and examples for its PDF and CDF are illustrated in Figure 35.3 and Figure 35.4.

Table 35.3: Parameters of the Poisson distribution.

parameter	definition	
parameters	$\lambda = \mu t > 0$	(35.102)
PMF	$P(X = x) = f_X(x) = \frac{(\mu t)^x}{x!} e^{-\mu t} = \frac{\lambda^x}{x!} e^{-\lambda}$	(35.103)
CDF	$P(X \leq x) = F_X(x) = \frac{\Gamma(k+1,\lambda)}{k!} = \sum_{i=0}^{x} \frac{e^{-\lambda}\lambda^i}{i!}$	(35.104)
mean	$EX = \mu t = \lambda$	(35.105)
median	$med \approx \lfloor \lambda + \frac{1}{3} - \frac{1}{5\lambda} \rfloor$	(35.106)
mode	$mod = \lfloor \lambda \rfloor$	(35.107)
variance	$D^2X = \mu t = \lambda$	(35.108)
skewness	$\gamma_1 = \lambda^{-\frac{1}{2}}$	(35.109)
kurtosis	$\gamma_2 = \frac{1}{\lambda}$	(35.110)
entropy	$H(X) = \lambda(1 - \ln \lambda) + e^{-\lambda} \sum_{k=0}^{\infty} \frac{\lambda^k \ln(k!)}{k!}$	(35.111)
mgf	$M_X(t) = e^{\lambda(e^t - 1)}$	(35.112)
char. func.	$\varphi_X(t) = e^{\lambda(e^{it} - 1)}$	(35.113)

Poisson Process

The Poisson process[39] [1135] is a process that obeys the Poisson distribution – just like the example of the telephone switchboard mentioned before. Instead of the directly calculating with the λ-values, we use μ, the intensity of the process, which normally describes a frequency, for example $\frac{1}{min}$, and t, the time (for example 1 min) in practical scenarios. Both, the expected value as well as the variance of the Poisson process are $\lambda = \mu t$. In Equation 35.114, the probability that k events occur in a Poisson process in a time interval of the length t is denoted.

[37] `http://en.wikipedia.org/wiki/Poisson_distribution` [accessed 2007-07-03]

[38] More information on the gamma function (Γ) used in Equation 35.104 can be found in Section 35.9.1 on page 570.

[39] `http://en.wikipedia.org/wiki/Poisson_process` [accessed 2007-07-03]

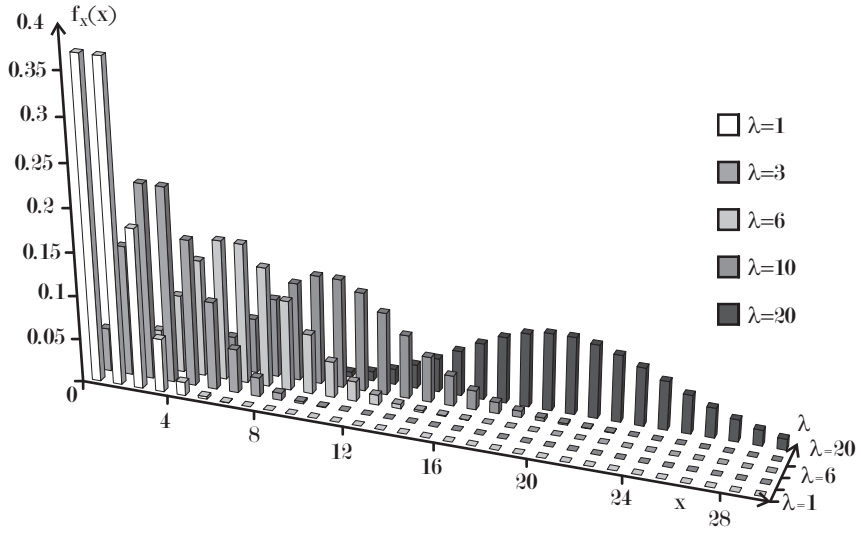


Fig. 35.3: The PMFs of some Poisson distributions

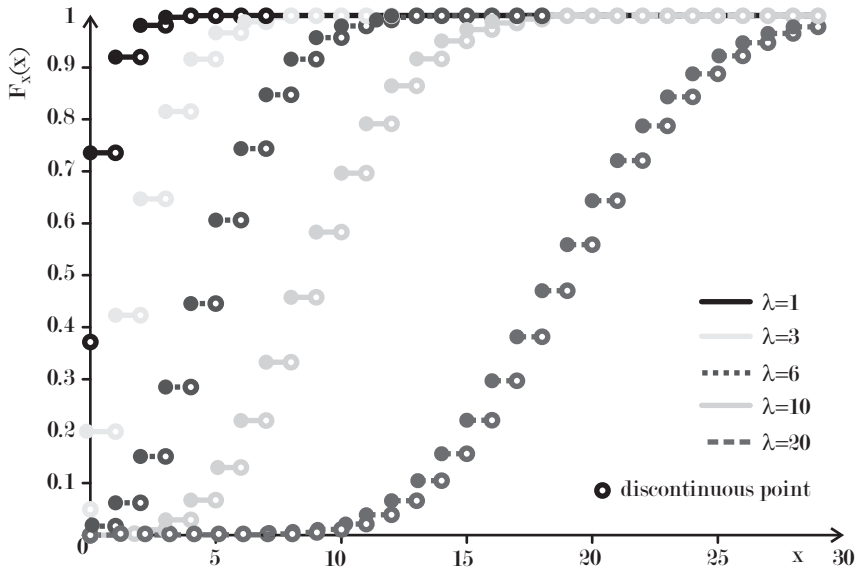


Fig. 35.4: The CDFs of some Poisson distributions

$$P(X_t = k) = \frac{(\mu t)^k}{k!} e^{-\mu t} = \frac{\lambda^k}{k!} e^{-\lambda} \tag{35.114}$$

The probability that in a time interval $[t, t + \Delta t]$

- no events occur is $1 - \lambda \Delta t + o(\Delta t)$.
- exactly one event occurs is $\lambda \Delta t + o(\Delta t)$.
- multiple events occur $o(\Delta t)$.

We abuse the small-o notation[40] here a little bit by simply saying the $o(\Delta t)$ 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:

$$\lim_{t \to 0} P(X_t > 1) = 0 \tag{35.115}$$

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 (Section 35.4.3). The expected value of the count 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 Poisson process is exponentially distributed with the same lambda value as the Poisson process, as illustrated in Equation 35.116.

$$X_i \sim \pi_\lambda \Leftrightarrow (t(X_{i+1}) - t(X_i)) \sim exp(\lambda) \; \forall i \in \mathbb{N} \tag{35.116}$$

35.3.3 Binomial Distribution $B(n, p)$

The binomial distribution[41] $B(n, p)$ is the probability distribution of successes of n independent experiments with the success probability p. Such an experiment is called Bernoulli experiment or Bernoulli trial. For $n = 1$, the binomial distribution is a Bernoulli distribution[42].

[40] See Section 37.1.3 on page 589 and Definition 206 on page 590 for a detailed elaboration on the small-o notation. The statement that $f \in o(\xi) \Rightarrow |f(x)| \ll |\xi(x)|$ is generally only valid for $x \to \infty$, which is not the case here.

[41] `http://en.wikipedia.org/wiki/Binomial_distribution` [accessed 2007-10-01]

[42] `http://en.wikipedia.org/wiki/Bernoulli_distribution` [accessed 2007-10-01]

Table 35.4[43] points out some of the properties of the binomial distribution. Some examples for PMFs and CDFs of different binomial distributions are given in Figure 35.5 and Figure 35.6.

Table 35.4: Parameters of the Binomial distribution.

parameter	definition	
parameters	$n \in \mathbb{N}_0,\ 0 \leq p \leq 1,\ p \in \mathbb{R}$	(35.117)
PMF	$P(X = x) = f_X(x) = \binom{n}{x} p^x (1-p)^{n-x}$	(35.118)
CDF	$P(X \leq x) = F_X(x) = \sum_{i=0}^{\lfloor x \rfloor} f_X(x) = I_{1-p}\left(n - \lfloor x \rfloor, 1 + \lfloor x \rfloor\right)$	(35.119)
mean	$EX = np$	(35.120)
median	med is one of $\{\lfloor np \rfloor - 1, \lfloor np \rfloor, \lfloor np \rfloor + 1\}$	(35.121)
mode	$mod = \lfloor (n+1)p \rfloor$	(35.122)
variance	$D^2X = (np)(1-p)$	(35.123)
skewness	$\gamma_1 = \frac{1-2p}{\sqrt{np(1-p)}}$	(35.124)
kurtosis	$\gamma_2 = \frac{1-6p(1-p)}{np(1-p)}$	(35.125)
entropy	$H(X) = \frac{1}{2}\ln(2\pi nep(1-p)) + O\left(\frac{1}{n}\right)$	(35.126)
mgf	$M_X(t) = (1-p+pe^t)^n$	(35.127)
char. func.	$\varphi_X(t) = (1-p+pe^{it})^n$	(35.128)

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 35.4.2) $N(np, np(1-p))$. 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. $F_{X,bin}(x) \approx F_{X,normal}(x + 0.5)$.

[43] I_{1-p} in Equation 35.119 denotes the regularized incomplete beta function.

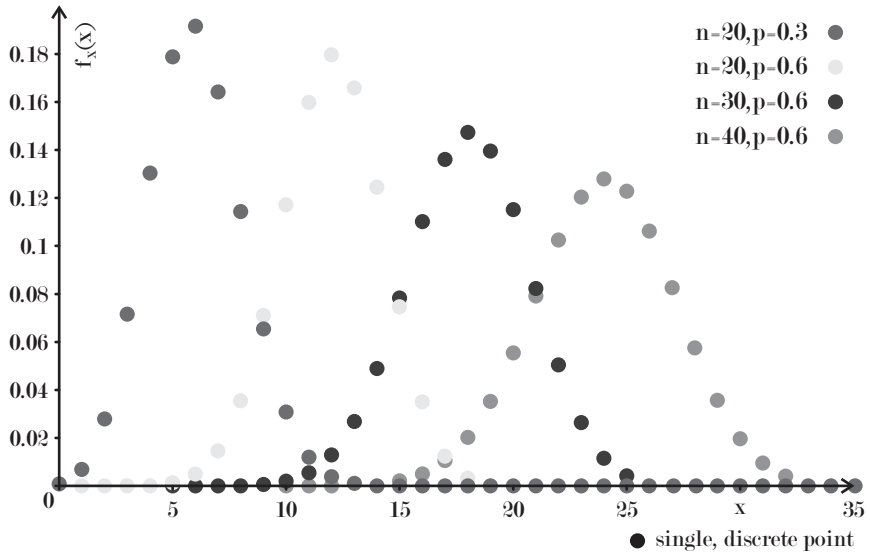


Fig. 35.5: The PMFs of some binomial distributions

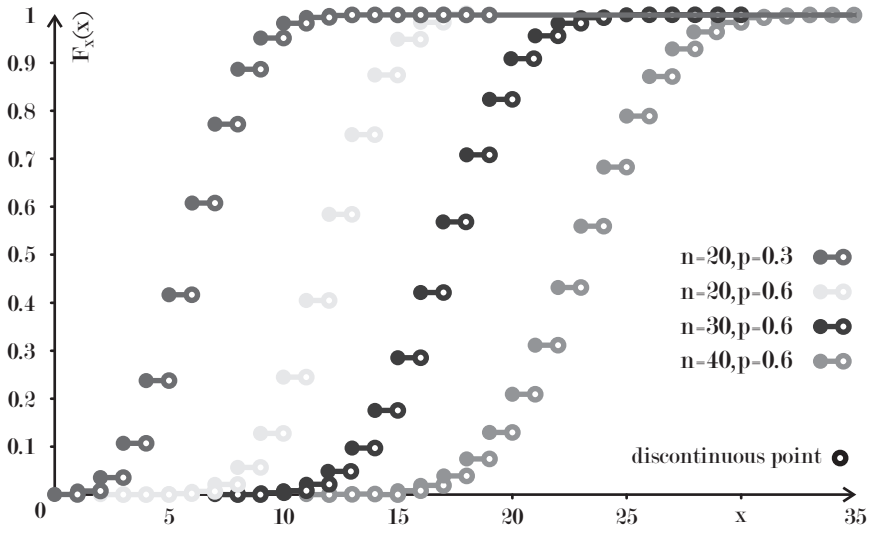


Fig. 35.6: The CDFs of some binomial distributions

35.4 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.

35.4.1 Continuous Uniform Distribution

After discussing the discrete uniform distribution in Section 35.3.1, we now elaborate on its continuous form[44].

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 35.5. Examples of its probability density function is illustrated in Figure 35.7 whereas the according cumulative density functions are outlined Figure 35.8.

Table 35.5: Parameters of the continuous uniform distribution.

parameter	definition	
parameters	$a, b \in \mathbb{R}, a \geq b$	(35.129)
PDF	$f_X(x) = \begin{cases} \frac{1}{b-a} & \forall x \in [a, b] \\ 0 & else \end{cases}$	(35.130)
CDF	$P(X \leq x) = F_X(x) = \begin{cases} 0 & \forall x < a \\ \frac{x-a}{b-a} & \forall x \in [a, b] \\ 1 & \forall x > b \end{cases}$	(35.131)
mean	$EX = \frac{1}{2}(a + b)$	(35.132)
median	$med = \frac{1}{2}(a + b)$	(35.133)
mode	$mod = \emptyset$	(35.134)
variance	$D^2X = \frac{1}{12}(b - a)^2$	(35.135)
skewness	$\gamma_1 = 0$	(35.136)
kurtosis	$\gamma_2 = -\frac{6}{5}$	(35.137)
entropy	$H(X) = \ln(b - a)$	(35.138)
mgf	$M_X(t) = \frac{e^{tb} - e^{ta}}{t(b-a)}$	(35.139)
char. func.	$\varphi_X(t) = \frac{e^{itb} - e^{ita}}{it(b-a)}$	(35.140)

[44] http://en.wikipedia.org/wiki/Uniform_distribution_%28continuous%29 [accessed 2007-07-03]

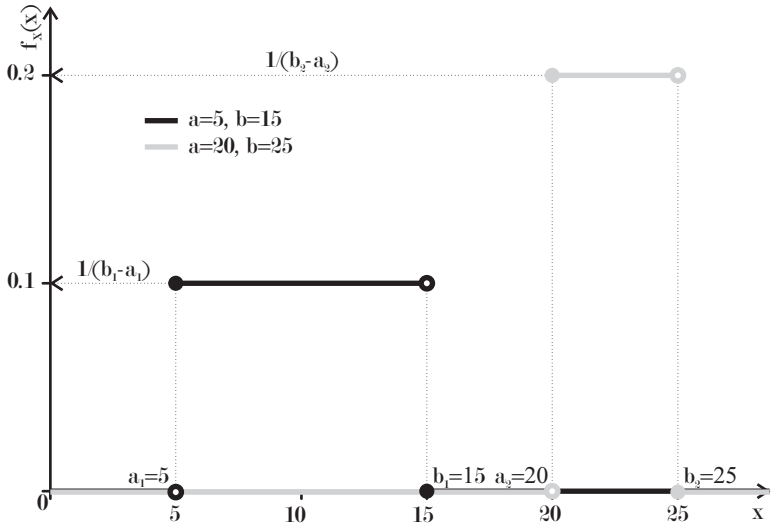


Fig. 35.7: The PDFs of some continuous uniform distributions

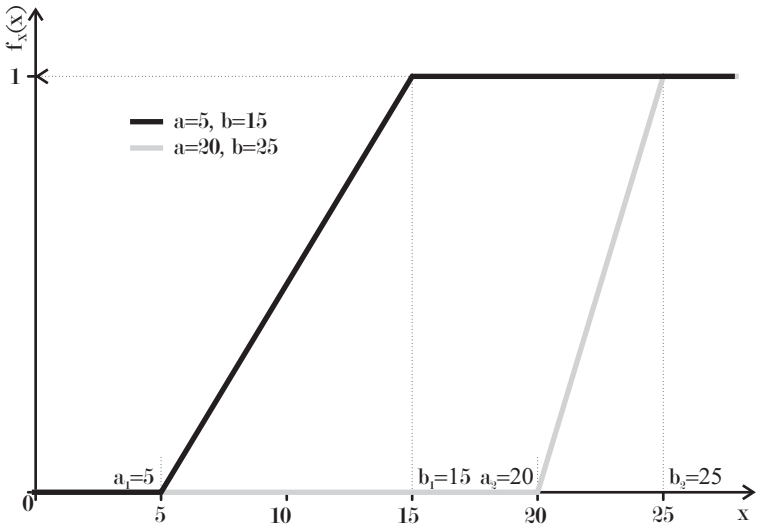


Fig. 35.8: The CDFs of some continuous uniform distributions

35.4.2 Normal Distribution $N(\mu, \sigma^2)$

Many phenomena in nature like the size of chicken eggs, noise, errors in measurement, and such and such can be approximated by the normally distributed[45] $N(\mu, \sigma^2)$ [1136]. Its probability density function, shown for some example values in Figure 35.9, is symmetric to the expected value μ and becomes flatter the higher the standard deviation σ gets. The cumulative density function is outline for the same example values in Figure 35.10. Other characteristics of the normal distribution can be found in Table 35.6.

Table 35.6: Parameters of the normal distribution.

parameter	definition	
parameters	$\mu \in \mathbb{R}, \sigma \in \mathbb{R}^+$	(35.141)
PDF	$f_X(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$	(35.142)
CDF	$P(X \leq x) = F_X(x) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{(z-\mu)^2}{2\sigma^2}} dz$	(35.143)
mean	$EX = \mu$	(35.144)
median	$med = \mu$	(35.145)
mode	$mod = \mu$	(35.146)
variance	$D^2X = \sigma^2$	(35.147)
skewness	$\gamma_1 = 0$	(35.148)
kurtosis	$\gamma_2 = 0$	(35.149)
entropy	$H(X) = \ln(\sigma\sqrt{2\pi e})$	(35.150)
mgf	$M_X(t) = e^{\mu t + \frac{\sigma^2 t^2}{2}}$	(35.151)
char. func.	$\varphi_X(t) = e^{\mu i t + \frac{\sigma^2 t^2}{2}}$	(35.152)

Definition 161 (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 the CDF of any normal distribution using the one of the standard normal distribution by applying Equation 35.153.

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{x} e^{-\frac{z^2}{2}} dz \tag{35.153}$$

$$P(X \leq x) = \Phi\left(\frac{x-\mu}{\sigma}\right) \tag{35.154}$$

You can find some values of $\Phi(x)$ in Table 35.7. 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 $\Phi(2.13) \approx 0.9834$.

[45] http://en.wikipedia.org/wiki/Normal_distribution [accessed 2007-07-03]

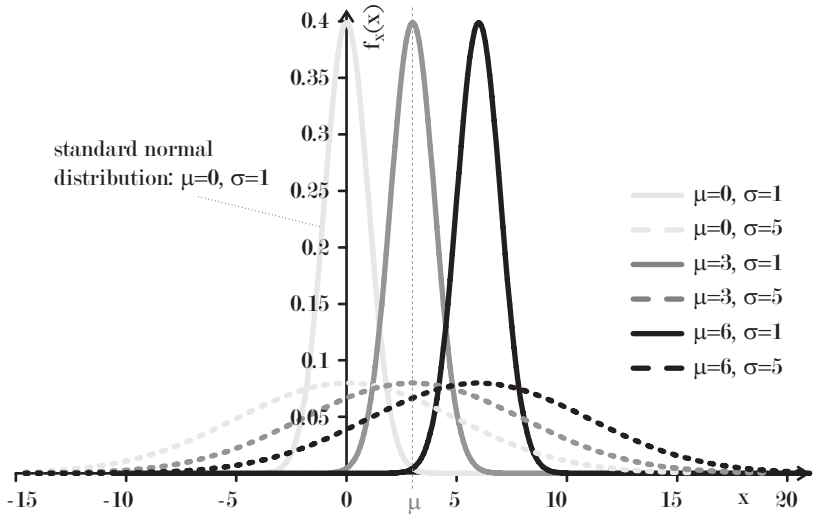


Fig. 35.9: The PDFs of some normal distributions

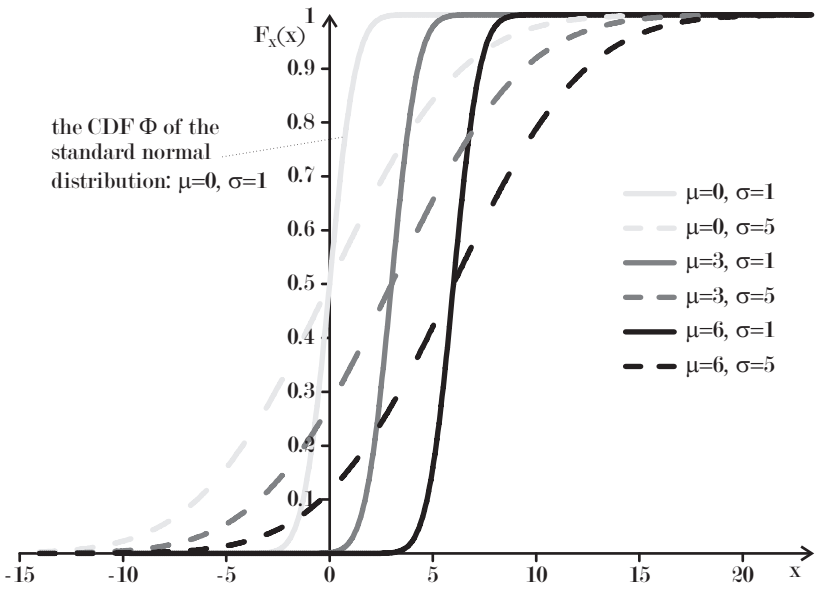


Fig. 35.10: The CDFs of some normal distributions

Table 35.7: Some values of the standardized normal distribution.

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

Definition 162 (Probit). The inverse cumulative distribution function of the standard normal distribution is called *probit* function. It is also often denoted as z-quantil of the standard normal distribution.

$$z(y) \equiv probit(y) \equiv \Phi^{-1}(y) \tag{35.155}$$

$$y = \Phi(x) \Rightarrow \Phi^{-1}(y) = z(y) = x \tag{35.156}$$

The probability density function PDF of the multivariate normal distribution[46] [1137, 1138, 1139] is illustrated in Equation 35.157 and Equation 35.158

[46] `http://en.wikipedia.org/wiki/Multivariate_normal_distribution` [accessed 2007-07-03]

in the general (where $\mathbf{\Sigma}$ is the covariance matrix) and in Equation 35.159 in the uncorrelated form. If the distributions, additional 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 35.160.

$$f_X(\mathbf{x}) = \frac{\sqrt{\mathbf{\Sigma}^{-1}}}{(2\pi)^{\frac{n}{2}}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})} \tag{35.157}$$

$$= \frac{1}{(2\pi)^{\frac{n}{2}} \mathbf{\Sigma}^{\frac{1}{2}}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T \mathbf{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})} \tag{35.158}$$

$$f_X(\mathbf{x}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_i} e^{-\frac{(x_i-\mu_i)^2}{2\sigma_i^2}} \tag{35.159}$$

$$f_X(\mathbf{x}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x_i-\mu_i)^2}{2\sigma^2}}$$

$$= \left(\frac{1}{2\pi\sigma^2}\right)^{\frac{n}{2}} e^{-\frac{\sum_{i=1}^{n}(x_i-\mu)^2}{2\sigma^2}} \tag{35.160}$$

35.4.3 Exponential Distribution $exp(\lambda)$

The exponential distribution[47] $exp(\lambda)$ [1140] is especially 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 35.3.2 on page 532) has to be determined. Its PDF is sketched in Figure 35.11 for some example values of λ the according cases of the CDF are illustrated Figure 35.12. The most important characteristics of the exponential distribution can be obtained from Table 35.8.

[47] http://en.wikipedia.org/wiki/Exponential_distribution [accessed 2007-07-03]

Table 35.8: Parameters of the exponential distribution.

parameter	definition	
parameters	$\lambda \in \mathbb{R}^+$	(35.161)
PDF	$f_X(x) = \begin{cases} 0 & \forall x \leq 0 \\ \lambda e^{-\lambda x} & \forall x > 0 \end{cases}$	(35.162)
CDF	$P(X \leq x) = F_X(x) = \begin{cases} 0 & \forall x \leq 0 \\ 1 - e^{-\lambda x} & \forall x > 0 \end{cases}$	(35.163)
mean	$EX = \frac{1}{\lambda}$	(35.164)
median	$med = \frac{\ln 2}{\lambda}$	(35.165)
mode	$mod = 0$	(35.166)
variance	$D^2X = \frac{1}{\lambda^2}$	(35.167)
skewness	$\gamma_1 = 2$	(35.168)
kurtosis	$\gamma_2 = 6$	(35.169)
entropy	$H(X) = 1 - \ln \lambda$	(35.170)
mgf	$M_X(t) = \left(1 - \frac{t}{\lambda}\right)^{-1}$	(35.171)
char. func.	$\varphi_X(t) = \left(1 - \frac{it}{\lambda}\right)^{-1}$	(35.172)

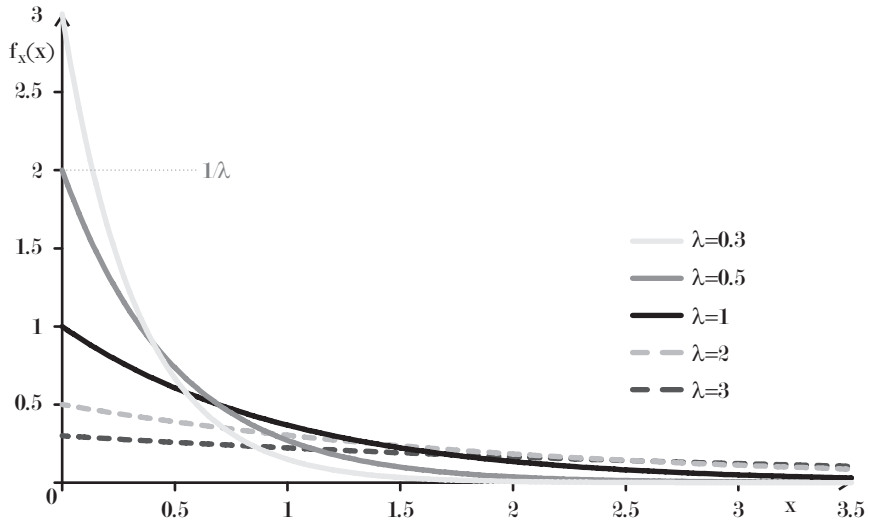


Fig. 35.11: The PDFs of some exponential distributions

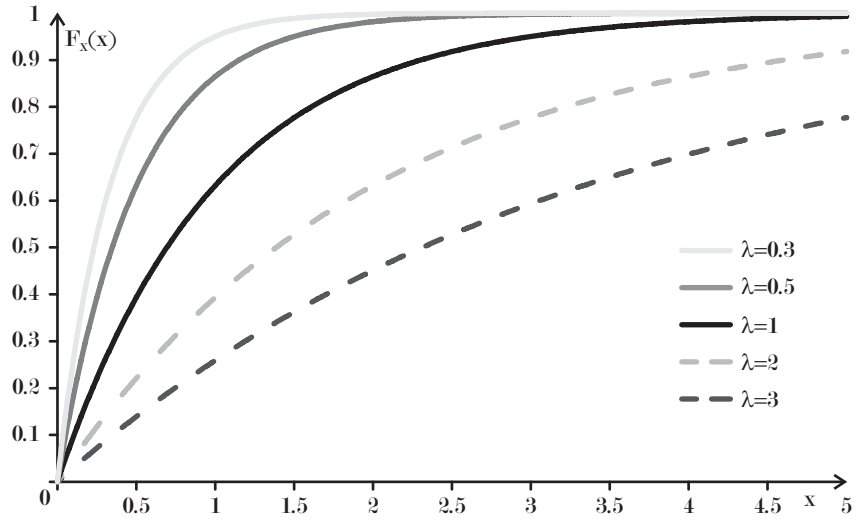


Fig. 35.12: The CDFs of some exponential distributions

35.4.4 Chi-square Distribution

The chi-square (or χ^2) distribution[48] 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. It is also used to describe the sum of independent standardized normal distributions. Its sole parameter, n, denotes the degrees of freedom.

In Table 35.9[49], 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 35.13 and Figure 35.14.

Table 35.10 provides some selected values of χ^2 distributions with n degrees of freedom. 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 $(n, F_X(x))$ combinations.

[48] `http://en.wikipedia.org/wiki/Chi-square_distribution` [accessed 2007-09-30]

[49] $\gamma(n, z)$ in Equation 35.175 is the lower incomplete Gamma function and $P_\gamma(n, z)$ is the regularized Gamma function.

Table 35.9: Parameters of the χ^2 distribution.

parameter	definition	
parameters	$n \in \mathbb{R}^+,\ n > 0$	(35.173)
PDF	$f_X(x) = \begin{cases} 0 & \forall x \leq 0 \\ \frac{1}{2^{n/2}\Gamma(n/2)} x^{n/2-1} e^{-x/2} & \forall x > 0 \end{cases}$	(35.174)
CDF	$P(X \leq x) = F_X(x) = \frac{\gamma(n/2, x/2)}{\Gamma(n/2)} = P_\gamma(n/2, x/2)$	(35.175)
mean	$EX = n$	(35.176)
median	$med = \approx n - \frac{2}{3}$	(35.177)
mode	$mod = n - 2$ if $n \geq 2$	(35.178)
variance	$D^2X = 2n$	(35.179)
skewness	$\gamma_1 = \sqrt{\frac{8}{n}}$	(35.180)
kurtosis	$\gamma_2 = \frac{12}{n}$	(35.181)
entropy	$H(X) = \frac{n}{2} + \ln(2\Gamma(n/2)) + (1 - n/2)\Psi(n/2)$	(35.182)
mgf	$M_X(t) = (1 - 2t)^{-n/2}$ for $2t < 1$	(35.183)
char. func.	$\varphi_X(t) = (1 - 2it)^{-n/2}$	(35.184)

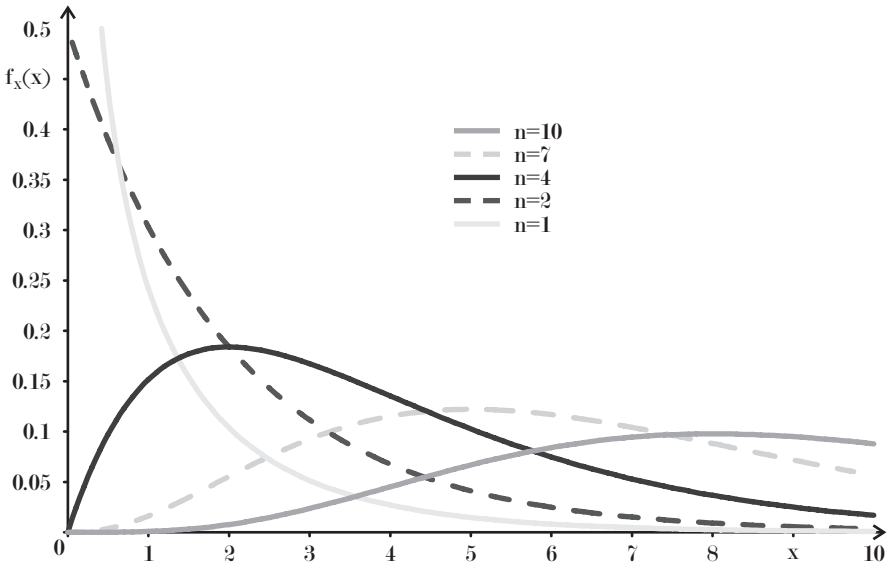


Fig. 35.13: The PDFs of some χ^2 distributions

Table 35.10: Some values of the χ^2 distribution.

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

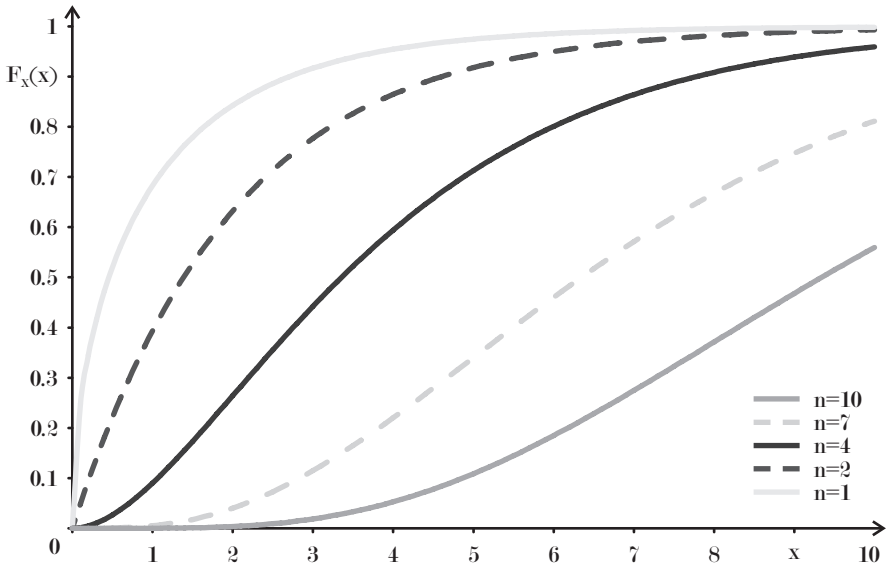


Fig. 35.14: The CDFs of some χ^2 distributions

35.4.5 Student's t-Distribution

The Student's t-distribution[50] 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 [1141, 1142, 1143]. It has been design by William Sealy Gossett 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 35.11[51] and examples for its PDF and CDF are illustrated in Figure 35.15 and Figure 35.16.

Table 35.12 provides some selected values for quantiles $t_{1-\alpha,n}$ of t-distributions with n degrees of freedom (one-sided confidence intervals, see Section 35.6.3 on page 556). The table's headline contains results of the cumulative distribution function $F_X(x)$ of a Student's t-distribution with n

[50] `http://en.wikipedia.org/wiki/Student%27s_t-distribution` [accessed 2007-09-30]

[51] More information on the gamma function (Γ) used in Equation 35.186 and Equation 35.187 can be found in Section 35.9.1 on page 570. ${}_2F_4$ in Equation 35.187 stands for the hypergeometric function, Ψ and B in Equation 35.194 are the digamma and the beta function.

degrees of freedom (values in the first column). The cells now denote the x values that belong to these $(n, F_X(x))$ combinations.

Table 35.11: Parameters of Student's t-distribution.

parameter	definition	
parameters	$n \in \mathbb{R}^+, n > 0$	(35.185)
PDF	$f_X(x) = \frac{\Gamma((n+1)/2)}{\sqrt{n\pi}\Gamma(n/2)} \left(1 + x^2/n\right)^{-(n+1)/2}$	(35.186)
CDF	$P(X \leq x) = F_X(x) = \frac{1}{2} + \frac{x\Gamma((n+1)/2)\ {}_2F_1\left(\frac{1}{2}, (n+1)/2; \frac{3}{2}; -\frac{x^2}{n}\right)}{\sqrt{n\pi}\Gamma(n/2)}$	(35.187)
mean	$EX = 0$	(35.188)
median	$med = 0$	(35.189)
mode	$mod = 0$	(35.190)
variance	$D^2X = \frac{n}{n-2}$ for $n > 2$	(35.191)
skewness	$\gamma_1 = 0$ for $n > 3$	(35.192)
kurtosis	$\gamma_2 = \frac{6}{n-4}$ for $n > 4$	(35.193)
entropy	$H(X) = \frac{n}{2}\left[\Psi\left(\frac{n+1}{2}\right) - \Psi\left(\frac{n}{2}\right)\right] + \log\left[\sqrt{n}B\left(\frac{n}{2}, \frac{1}{2}\right)\right]$	(35.194)
mgf	undefined	(35.195)

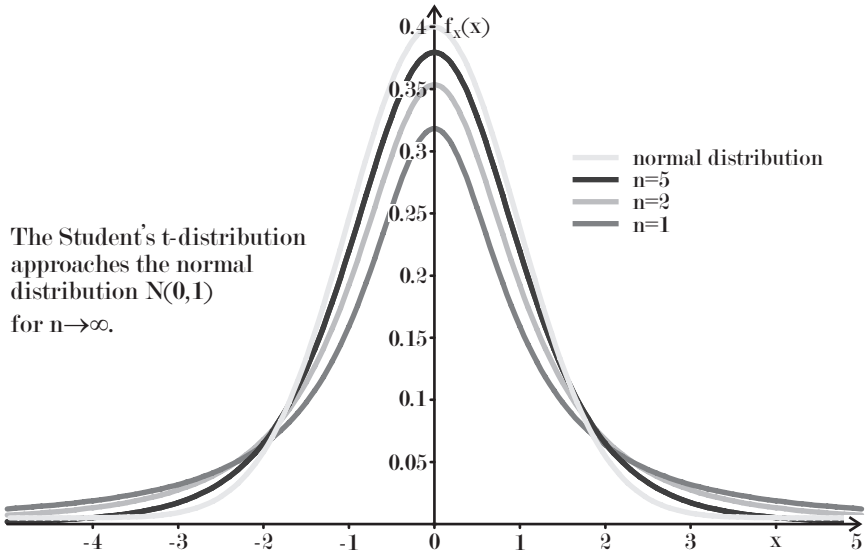


Fig. 35.15: The PDFs of some Student's t-distributions

Table 35.12: Table of Student's t-distribution with right-tail probabilities.

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

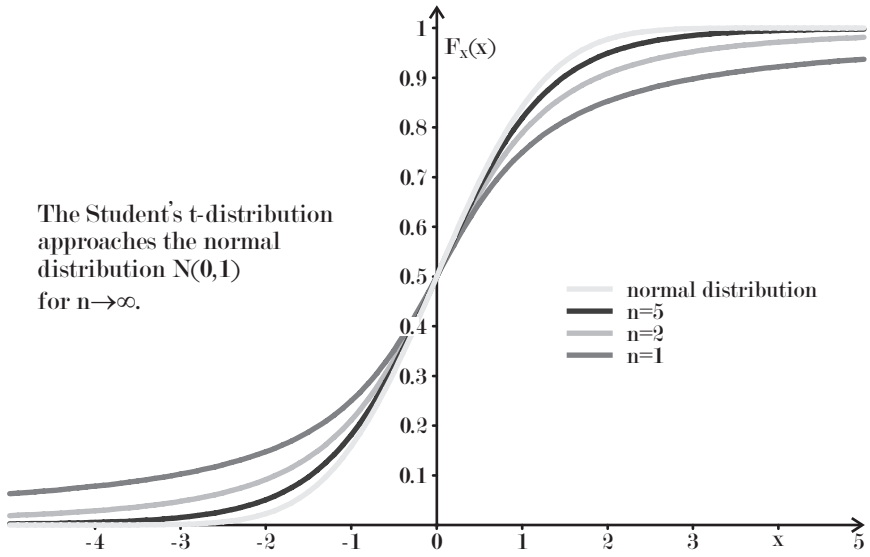


Fig. 35.16: The CDFs of some Student's t-distributions

35.5 Example - Throwing a Dice

Let us now discuss the different parameters of a random variable at the example of throwing a dice. On the dice, the numbers one to six are written and the result of throwing a dice 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

$$\Omega = \Big\{\boxed{1}, \boxed{2}, \boxed{3}, \boxed{4}, \boxed{5}, \boxed{6}\Big\} \tag{35.196}$$

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:

$$X \in \{1, 2, 3, 4, 5, 6\} \tag{35.197}$$

It is obviously a uniformly distributed discrete random variable (see Section 35.3.1 on page 527) 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 35.17):

$$F_X = P(X \leq x) = \begin{cases} 0 & if\ x < 1 \\ \frac{x}{6} & if\ 1 \leq x \leq 6 \\ 1 & otherwise\ (x > 6) \end{cases} \tag{35.198}$$

$$f_X = P(X = x) = \begin{cases} 0 & if\ x < 1 \\ \frac{1}{6} & if\ 1 \leq x \leq 6 \\ 0 & otherwise\ (x > 6) \end{cases} \tag{35.199}$$

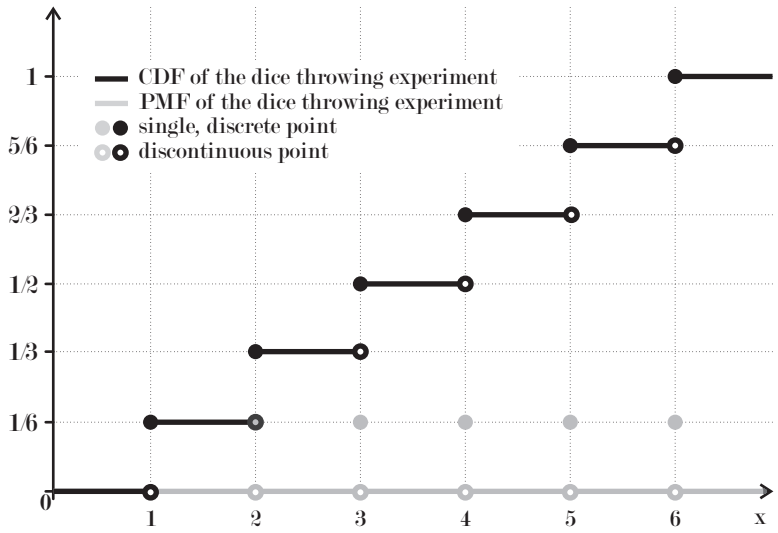


Fig. 35.17: The PMF and CMF of the dice throw

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 35.18):

$$A = \{4, 5, 3, 2, 4, 6, 4, 2, 5, 3\} \tag{35.200}$$

Table 35.13 outlines how the parameters of the random variable are computed. The real value 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.

35.6 Estimation Theory

Estimation theory is the science of estimating the values of parameters based on measurements or otherwise obtained sample data [1144, 1145, 1146, 1147,

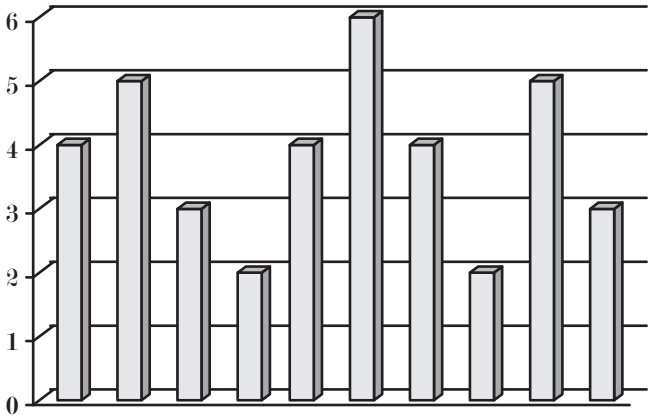


Fig. 35.18: The numbers thrown in the dice example

Table 35.13: The statistical parameters of the dice throw experiment

parameter	true value	estimation
count	non existent	$n = \|A\| = 10$
minimum	$a = \min\{x : f_X(x) > 0\} = 1$	$a = \min\{a \in A\} = 2$
maximum	$b = \max\{x : f_X(x) > 0\} = 6$	$b = \max\{a \in A\} = 6$
range	$range = r = b - a + 1 = 6$	$range = r = b - a + 1 = 6$
mean	$EX = \frac{a+b}{2} = \frac{7}{2} = 3.5$	$\overline{a} = \frac{1}{n}\sum_{i=0}^{n-1} a_i = \frac{19}{5}$
median	$med = \frac{a+b}{2} = \frac{7}{2} = 3.5$	$med = \frac{sort(A)\left[\frac{n}{2}\right]+sort(A)\left[\frac{n}{2}-1\right]}{2} = 4$
mode	$mod = \emptyset$	$mod = 4$
variance	$D^2X = \frac{r^2-1}{12} = \frac{35}{12} \approx 2.917$	$s^2 = \frac{1}{n-1}\sum_{i=0}^{n-1}(a_i - \overline{a})^2 = \frac{26}{15} \approx 1.73$
skewness	$\gamma_1 = 0$	$G_1 \approx 0.0876$
kurtosis	$\gamma_2 = -\frac{6(r^2+1)}{5(r^2-1)} = -\frac{222}{175} \approx -1.269$	$G_2 \approx -0.7512$

1148]. 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 163 (Estimator). An estimator[52] $\tilde{\theta}$ is a rule (most often a mathematical function) that takes a set of sample data as input and returns an estimation of one parameter θ of this data set.

We have already discussed some estimators in Section 35.2 – the arithmetic mean of a sample data set (see Definition 146 on page 521) for example is an estimator for the expected value (see Definition 144 on page 521) and

[52] `http://en.wikipedia.org/wiki/Estimator` [accessed 2007-07-03], `http://mathworld.wolfram.com/Estimator.html` [accessed 2007-07-03]

in Equation 35.65 on page 523 we have introduced an estimator for the sample variance.

Obviously, the estimator $\tilde{\theta}$ is the better the closer its results (the estimates) come to the real values of the parameter θ.

Definition 164 (Point Estimator). We define a point estimator $\tilde{\theta}$ to be an estimator which is a mathematical function $\tilde{\theta} : \mathbb{R}^n \mapsto \mathbb{R}$. This function takes the real vector $\mathbf{x} \in \mathbb{R}^n$ representing the sample data set as input and returns the estimate in the form of a real, scalar value.

Definition 165 (Error). The (estimation) error ε[53] is the difference between the value returned by a point estimator $\tilde{\theta}$ of a parameter θ for a certain input $\mathbf{x}$ and its real value. Notice that the error ε can be zero, positive, or negative.

$$\varepsilon(\tilde{\theta}, \mathbf{x}) = \tilde{\theta}(\mathbf{x}) - \theta \tag{35.201}$$

Definition 166 (Bias). The bias $Bias(\tilde{\theta})$ of an estimator $\tilde{\theta}$ is the expected value of the difference of the estimate and the real value. This mean error is null for all unbiased estimators.

$$Bias(\tilde{\theta}) = E\tilde{\theta} - \theta = E\,\varepsilon(\tilde{\theta}) \tag{35.202}$$

Definition 167 (Unbiased Estimator). An unbiased estimator has a zero bias.

$$Bias(\tilde{\theta}) = E\tilde{\theta} - \theta = E\,\varepsilon(\tilde{\theta}) = 0 \Leftrightarrow E\tilde{\theta} = \theta \tag{35.203}$$

Definition 168 (Mean Square Error). The mean square error[54] $MSE(\tilde{\theta})$ of an estimator $\tilde{\theta}$ is the expected value of the square of the error ε. It is also the sum of the variance of the estimator and the square of its bias. The MSE represents how much an estimator differs from the quantity to be estimated.

$$MSE(\tilde{\theta}) = E\left((\tilde{\theta} - \theta)^2\right) = E\left(\varepsilon(\tilde{\theta})^2\right) \tag{35.204}$$

$$MSE(\tilde{\theta}) = D^2\tilde{\theta} + \left(Bias(\tilde{\theta})\right)^2 \tag{35.205}$$

Notice that the MSE of unbiased estimators coincides with the variance of $\tilde{\theta}$.

For estimating the mean square error of an estimator $\tilde{theta}$, we use the sample mean:

$$\tilde{MSE}(\tilde{\theta}) = \frac{1}{n}\sum_{i=1}^{n}\left(\tilde{\theta}_i - \theta\right) \tag{35.206}$$

[53] http://en.wikipedia.org/wiki/Errors_and_residuals_in_statistics [accessed 2007-07-03]

[54] http://en.wikipedia.org/wiki/Mean_squared_error [accessed 2007-07-03]

35.6.1 Likelihood and Maximum Likelihood Estimators

Definition 169 (Likelihood). Likelihood[55] 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 170 (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.

$$L(\varphi|\mathbf{y}) \propto P(\mathbf{y}|\varphi) \tag{35.207}$$

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

We are given a finite set S of n sample data points.

$$S = \{(x_1, y_1), (x_{2,y}\, 2), \ldots, (x_n, y_n)\}, x_i, y_i \in \mathbb{R} \tag{35.208}$$

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[56] η_i.

$$y_i = \varphi(x_i) + \eta_i \; \forall 0 < i \leq n \tag{35.209}$$

About this measurement error η we make the following assumptions:

$$E\eta = 0 \tag{35.210}$$

$$\eta \sim N(0, \sigma^2) \; : \; 0 < \sigma < \infty \tag{35.211}$$

$$cov(\eta_i, \eta_j) = 0 \; \forall i \neq j, 0 < i \leq n, 0 < j \leq n \tag{35.212}$$

- The expected values of η in Equation 35.210 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 $E\eta_i$ from all measurements and would obtain unbiased observations.

[55] `http://en.wikipedia.org/wiki/Likelihood` [accessed 2007-07-03]

[56] `http://en.wikipedia.org/wiki/Measurement_error` [accessed 2007-07-03]

- Furthermore, Equation 35.211 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[57] in transmission of signals for example is often modeled with Gaussian distributed[58] amplitudes. This second assumption includes, of course, the first one: being normally distributed with $N(\mu = 0, \sigma^2)$ implies a zero expected value.
- With Equation 35.212, we expect the errors η_i of the single measurements to be 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) $f \in F$.

$$f \in F \Rightarrow f : \mathbb{R} \mapsto \mathbb{R} \tag{35.213}$$

From this set we want to pick the function $f^\star \in F$ with $f : \mathbb{R} \mapsto \mathbb{R}$ that resembles φ the best close (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 165 on page 551) from y_i-values. The estimation error depends on f and may vary for different estimators.

$$y_i = f(x_i) + \varepsilon_i(f) \; \forall 0 < i \leq n \tag{35.214}$$

We can regard all $f \in F$ as estimators for φ and simple look for the one that "fits best". We now can combine Equation 35.214 with Equation 35.209:

$$f(x_i) + \varepsilon_i(f) = y_i = \varphi(x_i) + \eta_i \; \forall 0 < i \leq n \tag{35.215}$$

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 35.211). In other words, 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 $\varepsilon_i(f^\star) = \eta_i$ and thus, $f^\star \equiv \varphi(x_i)$.

[57] http://en.wikipedia.org/wiki/White_noise [accessed 2007-07-03]
[58] http://en.wikipedia.org/wiki/Gaussian_noise [accessed 2007-07-03]

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 $\boldsymbol{\varepsilon}_i(f)$ to be the vector containing all the single $\varepsilon_i(f)$.

$$\boldsymbol{\varepsilon}(f) = \begin{pmatrix} \varepsilon_1(f) \\ \varepsilon_2(f) \\ \vdots \\ \varepsilon_n(f) \end{pmatrix} \tag{35.216}$$

The probability density function of a multivariate normal distribution with independent variables ε_i that have the same variance σ^2, as defined in Equation 35.160 on page 540, looks like this:

$$f_X(\boldsymbol{\varepsilon}(f)) = \left(\frac{1}{2\pi\sigma^2}\right)^{\frac{n}{2}} e^{-\frac{\sum_{i=1}^{n}(\varepsilon_i(f)-\mu)^2}{2\sigma^2}} \tag{35.217}$$

$$= \left(\frac{1}{2\pi\sigma^2}\right)^{\frac{n}{2}} e^{-\frac{\sum_{i=1}^{n}\varepsilon_i(f)^2}{2\sigma^2}} \tag{35.218}$$

Amongst all possible vectors $\boldsymbol{\varepsilon}(f) : f \in F$ we need to find the most probable one $\boldsymbol{\varepsilon}^\star = \boldsymbol{\varepsilon}(f)^\star$ according to Equation 35.218. 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 170. Here we can use the probability density f_X, since the maximal values of f_X are those that are most probable to occur.

$$L(\boldsymbol{\varepsilon}(f)|f) = f_X(\varepsilon(f)) = \left(\frac{1}{2\pi\sigma^2}\right)^{\frac{n}{2}} e^{-\frac{\sum_{i=1}^{n}\varepsilon_i(f)^2}{2\sigma^2}} \tag{35.219}$$

$$f^\star \in F : L(\boldsymbol{\varepsilon}(f^\star)|f^\star) = \max_{\forall f \in F} L(\boldsymbol{\varepsilon}(f)|f) \tag{35.220}$$

$$= \max_{\forall f \in F} \left(\frac{1}{2\pi\sigma^2}\right)^{\frac{n}{2}} e^{-\frac{\sum_{i=1}^{n}\varepsilon_i(f)^2}{2\sigma^2}} \tag{35.221}$$

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.

$$f^\star \in F : \sum_{i=1}^{n} \varepsilon_i(f^\star)^2 = \min_{\forall f \in F} \sum_{i=1}^{n} \varepsilon_i(f)^2 \tag{35.222}$$

According to Equation 35.214 we can now substitute the ε_i-values with the difference between the observed outcomes y_i and the estimates $f(x_i)$.

$$\sum_{i=1}^{n} \varepsilon_i(f)^2 = \sum_{i=1}^{n} (y_i - f(x_i))^2 \tag{35.223}$$

Definition 171 (Maximum Likelihood Estimator). A maximum likelihood estimator[59] [1149] $f^\star$ is an estimator which fits with maximum likelihood to a given set of sample data S. Under the particular assumption of uncorrelated error terms normally distributed around zero, a MLE minimizes Equation 35.224.

$$f^\star \in F : \sum_{i=1}^{n} (y_i - f^\star(x_i))^2 = \min_{\forall f \in F} \sum_{i=1}^{n} (y_i - f(x_i))^2 \tag{35.224}$$

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 168) is the best estimator according to the likelihood of the produced outcomes.

$$f^\star \in F : \frac{1}{n} \sum_{i=1}^{n} (y_i - f^\star(x_i))^2 = \min_{\forall f \in F} \frac{1}{n} \sum_{i=1}^{n} (y_i - f(x_i))^2 \tag{35.225}$$

$$f^\star \in F : MSE(f^\star) = \min_{\forall f \in F} MSE(f) \tag{35.226}$$

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 is however, that we maximize the likelihood of the resulting estimator.

At this point, one should also notice that the x_i 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 defined very closely. It usually contains one type of parameterized function so we only have to determine the unknown parameters in order to find $f^\star$. If we for example specify $F = \{\forall\, f(x) = ax + b : a, b \in R\}$, we have minimize Equation 35.227 for a and b.

$$MSE(f(x|a,b) = \frac{1}{n} \sum_{i=1}^{n} (ax_i + b - y_i)^2 \tag{35.227}$$

If we 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. This perfect estimator would be the solution of the over-determined system of linear equations illustrated in Equation 35.228.

[59] http://en.wikipedia.org/wiki/Maximum_likelihood [accessed 2007-07-03]

$$\begin{aligned} 0 &= ax_1 + b - y_1 \\ 0 &= ax_2 + b - y_2 \\ \dots &\quad \dots \\ 0 &= ax_n + b - y_n \end{aligned} \tag{35.228}$$

Since it is normally not possible to obtain such a perfect estimator (because there are measurement errors or other uncertainties like unknown dependencies), the system in Equation 35.228 often cannot be solved and only minimized.

35.6.2 Best Linear Unbiased Estimators

The Gauss-Markov Theorem[60] defines BLUEs (best linear unbiased estimators) according to the facts just discussed:

Definition 172 (BLUE). In a linear model in which the measurement errors η_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 [1150].

Hence, for the best linear unbiased estimator also the same three assumptions (Equation 35.210, Equation 35.211, and Equation 35.212 on page 552) as for the maximum likelihood estimator hold.

35.6.3 Confidence Intervals

Definition 173 (Confidence Interval). Unlike point estimators, which approximate a parameter of a data sample with a single value, confidence intervalls[61] (CIs) are estimations that give certain upper and lower boundaries in which the parameter will be with a predefined probability. [1151, 1152]

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. Narrow confidence intervals 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 an 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):

[60] `http://en.wikipedia.org/wiki/Gauss-Markov_theorem` [accessed 2007-07-03], `http://www.answers.com/topic/gauss-markov-theorem` [accessed 2007-07-03]

[61] `http://en.wikipedia.org/wiki/Confidence_interval` [accessed 2007-10-01]

$$A = \begin{Bmatrix} 120, 121, 119, 116, 115, 122, 121, 123, 122, 120 \\ 119, 122, 121, 120, 119, 121, 123, 117, 118, 121 \end{Bmatrix} \tag{35.229}$$

$$n = |A| = 20 \tag{35.230}$$

From these measurements, we can determine the arithmetic mean $\overline{a}$ and the sample variance s^2 according to Equation 35.58 on page 522 and Equation 35.65 on page 523:

$$\overline{a} = \frac{1}{n}\sum_{i=0}^{n-1} A[i] = \frac{2400}{20} = 120 \tag{35.231}$$

$$s^2 = \frac{1}{n-1}\sum_{i=0}^{n-1}(a_i - \overline{a})^2 = \frac{92}{19} \tag{35.232}$$

The question that arises now is if the mean of 120 is a significant value and in which intervals we can expect the egg weights generally to be. 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 that it is the normal distribution discussed in Section 35.4.2 on page 537. Knowing that, we can now calculate an interval which includes the unknown parameter μ (i. e. the real expected value) with a confidence probability of γ. $\gamma = 1 - \alpha$ is the so-called confidence coefficient and α 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 $\gamma = 1-\alpha = 95\%$. Thus, $\alpha = 0.05$. Therefore, we have to pick the right formula from Section 35.6.3 on the next page (here it is Equation 35.245 on the following page) and substitute in the proper values:

$$\mu_\gamma \in \left[\overline{a} \pm t_{1-\frac{\alpha}{2},n-1}\frac{s}{\sqrt{n}}\right] \tag{35.233}$$

$$\mu_{95\%} \in \left[120 \pm t_{0.975,19} * \frac{\sqrt{\frac{92}{19}}}{\sqrt{19}}\right] \tag{35.234}$$

$$\mu_{95\%} \in [120 \pm 2.093 * 0.5048] \tag{35.235}$$

$$\mu_{95\%} \in [118.94, 121.06] \tag{35.236}$$

The value of $t_{19,0.025}$ can easiliy be obtained from Table 35.12 on page 547 which contains the respective quantiles of Student's t-distribution discussed in Section 35.4.5 on page 545.

Let us repeat the procedure in order to find the interval that will contain μ with probabilities $1 - \alpha = 99\% \Rightarrow \alpha = 0.01$ and $1 - \alpha = 90\% \Rightarrow \alpha = 0.1$

$$\mu_{99\%} \in [120 \pm t_{0.995,19} * 0.5048] \tag{35.237}$$

$$\mu_{99\%} \in [120 \pm 2.861 * 0.5048] \tag{35.238}$$
$$\mu_{99\%} \in [118.56, 121.44] \tag{35.239}$$
$$\mu_{90\%} \in [120 \pm t_{0.95,19} * 0.5048] \tag{35.240}$$
$$\mu_{90\%} \in [120 \pm 1.729 * 0.5048] \tag{35.241}$$
$$\mu_{90\%} \in [119.13, 120.87] \tag{35.242}$$
$$\tag{35.243}$$

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 receive a range $p_\gamma \in [p' - x, p' + x]$ that contains the parameter p with γ probability based on the estimate p'. If you need a one-sided confidence interval like $p_\gamma \in (-\infty, p' + x]$ or $p_\gamma \in [p' - x, \infty)$, replace $1 - \frac{\alpha}{2}$ with $1 - \alpha$ in the equations.

Expected Value of a Normal Distribution

With known 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 $\overline{x}$ according to Equation 35.58 on page 522, the two-sided confidence interval (of probability γ) for the expected value of the normal distribution is:

$$\mu_\gamma \in \left[\overline{x} \pm z\left(1 - \frac{\alpha}{2}\right)\frac{\sigma}{\sqrt{n}}\right] \tag{35.244}$$

Where $z(y) \equiv \Phi^{-1}(y)$ is the y-quantil of the standard normal distribution (see Definition 162 on page 539) which can for example be looked up in Table 35.7.

With estimated sample variance s^2: If the true variance of the distribution is not known and instead estimated with the sample variance s^2 according to Equation 35.65 on page 523. The two-sided confidence interval (of probability γ) for the expected value can then be computed using the arithmetic mean $\overline{x}$ and the estimate of the standard deviation $s = \sqrt{s^2}$ of the sample and the $t_{n-1,1-\frac{\alpha}{2}}$ quantil of Student's t-distribution which can be looked up in Table 35.12 on page 547.

$$\mu_\gamma \in \left[\overline{x} \pm t_{1-\frac{\alpha}{2},n-1}\frac{s}{\sqrt{n}}\right] \tag{35.245}$$

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 $\chi^2(p,k)$-quantil of the χ^2 distribution which can be looked up in Table 35.10 on page 544.

$$\sigma_\gamma^2 \in \left[\frac{(n-1)s^2}{\chi^2\left(1-\frac{\alpha}{2}, n-1\right)}, \frac{(n-1)s^2}{\chi^2\left(\frac{\alpha}{2}, n-1\right)} \right] \tag{35.246}$$

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:

$$p_\gamma \in \left[\frac{n}{n + z_{1-\frac{\alpha}{2}}^2} \left(\overline{x} + \frac{1}{2n} z_{1-\frac{\alpha}{2}}^2 \pm z_{1-\frac{\alpha}{2}}^2 \sqrt{\frac{\overline{x}(1-\overline{x})}{n} + \left(\frac{1}{2n} z_{1-\frac{\alpha}{2}}^2 \right)^2} \right) \right] \tag{35.247}$$

Expected Value of an Unknown Distribution with Sample Variance

The two-sided confidence interval (of probability γ) of the expected value μ of an unknown distribution with an unknown real variance σ^2 can be determined using the arithmetic mean $\overline{x}$ and the sample variance s^2 if the sample data set contains more than $n = 50$ elements.

$$\mu_\gamma \in \left[\overline{x} z \left(1 - \frac{\alpha}{2}\right) \frac{s}{\sqrt{n}} \right] \tag{35.248}$$

35.7 Generating Random Numbers

Definition 174 (Random Number). Random numbers are the values taken on by a random variable. A random number generator[62] produces a sequence $r = (r_1, r_2, \ldots)$ 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 obeying the law of large numbers (see Section 35.2.8 on page 527).

[62] `http://en.wikipedia.org/wiki/Random_number_generator` [accessed 2007-07-03], `http://en.wikipedia.org/wiki/Random_number_generation` [accessed 2007-07-03]

For true random number generators, there exists no function or algorithm $f(i) = r_i$ or $f(r_{i-n+1}, r_{i-n+2}, \ldots, r_i) = r_{i+1}$ 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. Today, there exist many such so-called hardware random number generators [63] [1153, 1154, 1155, 1156].

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 however 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.

35.7.1 Generating Pseudorandom Numbers

In security-sensitive areas like cryptography, we need true random numbers [1157, 1156, 1158]. For normal PC applications and most scientific purposes pseudorandom number generators[64] are sufficient.

The principle of pseudorandom number generators is to produce a sequence of numbers $r = (r_1, r_2, \ldots, r_i), r_j \in R \, \forall j \in \mathbb{N}, R \subseteq \mathbb{R}$ which are not *obviously* interdependent, i. e. if knowing a number r_i there is not a simple way to find out the value of r_{i+1}.

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 variable. This variable is referred to as seed and normally changes 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. 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 [1159, 1160, 1161]. It should be

[63] `http://en.wikipedia.org/wiki/Hardware_random_number_generator` [accessed 2007-07-03]

[64] `http://en.wikipedia.org/wiki/Pseudorandom_number_generator` [accessed 2007-07-03], `http://en.wikipedia.org/wiki/Pseudorandomness` [accessed 2007-07-03]

noted that there also exist cryptographically secure pseudorandom number generators[65] which here could provide a valuable alternative.

There exists a variety of algorithms that generate pseudorandom numbers [1162, 1163, 1164, 1165] and many implementations for different programming languages and architectures [1166, 1167, 1168].

Linear Congruential Generator (LCG)

The linear congruential generator[66] (LCG) was first proposed by Lehmer [1169, 1170] and is one of the most frequently used and simplest pseudo random number generators. It updates an internal integer number $v \in V = (0 \ldots (m-1)), m \in \mathbb{N}$ in each step according to Equation 35.249. The modulus m is a natural number which defines the maximum count 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 uniformly distributed in the interval $[0, m)$ (see Section 35.3.1) and can be computed as proposed in Equation 35.250.

$$v_i = (av_{i-1} + b) mod\ m \tag{35.249}$$

$$r_i = \frac{v_i}{m} \tag{35.250}$$

If the full period can really be reached depends a lot on the parameters a, b, and m. There are many constellations known where only a small fraction of the period m is utilized [1171]. In order to produce the full period, the following requirements should be met according to wikipedia.

- b and m are relatively prime
- $a - 1$ is divisible by all prime factors of m
- $a - 1$ is a multiple of 4 if m is a multiple of 4
- $m > \max(a, b, v_0)$
- $a > 0, b > 0$

Good standard values for the constants are $a = 1'664'525$, $b = 1'013'904'223$, and $m = 2^{32}$. Knuth describes the realization of LCGs in [1172]. In Java, the class `java.util.Random` uses this approach with the settings $a = 25'214'903'917$, $b = 11$, and $m = 2^{48}$.

35.7.2 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 random number generator

[65] `http://en.wikipedia.org/wiki/Cryptographically_secure_pseudorandom_number_generator` [accessed 2007-07-03]

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

for uniformly distributed random numbers like elaborated in Section 35.7.1 but may need normally distributed random numbers.

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

$$s_i = a_2 + (b_2 - a_2)\frac{r_i - a_1}{b_1 - a_1} \tag{35.251}$$

Uniform Distribution → Normal Distribution

In order to transform random numbers uniformly distributed in the interval $[0, 1)$ to standard-normally distributed random numbers ($\mu = 0$, $\sigma^2 = 1$), we can apply the Box-Muller[67] transformation [1173]. This approach creates two standard-normally distributed random numbers n_1, n_2 from two random numbers r_1, r_2 uniformly distributed in $[0, 1)$ according to Equation 35.252. 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.

$$\begin{aligned} n_1 &= \sqrt{-2\ln r_1}\cos(2\pi r_2) \\ n_2 &= \sqrt{-2\ln r_1}\sin(2\pi r_2) \end{aligned} \tag{35.252}$$

The polar form of this method, illustrated as Algorithm 35.1, is not only faster, but also numerically more robust [1140]. 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 can now 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 35.252.

The implementation of this algorithm is discussed in [1172] which is the foundation of the method `nextGaussian` of the Java-class `java.util.Random`.

Normal Distribution → Normal Distribution

With Equation 35.253, a normally distributed random number $n_1 \sim N(\mu_1, \sigma_1^2)$ needs to be transformed to another normally distributed random number $n_2 \sim N(\mu_2, \sigma_2^2)$.

$$n_2 = \mu_2 + \sigma_2 * \frac{n_1 - \mu_1}{\sigma_1} \tag{35.253}$$

[67] `http://en.wikipedia.org/wiki/Box_muller` [accessed 2007-07-03]

Algorithm 35.1: $(n_1, n_2) = random_{n,p}()$

Data: n_1,n_2 the intermediate and result variables
Data: w the polar radius
Output: A tuple (n_1, n_2) of two 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 35.254, a random number r uniformly distributed in the interval $(0, 1)$ (0 is excluded) can be transformed into a exponentially distributed random number $s \sim exp(\lambda)$.

$$s = \frac{-\ln r}{\lambda} \tag{35.254}$$

Exponential Distribution → Exponential Distribution

With Equation 35.255, an exponentially distributed random number $r_1 \sim exp(\lambda_1)$ can be transformed to an exponentially distributed number $r_2 \sim exp(\lambda_2)$.

$$r_2 = \frac{\lambda_1}{\lambda_2} r_1 \tag{35.255}$$

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 bell-shaped, the result will be smaller or larger than x with the same probability and results which are close to x are 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 Wongpoowarak [880, 881], defined here as Algorithm 35.2. It is algorithmically close to the polar form of

the Box-Muller transform for the normal distribution (see Algorithm 35.1 on the previous page) 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.

Algorithm 35.2: $y = random_{bs}(\mu, \sigma)$

Input: μ the mean value of the bell-shaped distribution
Input: σ the standard deviation of the bell-shaped distribution
Data: w a uniformely distributed random number $w \in (0, 1)$
Output: A bell-shaped distributed random number y

```
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
```

You may have wondered about the factor 0.5513 in the algorithm. This number "normalizes" the standard deviation of the bell-shaped distribution, since $D^2X\ \left(r(y) = \ln\left(\frac{1-y}{y}\right)\right) \neq 1$. We can show this by first determining the cumulative distribution function $F_X(x)$ for $r(y)$ in Equation 35.258 and then differentiating in order to obtain the probability density function $f_X(x)$ (Equation 35.260).

$$F_X(x) \equiv r^{-1}(0,1) \tag{35.256}$$

$$x = r(y) = \ln\left(\frac{y}{1-y}\right) \tag{35.257}$$

$$F_X(x) = y = \frac{e^x}{1+e^x} \tag{35.258}$$

$$f_X(x) = F'_X(x) = F_X(x)\frac{dx}{dy} \tag{35.259}$$

$$\left(\frac{e^x}{1+e^x}\right)' = \frac{e^x\left(1+e^x\right) - e^x\left(e^x\right)}{\left(1+e^x\right)^2}$$

$$f_X(x) = \frac{e^x}{\left(1+e^x\right)^2} \tag{35.260}$$

Unfortunately, here it stops. We can neither apply Equation 35.54 on page 521 or Equation 35.61 on page 522 in order to determine the expected value or the variance, since both will result in integrals that the author cannot

```
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 35.1: Approximating D^2X of $r(y)$.

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 35.1 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 an 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 $D^2X = EX^2 - (EX)^2$ (see Equation 35.59 on page 522) to $D^2X = EX^2$.

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 35.63 on page 522) so the standard deviation the bell-shaped distribution $random_{bs}$ will become $DX\ random_{bs}(\mu, \sigma) \approx \sigma$.

35.7.3 Definitions of Random Functions

Definition 175 (Random Function). A random function *random* is a construct that eases the utilization of random numbers and random variables in the algorithms of this book. 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 176 (Uniform Distributed Random Number Generator). We define the function $random_u(r_{min}, r_{max})$ to draw uniformly distributed

(see Section 35.4.1 on page 535) random numbers from the interval with the boundaries r_{min} (inclusive) and r_{max} (exclusive). The parameter-less function $random_u()$ will return an uniformly distributed number from the interval spanning from 0 inclusively to 1 exclusively.

$$random_u(r_{min}, r_{max}) \in [r_{min}, r_{max}) \subseteq \mathbb{R}, r_{min} \in \mathbb{R}, r_{max} \in \mathbb{R} \quad (35.261)$$

$$random_u(r_{max} \in \mathbb{R}) \equiv random_u(0, r_{max}) \quad (35.262)$$

$$random_u() \equiv random_u(0, 1) \quad (35.263)$$

The $random_u$-function can be realized with, for example, linear congruential pseudorandom number generators as described in Section 35.7.1.

Definition 177 (Normal Distributed Random Number Generator). We define the function $random_n(\mu, \sigma^2)$ to draw normally distributed (see Section 35.4.2 on page 537) random numbers with the expected value μ and the variance σ^2. The parameter-less function $random_n()$ will return a normally distributed number with $\mu = 0$ and $\sigma^2 = 1$.

$$random_n(\mu, \sigma^2) \sim N(\mu, \sigma^2) \quad (35.264)$$

$$random_n() \equiv random_n(0, 1) \quad (35.265)$$

Cut-off Random Functions

We 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[68] around some mean μ with a variance $\sigma^2 \neq 0$. In the simulation, a series of eggs weights is created simply be drawing subsequent such random numbers by calling $random_n(\mu, \sigma^2)$ 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 ($> 10tons$) weights. In reality however, such has not yet been observed.

What we need here is a cut-off mechanism for our random function $random_n$ that still preserves as many of its properties as possible. Given a random function $random$, the function $random_l$, defined as Algorithm 35.3, ensures that $low \leq random_l(random, low, high) < high$).

[68] see Section 35.4.2 on page 537

Algorithm 35.3: $r = random_l(random, low, high)$

Input: *random* a random function (maybe with further implicit parameters)
Input: $low \in \mathbb{R}$ the inclusive, lower bound of the random result
Input: $high \in \mathbb{R}, high > low$ the exclusive, upper bound of the random result
Data: r the intermediate random value
Output: a value r 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
```

35.8 Density Estimation

In this section we discuss density estimation[69] techniques [1174, 1175] which are used by several optimization algorithms in order to check how crowded areas of the solution space are, for instance.

Definition 178 (Density Measure). A density estimation $\rho(x)$ approximates an unobservable probability density function $f_X(x)$ (see Section 35.1.8 on page 519) using a set of sample data X_s (see Equation 35.44).

$$\rho(x) \approx f_X(x) \tag{35.266}$$

$$\rho : \mathbb{R} \to [0, \infty) \tag{35.267}$$

35.8.1 Histograms

TODO

35.8.2 The k^{th} Nearest Neighbor Method

Definition 179 (k^{th} Nearest Neighbor Distance). The k^{th} nearest neighbor distance function $distf_{nn,k}$ is the distance of one element x to its k^{th} nearest neighbor in the set of all elements X_s. It relys on a distance measure (here called *dist*) to compute the element distances. See Section 36.1 on page 574 for more details.

$$distf_{nn,k}(x, X_s) = dist(x, x_k) \ : \ |\forall x_s \in X_s : dist(x_s, x) < dist(x_k, x)| = k - 1 \tag{35.268}$$

[69] http://en.wikipedia.org/wiki/Density_estimation [accessed 2007-07-03]

Using the k^{th} nearest neighbor method [1176], the PDF of an element x is estimated by its distance to its k^{th} nearest neighbor x_k in the test set X_s (with $k < |X_s|$). K nearest neighbor uses internally the Euclidian distance measure $dist_{eucl} \equiv dist_{n,2}$ (see Definition 188 on page 574), but theoretically any other one of the distance measures presented in Section 36.1 could also be applied.

$$\rho_{nn,k}(x) = \frac{k}{2\,|X_s|\,dist_{nn}^{k}(x, X_s)} \tag{35.269}$$

Normally, k is chosen to be $\sqrt{|X_s|}$.

35.8.3 Crowding Distance

Crowding distance [347] treats every element $x \in X_s$ 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 x we regard every single dimension i of the element x separately. For each of its dimensions, we determine the nearest neighbor to the left x^l and the nearest neighbor to the right x^r. The crowding distance of the element x in the dimension i is then $x_i^r - x_i^l$, the distance of the (objective) values of the left and right neighbors of x in the dimension i. This distance is normalized so that the maximum crowding distance of all elements in X_s 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 test set X_s, its crowding distance in the dimension is also set to 1.

The original source [347] does not mention normalization explicitly and sets the crowding distance of edge elements to ∞, which both is 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 X_s which could influence processes relying on the crowding distance in a very strong way.

The total crowding distance of an element x is the sum of the distance values corresponding to each dimension. Algorithm 35.4 on the facing page computes a function $\mathfrak{cd}(x)$ which relates each element x to its crowding distance. Since computing the crowding distance can be performed best by sorting the individuals according to their values in the single dimensions, we define the sorting function $ds_i(a, b)$ as follows:

$$ds_i(a,b) = \begin{cases} -1 \; if \; a_i < b_i \\ 0 \; if \; a_i = b_i \\ 1 \; if \; a_i > b_i, \quad a, b \in X_s \end{cases} \tag{35.270}$$

Algorithm 35.4: $\mathfrak{cd}(x) = computeCrowdingDistance(X_s)$

Input: X_s the set of sample data
Data: dd a list used as store for the crowind distances of the single dimensions
Data: X_o the list representation of X_s
Data: dim the dimension counter
Data: j the element counter
Data: max the maximum crowding distance of the current dimension
Output: the crowding distance function $\mathfrak{cd}$

```
1  begin
2      dd ⟵ createList(|X_s|, 0)
3      dd[0] ⟵ 1
4      dd[|X_s| − 1] ⟵ 1
5      X_o ⟵ setToList(X_s)
6      dim ⟵ n
7      while dim > 0 do
8          X_o ⟵ sort_a(X_o, ds_dim)
9          max ⟵ 0
10         j ⟵ |X_s| − 2
11         while j > 0 do
12             dd[j] ⟵ X_o[j + 1]_dim − X_o[j − 1]_dim
13             if dd[j] > max then max ⟵ dd[j]
14             j ⟵ j − 1
15         if max > 0 then
16             j ⟵ |X_s| − 2
17             while j > 0 do
18                 dd[j] ⟵ dd[j]/max
19                 j ⟵ j − 1
20         j ⟵ |X_s| − 1
21         while j ≥ 0 do
22             cd(X_o[j]) ⟵ cd(X_o[j]) + dd[j]
23             j ⟵ j − 1
24         dim ⟵ dim − 1
25     return cd
26 end
```

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 $\mathfrak{cd}$ indicate dense portions of X_s. A density estimate derived from the crowding distance will therefore be inversely proportional to it. We thus define $\rho_{\mathfrak{cd}}$ as the difference of 1 and $\mathfrak{cd}(x)$ divided by n, obtaining a value in $[0, 1]$ that is big if x is crowded region and small if it is situated in a sparsely covered area. This density estimate is mathematically not fully correct, it only displays the crowding information.

$$\rho_{\mathfrak{cd}}(x) = 1 - \frac{\mathfrak{cd}(x)}{n} \tag{35.271}$$

35.8.4 Parzen Window / Kernel Density Estimation

Another density estimation is the Parzen window method[70], also called kernel density estimation [1177].

TODO

35.9 Functions Often used in Statistics

35.9.1 Gamma Function

Definition 180 (Gamma Function). The Gamma function[71] $\Gamma : \mathbb{C} \mapsto \mathbb{R}$ is the extension of the factorial (see Definition 125 on page 514) to the real and complex numbers. For complex numbers $z \in \mathbb{C}$ with a positive real part $Re(z) > 0$ it is defined as:

$$\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt \tag{35.272}$$

$$\Gamma(z+1) = z\Gamma(z) \tag{35.273}$$

$$\Gamma(1) = 1 \tag{35.274}$$

$$\Gamma(z) = (z-1)! \; \forall \, z \in \mathbb{N} \tag{35.275}$$

$$\Gamma(z) = \lim_{n \to \infty} \frac{n! n^z}{z(z+1)\dots(z+n)} \tag{35.276}$$

$$\Gamma(z) = \frac{e^{\gamma z}}{z} \prod_{n=1}^{\infty} \left(1 + \frac{z}{n}\right)^{-1} e^{\frac{z}{n}} \tag{35.277}$$

γ in Equation 35.277 denotes the Euler-Mascheroni constant[72].

[70] http://en.wikipedia.org/wiki/Parzen_window [accessed 2007-07-03]

[71] http://en.wikipedia.org/wiki/Gamma_function [accessed 2007-09-30]

[72] http://en.wikipedia.org/wiki/Euler-Mascheroni_constant [accessed 2007-09-30]

36

Clustering

Clustering algorithms[1] divide a dataset into several disjoint subsets. All elements in a subset share common features like, for example, spatial proximity. Clustering has many different applications like:

- data mining [1178, 1179, 1180, 1181],
- information processing and management [1182, 1183, 1184, 1185],
- pattern recognition [1186, 1187, 1188],
- image processing [1018, 1189], and
- medicine [1190, 1191, 1192].

Definition 181 (Clustering). Clustering is the unsupervised classification of patterns (observations, data items, or feature vectors) into groups (clusters) [1193]. 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 36.1 illustrates a possible result B of the application of a clustering algorithm to a set A of elements with two features.

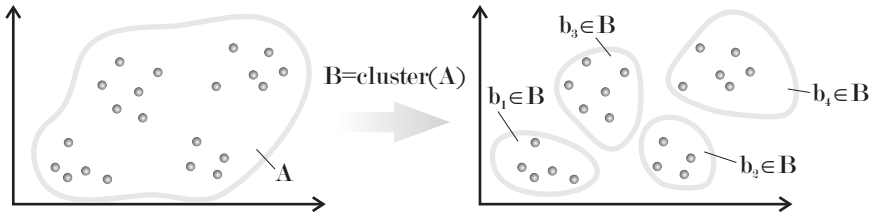


Fig. 36.1: A clustering algorithm applied to a two-dimensional dataset A.

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

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 is preserved, just their number is reduced. This is especially the case in elitist evolutionary algorithms (see Definition 37 on page 55) 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 find successively new clusters. The result of such an algorithm is a hierarchy of clusters. Partitional algorithms on the other hand determine 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] [1194, 1195] methods that do not create clear divisions but assign a vector of probabilities to each element. This vector contains a component for each cluster that 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 182 (Clustering Algorithm). A clustering algorithm *cluster* constructs a set B which elements are disjoint subsets of a set A and, if united, cover A completely (see also Figure 36.1).

$$\begin{aligned} B = cluster(A) \Rightarrow\ & \forall b \in B, \forall a \in b \Rightarrow a \in A \wedge \\ & \forall b_1 \neq b_2,\ b_1, b_2 \in B \Rightarrow b_1 \cap b_2 = \emptyset \wedge \\ & \forall a \in A\ \exists b \in B : a \in b \qquad (36.1) \\ & \textit{deduced: } \bigcup_{\forall b \in B} = A \qquad (36.2) \\ & \textit{deduced: } B \subset \mathcal{P}(A) \qquad (36.3) \end{aligned}$$

For the last deduced formula see the definition of the power set $\mathcal{P}$, Definition 94 on page 504.

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 *equal* elements. This is our little dirty backdoor here, we consider $A = \{a_1, a_2, \ldots, a_n\}$ 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

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

and elegant set-based definitions as done in Definition 182 without loss of generality.

Definition 183 (Partitions in Clustering). We define the set $\mathbb{B}$ of all possible partitions of A into clusters B. Furthermore, the subset $\mathbb{B}_k \subseteq \mathbb{B}$ is the set of all partitions of A into k clusters. The count of possible configurations $\mathbb{B}_k$ for a given k equals the Sterling number $S(|A|, k)$ [1196].

$$\begin{aligned} \forall B \in \mathbb{B} \Leftrightarrow \forall b \in B, \forall a \in b \Rightarrow a \in A \wedge \\ \forall b_1, b_2 \in B \Rightarrow b_1 \cap b_2 = \emptyset \wedge \\ \forall a \in A \, \exists b \in B \ : \ a \in b \end{aligned} \tag{36.4}$$

$$B \in \mathbb{B}_k \Leftrightarrow B \in \mathbb{B} \wedge |B| = k \tag{36.5}$$

$$|\mathbb{B}|_k = S(|A|, k) = \frac{1}{k!} \sum_{i=1}^{k} (-1)^{k-i} \binom{k}{i} i^n \tag{36.6}$$

$$|\mathbb{B}| = \sum_{k=1}^{n} |\mathbb{B}_k| = \sum_{k=1}^{n} S(|A|, k) \tag{36.7}$$

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 the set 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.

In the context of global optimization, a would for example be solution candidates like evolved programs 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 most probably not possible to have for each fitness configuration for a given problem one program that scores exactly this fitness.

Definition 184 (Centroid). The centroid[3] [1197] of a cluster is its center, the average of all its points to put it plain and simple.

$$centroid(b) = \frac{1}{|b|} \sum_{\forall a \in b} a \tag{36.8}$$

[3] `http://en.wikipedia.org/wiki/Centroid` [accessed 2007-07-03]

36.1 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 185 (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.

$$dist(a_1, a_2) \in \mathbb{R}^+,\ a_1 \in A, a_2 \in A \tag{36.9}$$

36.1.1 Distance Measures for Strings of Equal Length

Definition 186 (Hamming Distance). The Hamming Distance[4] [1198] $dist_{ham}(a_1, a_2)$ denotes the number of positions for which the corresponding symbols are different. 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.

36.1.2 Distance Measures for Real-Valued Vectors

As already mentioned in Chapter 36, we assume that there is a real-values vector in $\mathbb{R}^n$ assigned to each element $a \in A$ by an implicit $\alpha : A \Leftarrow \mathbb{R}^n$-function. Therefore, the distance measures introduced here can be used for all A subject to clustering.

Definition 187 (Manhattan Distance). The Manhattan distance[5] $dist_{man}(a_1, a_2)$ denotes the sum of the absolute distances of the coordinates of the two vectors.

$$dist_{man}(a_1, a_2) = \sum_{i=1}^{n} |a_{1,i} - a_{2,i}|\ \forall a_1, a_2 \in A \subseteq \mathbb{R}^n \tag{36.10}$$

Thus, the Manhattan distance of $(1, 2, 3)^T$ and $(3, 2, 1)^T$ is 4.

Definition 188 (Euclidian Distance). The Euclidian distance[6] $dist_{eucl}(a_1, a_2)$ is the "ordinary" distance of two points (denoted by the two vectors a_1 and a_2)

[4] `http://en.wikipedia.org/wiki/Hamming_distance` [accessed 2007-07-03]
[5] `http://en.wikipedia.org/wiki/Manhattan_distance` [accessed 2007-07-03]
[6] `http://en.wikipedia.org/wiki/Euclidean_distance` [accessed 2007-07-03]

in Euclidian space. This value is obtained by application of the Pythagorean theorem[7].

$$dist_{eucl}(a_1, a_2) = \sqrt{\sum_{i=1}^{n} (a_{1,i} - a_{2,i})^2} \; \forall a_1, a_2 \in A \tag{36.11}$$

Therefore, the Euclidian distance of $(1,2,3)^T$ and $(3,2,1)^T$ is $\sqrt{8}$.

Definition 189 (Norm). A vector norm[8], denoted by $||a||$ is a function which assigns a positive length or size to all vectors a in a vector space, other than the zero vector.

Some common norms of the element $a_i \in A$ are:

- The Manhattan norm[9]:
$$||a_i||_1 = \sum_{j=1}^{n} |a_{i,j}|$$
- The Euclidian norm:
$$||a_i||_2 = \sqrt{\sum_{j=1}^{n} (a_{i,j})^2}$$
- The p-norm is a generalization of the two examples above:
$$||a_i||_p = \left(\sum_{j=1}^{n} (a_{i,j})^p \right)^{\frac{1}{p}}$$
- The infinity norm[10] is the special case of the p-norm for $p \to \infty$:
$$||a_i||_\infty = \max \{|a_{i,1}|, |a_{i,2}|, ..., |a_{i,n}|\}$$

Such norms can be used as distance measures, and we hence define a new distance measurement function as:

$$dist_{n,p}(a_1, a_2) = ||a_1 - a_2||_p \; \forall a_1, a_2 \in A \subseteq \mathbb{R}^n \tag{36.12}$$
$$dist_{man} \equiv dist_{n,1} \tag{36.13}$$
$$dist_{eucl} \equiv dist_{n,2} \tag{36.14}$$

If the places of the vectors a have different ranges, for example $a_{.,1} \in [0...1]$ and $a_{.,2} \in [0...100000]$, a norm of the difference of two such vectors may not

[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]

represent their true distance. Therefore, an additional distance measure, the $dist_{norm^2,p}$-distance is used which normalizes the vector places before finally computing the norm.

$$span_j = \max\{|a_{i_1,j} - a_{i_2,j}|,\ a_{i_1}, a_{i_2} \in A\} \tag{36.15}$$

$$divisor_j = \begin{cases} span_j \text{ if } span_j > 0 \\ 1 \text{ else} \end{cases} \tag{36.16}$$

$$dist_{n^2,p}(a_1, a_2) = \left(\sum_{i=1}^{n} \left(\frac{a_{1,i} - a_{2,i}}{divisor_i} \right)^p \right)^{\frac{1}{p}} \tag{36.17}$$

36.1.3 Distance Measures Between Clusters

In order to determine the distance between two clusters, again 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, computed using a secondary distance function. We will abbreviate this secondary distance function by $dist_2$ whereas $dist_2$ 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 $dist = dist_{eucl} \equiv dist_{n,2}$. Let b_1 and b_2 be two clusters $\in B$, than we can define the following distance measures between them:

- The maximum distance between the elements of the two clusters (also called complete linkage):

$$dist_{max} = \max\{dist_2(a_1, a_2),\ a_1 \in b_1, a_2 \in b_2\}$$

- The minimum distance between the elements of the two clusters (also called single linkage):

$$dist_{min} = \min\{dist_2(a_1, a_2),\ a_1 \in b_1, a_2 \in b_2\}$$

- The mean distance between the elements of the two clusters (also called average linkage):

$$dist_{avg} = \frac{1}{|b_1| * |b_2|} \sum_{a_1 \in b_1} \sum_{a_2 \in b_2} dist_2(a_1, a_2)$$

- The increase in variance $dist_{var}$ for the cluster being merged.
- The distance of their centers:

$$dist_{cent} = dist_2(centroid(b_1),\ centroid(b_2))$$

- The distance of their nuclei computed by the nucleus function *nucleus* (see the definition of nucleus in Section 36.2):

$$dist_{nuc} = dist_2(nucleus(b_1),\ nucleus(b_2))$$

36.2 Elements Representing a Cluster

On page 573 we stated that there is not necessarily assigned an $a \in A$ to each real vector in $\mathbb{R}^n$. Thus, there also does not necessarily exist an a in the center of a cluster b. For our purposes to come later 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 find different functions $nucleus(b \in B)$ to compute such nuclei which, in turn, depend on a distance measure. We will abbreviate this distance function by $dist$ whereas $dist$ is an implicit parameter (again the default value $dist = dist_{eucl} \equiv dist_{n,2}$) which can be replaced by any of the functions named in the above subsections.

The first possible method would be to take the element which is closest to the centeroid $c = centroid(b)$ of the cluster b:

$$n \in b = nucleus_c(b) \Leftrightarrow \forall a \in b \Rightarrow dist(a, c) \geq dist(n, c) \tag{36.18}$$

Another definition is that we take the element with the lowest average distance to all other elements in the cluster.

$$n \in b = nucleus_d(b) \Leftrightarrow \forall a \in b \Rightarrow \sum_{\forall \beta \in b} dist(a, \beta) \geq \sum_{\forall \beta \in b} dist(n, \beta) \tag{36.19}$$

36.3 Clustering Algorithms

36.3.1 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 [1199] which we generalize to fit any given distance measure $dist$:

Definition 190 (Clustering Error). Therefore, 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 $dist_{eucl} \equiv dist_{n,2}$ as distance measure.

$$error_c(b) = \sum_{\forall a \in b} dist(a, centroid(b)) \tag{36.20}$$

$$error_p(B) = \sum_{\forall b \in B} error_c(b) = \sum_{\forall b \in B} \sum_{\forall a \in b} dist(a, centroid(b)) \tag{36.21}$$

Normally, this error is minimized under the premise of a fixed count of clusters $k = |B|$. Then, an optimum configuration B_{opt} is searched within the set $\mathbb{B}$ of all possible partitions of a into clusters B. This optimum B_{opt}

is defined by $e_g(B_{opt}) = \min\{e_g(B)\,\forall B \in \mathbb{B}\}$. Since testing all possible configurations B is to expensive (see Equation 36.7), finding the optimum B_{opt} is an optimization tasks itself. Here we will introduce some algorithms which approximate good B.

36.3.2 *k*-means Clustering

k-means clustering[11] [1200, 1022, 1201] partitions the data points $a \in A$ into k disjoint subsets $b \subseteq A,\ b \in B$. 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. [1202]

k-means clustering works approximately as follows [1199]:

Step 1 Select an initial partition of k clusters.
Step 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.
Step 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 $kMeansModify$.

$$B_{new} = kMeansModify(B) \Rightarrow \forall a \in b_1 \in B\ \exists b_2 \in B_{new}\ : a \in b_2 \wedge \forall a \in b_2 \in B_{new}\ \exists b_1 \in B\ : a \in b_1 \quad (36.22)$$

One example for an implementation of $kMeansModify$ is Algorithm 36.1.

We demonstrate how k-means clustering works in Algorithm 36.2. As distance measure *dist* (lines 0, 22 and 24) usually the Euclidian distance between the centroids of the clusters, $dist_{cent,eucl}$, see page 576, is used.

36.3.3 n^{th} Nearest Neighbor Clustering

The n^{th} nearest neighbor clustering algorithm creates at most k clusters where the first $k-1$ clusters contain exactly one element and the while the rest is included in the remaining cluster. The elements of the single-element clusters are those which have the shortest 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 significant 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 far-away aggregations of $\leq n$ elements will be put into the "rest elements"-cluster. For n, normally a value of $n = \sqrt{k}$ is used.

[11] `http://en.wikipedia.org/wiki/K-nearest-neighbor_estimator` [accessed 2007-07-03]

Algorithm 36.1: $B_{new} = kMeansModify_k(B)$

Input: Implicit: k the count of clusters wanted, $k \leq |A|$
Input: Implicit: $dist$ the distance measure between clusters to be used
Input: B the list of clusters b to be modified
Data: m index of the cluster $B[m]$ with the lowest error
Data: n index of the cluster $B[n]$ nearest to $B[m]$
Data: s index of the cluster $B[s]$ with the highest error
Output: the modified tuple of clusters B_{new}

```
1 begin
2   m ⟵ m : error(B[m]) = min{error(B[i]) ∀i ∈ [0, k − 1]}
3   n ⟵ n : dist(B[m], B[n]) = min{dist(B[m], B[i]) ∀i ∈ [0, k − 1] \ {m}}
4   s ⟵ s : error(B[s]) = max{error(B[i])∀i ∈ [0, k − 1] \ {m, n}}
5   B[m] ⟵ B[m] ∪ B[n]
6   B[n] ⟵ a ∈ B[s]
7   B[s] ⟵ B[s] \ B[n]
8   return B
9 end
```

n^{th} nearest neighbor clustering uses the k^{th} nearest neighbor distance function $distf_{nn,k}$ introduced in Definition 179 on page 567. The parameter k of $distf_{nn,k}$ is set to n and we apply $distf_{nn,n}$ which relies on a secondary distance measure.

Notice that Algorithm 36.4 assumes that all elements $a \in A$ are unique (i.e. there exists no two equal elements in A) which, per definition, true for all sets. In a real implementation, clustering may be performed on a list containing the same elements multiple times. Since all equal elements all have the same distance to their n^{th} neighbor, it is possible 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, we can remove all duplicates before clustering which has the drawback that we could possible obtain a set B 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, an multiple runs are performed until k clusters have been created (see Section 30.2.1 on page 439).

36.3.4 Linkage Clustering

The linkage method [115, 349] is used to create a set B of clusters b with at most k clusters. This algorithm initially creates a cluster of each single element in the set A. After that, it reduces the set of cluster B by melting together the two closest clusters iteratively. Again, the distance measure function $dist$ (see lines 0 and 11 of Algorithm 36.4) used can be any of distance measures already introduced.

Algorithm 36.2: $B = kMeansCluster_k(A)$

Input: A the set of elements a to be clustered
Input: Implicit: k the count of clusters wanted, $k \leq |A|$
Input: Implicit: $dist$ and $dist_2$ the distance measures between clusters and elements to be used
Input: Implicit: $kMeansModify$ a function that modifies the cluster set
Data: B the tuple of clusters b computed, $|B| = k$
Data: A_{cpy} a temporary copy of A used for initialization
Data: B_{old} the cluster set of the previous inner iteration
Data: B_{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 B_{old}
Data: d_{min} the minimum distance between $\{a\}$ and any cluster in B_{old}
Data: i_{min} the index of that cluster with the minimum distance in B_{old}
Output: the set of clusters b computed - all the items of the tuple B represented as set

```
1  begin
2      A_cpy ⟵ A
3      B_new ⟵ createList(min{k, |A|}, ∅)
4      i ⟵ |B_new| − 1
5      while i > 0 do
6          B_new[i] ⟵ {a ∈ A_cpy}
7          A_cpy ⟵ A_cpy \ B[i]
8          i ⟵ i − 1
9      B_new[0] ⟵ A_cpy
10     repeat
11         B ⟵ B_new
12         B_new ⟵ kMeansModify_k(B_new)
13         repeat
14             B_old ⟵ B_new
15             i ⟵ |B_new| − 1
16             while i > 0 do
17                 B_new[i] ⟵ ∅
18                 i ⟵ i − 1
19             foreach a ∈ A do
20                 i ⟵ |B_new| − 1
21                 i_min ⟵ 0
22                 d_min ⟵ dist({a}, B_old[0])
23                 while i > 0 do
24                     d ⟵ dist({a}, B_old[i])
25                     if d < d_min then
26                         d_min ⟵ d
27                         i_min ⟵ i
28                     i ⟵ i − 1
29                 B_new[i_min] ⟵ B_new[i_min] ∪ {a}
30         until B_old = B_new
31     until error_p(B) < error_p(B_new)
32     return B[i]
33 end
```

Algorithm 36.3: $B = nNearestNeighborCluster_k^n(A)$

Input: A the set of elements a to be clustered
Input: Implicit: k the count of clusters wanted ($k > 0$)
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: B the set of clusters b computed, $|B| = k$

```
1  begin
2      L ⟵ sort_a(setToList(A), s(x_1, x_2) ≡ distf_{nn,n}(x_1) − distf_{nn,n}(x_2))
3      i ⟵ min{k, |L|}
4      B ⟵ ∅
5      while i > 0 do
6          B ⟵ B ∪ {L[i]}
7          A ⟵ A \ L[i]
8          i ⟵ i − 1
9      return B ∪ {A}
10 end
```

According to the cluster distance measure $dist_2$ chosen, *linkageCluster* realizes different types of linkage clustering algorithms[12] (see Section 36.1.3 on page 576):

- If $dist_2$ denotes the maximum distance of the elements in two clusters, complete linkage clustering is performed.
- If $dist_2$ denotes the mean distance of the elements in two clusters, average linkage clustering is performed.
- If $dist_2$ denotes the minimum distance of the elements in two clusters, single linkage clustering is performed.

36.3.5 Leader Clustering

The leader clustering algorithm is a very simple one-pass method to create clusters. Basically, they begin with an empty leader list and an empty set of clusters. Step by step elements a are extracted from the set A subject to clustering. a is compared to the elements in the leader list in order to find one leader l with $dist(a, l)$ 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 $dist(a, l) < D$ and assign a to its cluster ([338], Algorithm 36.5) or by comparing a to all possible leaders and

[12] `http://en.wikipedia.org/wiki/Data_clustering#Agglomerative_hierarchical_clustering` [accessed 2007-07-03]

Algorithm 36.4: $B = linkageCluster_k(A)$

Input: A the set of elements a to be clustered
Input: Implicit: k the count of clusters wanted ($k > 0$)
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: b_1 the first cluster to investigate
Data: b_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 b_{r1}, b_{r2} found in the current iteration
Data: b_{r1} the first cluster of the nearest cluster pair
Data: b_{r2} the second cluster of the nearest cluster pair
Output: B the set of clusters b computed, $|B| = k$

```
1  begin
2      B ⟵ ∅
3      foreach a ∈ A do B ⟵ B ∪ {a}
4      while |B| > k do
5          d_min ⟵ ∞
6          b_r1 ⟵ ∅
7          b_r2 ⟵ ∅
8          foreach b_1 ∈ B do
9              foreach b_2 ∈ B do
10                 if b_1 ≠ b_2 then
11                     d ⟵ dist(b_1, b_2)
12                     if d ≤ d_min then
13                         d_min ⟵ d
14                         b_r1 ⟵ b_1
15                         b_r2 ⟵ b_2
16         B ⟵ B \ b_r1
17         B ⟵ B \ b_r2
18         B ⟵ B ∪ {b_r1 ∪ b_r2}
19     return B
20 end
```

thus finding the leader closest to a $dist(l, a) < dist(l2, a)\ \forall l2 \in leaders$ ([1203], Algorithm 36.6).

Algorithm 36.5: $B = leaderCluster_D^f(A)$

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: B the set of clusters b computed

```
1  begin
2      L ⟵ ()
3      B ⟵ ()
4      foreach a ∈ A do
5          i ⟵ |L| − 1
6          while i ≥ 0 do
7              if dist(L[i], a) ≤ D then
8                  B[i] ⟵ B[i] ∪ {a}
9                  i ⟵ −2
10             i ⟵ i − 1
11         if i ≥ −1 then
12             L ⟵ addListItem(L, a)
13             B ⟵ addListItem(B, {a})
14     return listToSet(B)
15 end
```

Algorithm 36.6: $B = leaderCluster_D^a(A)$

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: B the set of clusters b computed

```
1  begin
2      L ⟵ ()
3      B ⟵ ()
4      foreach a ∈ A do
5          i ⟵ |L| − 1
6          j ⟵
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             B[j] ⟵ B[j] ∪ {a}
11         else
12             L ⟵ addListItem(L, a)
13             B ⟵ addListItem(B, {a})
14     return listToSet(B)
15 end
```

37

Theoretical Computer Science

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.

37.1 Algorithms

In this and the following section we want to gain insight into the topic of algorithms, both in local and distributed systems. 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 computes 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 the humble author tries to earn his scientific merits, by applying global optimization techniques to distributed computing. Therefore, 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.

37.1.1 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 the meal in what sequence and how it should be heated. The commands inside the algorithms can be very

[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]

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 such a wide field that there exist numerous different, rather fuzzy definitions for the word algorithm [1204, 1205, 1206, 957, 1207]:

Definition 191 (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 192 (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 end-state. The computational complexity and efficient implementation of the algorithm are important in computing, and this depends on suitable data structures.

Definition 193 (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 194 (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.

Whereas an algorithm is a set of directions in a representation that is especially 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. But for more than sixty years [1208], huge effort is being spent to allow us to write programs in more and more comprehensible syntax. One could say that a program is basically an algorithm realized for a given computer, as illustrated in Figure 37.1. The difference between programs and algorithm hence today lies primarily in the intention.

Definition 195 (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

[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]

[5] `http://www.nist.gov/dads/HTML/algorithm.html` [accessed 2007-07-03]

[6] `http://mathworld.wolfram.com/Algorithm.html` [accessed 2007-07-03]

[7] `http://en.wikipedia.org/wiki/Computer_program` [accessed 2007-07-03]

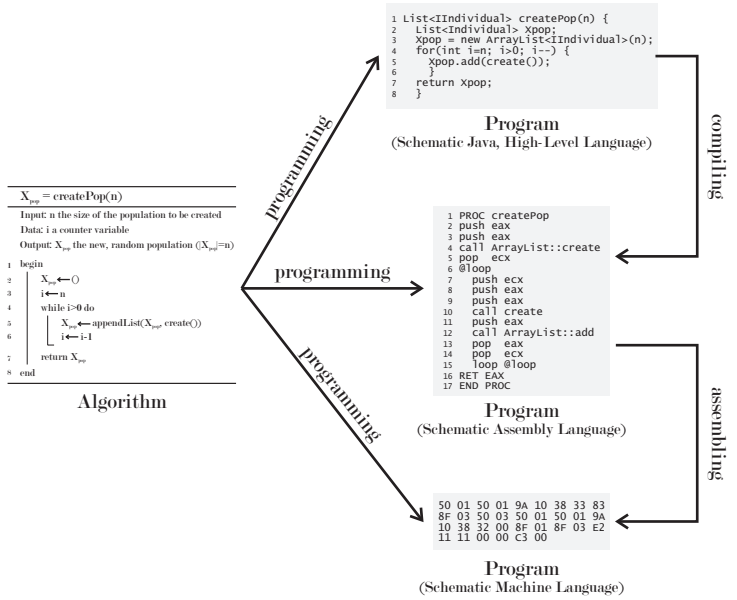


Fig. 37.1: The relation between algorithms and programs.

instructions must be present in either a form that the machine can process directly (machine code[8] [1209]), a form that can be translated (1:1) into such code (assembly language [1210], Java byte code [1087], etc.), or in a high-level programming language[9] [1211] which can be translated (n:m) into the latter using special software (compiler) [1212].

Definition 196 ((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 37.2, it consists of three parts:

- The input (IPO) is an external information or stimulus that enters the system.

[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]

Fig. 37.2: A process in the IPO model.

- The processing (IPO) 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.
- The output (IPO) comprises the results of the processing that leave the system.

37.1.2 Properties of Algorithms

Besides these definitions, algorithms all share the following properties with only few exceptions that we also will elaborate on.

Definition 197 (Abstraction). An algorithm describes the process of solving a problem on a certain level of abstraction which is determined by the elementary algorithms, elementary objects and the applied formalism. One of the most important method of abstraction is the definition and reuse of sub-algorithms.

Definition 198 (Discrete). A discrete algorithm works step-wise, i. e. is build of atomic executable instructions.

Definition 199 (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 200 (Termination). Each execute of an algorithm terminates after a finite number of steps and returns its results.

Definition 201 (Determinism). At 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 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 208 on page 591.

Definition 202 (Determined). An algorithm is determined if it always yields the same results (outputs) for the same inputs.

37.1.3 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 [1213, 1214].

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 they need for internal data [957]. We call them the space complexity and the time 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 to store intermediate values in order to produce the results. 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 [1215, 721, 1212]. 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] [1216, 1117] takes in average something about $n \log_2 n$ steps is good enough, even if the correct number is $2n \ln n \approx 1.39 n log_2 n$.

The Big-$\mathcal{O}$-family notations allow us to group functions together that rise at approximately the same speed.

Definition 203 (Big-$\mathcal{O}$ notation). The big-$\mathcal{O}$[15] notation is a mathematical notation used to describe the asymptotical upper bound of functions.

$$f(x) \in \mathcal{O}(g(x)) \Leftrightarrow \exists\, x_0, m \in \mathbb{R} : m > 0 \wedge |f(x)| \leq m|g(x)| \;\forall\, x > x_0 \quad (37.1)$$

In other words, a function $f(x)$ is in $\mathcal{O}$ 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.

Therefore, $x^3 + x^2 + x + 1 = f(x) \in \mathcal{O}(x^3)$ since for $m = 5$ and $x_0 = 2$ since $5x^3 > x^3 + x^2 + x + 1 \;\forall\, x \geq 2$.

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-$\mathcal{O}$ notation. A discussion of this topic and some examples can be found in Table 37.1.

[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]

Table 37.1: Some examples of the big-$\mathcal{O}$ notation

class	examples	description
$\mathcal{O}(1)$	$f_1(x) = 2^{222}$, $f_2(x) = sinx$	Algorithms that have constant runtime for all inputs are $\mathcal{O}(1)$.
$\mathcal{O}(\log n)$	$f_3(x) = \log x$, $f_4(x) = f_4(\frac{x}{2}) + 1; f_4(x < 1) = 0$	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.
$\mathcal{O}(n)$	$f_5(x) = 23n + 4$, $f_6(x) = \frac{n}{2}$	Algorithms of $\mathcal{O}(n)$ require to access and process their input a constant number of times. This is for example the case when searching in a linked list.
$\mathcal{O}(n \log n)$	$f_7(x) = 23x + x \log 7x$	Many sorting algorithms like quicksort and mergesort are $\mathcal{O}(n \log n)$.
$\mathcal{O}(n^2)$	$f_8(x) = 34x^2$, $f_9(x) = \sum_{i=0}^{x+3} x - 2$	Some sorting algorithms like selection sort have this complexity. For many problems, $\mathcal{O}(n^2)$-solutions are acceptable good.
$\mathcal{O}(n^i), i > 1, i \in \mathbb{R}$	$f_{10}(x) = x^5 - x^2$	The general polynomial complexity. In this group we find many algorithms that work on graphs.
$\mathcal{O}(2^n)$	$f_{11}(x) = 23 * 2^x$	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.

Definition 204 (Big-Ω notation). The big-Ω notation is a mathematical notation used to describe the asymptotical lower bound of functions.

$$f(x) \in \Omega(g(x)) \Leftrightarrow \exists\, x_0, m \in \mathbb{R} : m > 0 \wedge |f(x)| \geq m|g(x)| \;\forall\, x > x_0 \quad (37.2)$$

$$f(x) \in \Omega(g(x)) \Leftrightarrow g(x) \in \mathcal{O}(f(x)) \quad (37.3)$$

Definition 205 (Θ notation). The Θ notation is a mathematical notation used to describe both, an upper and a lower asymptotical bound of functions.

$$f(x) \in \Theta(g(x)) \Leftrightarrow f(x) \in \mathcal{O}(g(x)) \wedge f(x) \in \Omega(g(x)) \quad (37.4)$$

Definition 206 (Small-o notation). The small-o notation is a mathematical notation used to define that a function is asymptotical negligible compared to another one.

$$f(x) \in o(g(x)) \Leftrightarrow \lim_{n \to \infty} \left| \frac{f(x)}{g(x)} \right| = 0 \quad (37.5)$$

Definition 207 (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.

$$f(x) \in \omega(g(x)) \Leftrightarrow \lim_{n\to\infty} \left|\frac{f(x)}{g(x)}\right| = \infty \tag{37.6}$$

$$f(x) \in \omega(g(x)) \Leftrightarrow g(x) \in o(f(x)) \tag{37.7}$$

37.1.4 Randomized Algorithms

Deterministic algorithms[16] will always produce the same results when given the same inputs. If nothing else is stated, algorithms are considered to be deterministic.

For many problems however, deterministic algorithms are unfeasible. In global optimization (see Section 1.1.1 on page 4), the search space is often extremely large and the relation of an element's structure and its utility as solution is not directly known. Hence, the search space often cannot be partitioned wisely and an exhaustive search would be the only deterministic option left. Such an approach would take an arbitrary long time. Here, using a randomized algorithm can help.

Definition 208 (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 [1217, 1218, 1219, 1220, 1221].

There are two general classes of randomized algorithms: Las Vegas algorithms and the Monte Carlo algorithms.

Definition 209 (Las Vegas Algorithm). A Las Vegas algorithm[18] is a randomized algorithm that never returns a false result [1205, 1217, 1218, 1220].

It either returns the correct result, reports a failure, or does not return at all. If the Las Vegas algorithm returns, its outcome is deterministic (but not the algorithm itself). The termination (see Definition 200 on page 588) 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.

[16] `http://en.wikipedia.org/wiki/Deterministic_computation` [accessed 2007-07-03], see also Definition 201 on page 588

[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]

Definition 210 (Monte Carlo Algorithm). A Monte Carlo algorithm[19] is a numerical Monte Carlo method used to find solutions for mathematical problems especially suitable for high-dimensional problems. It always returns a result that may be correct or incorrect [1217, 1218, 1220].

In contrast to Las Vegas algorithms, Monte Carlo algorithms always terminate but are (partially) correctly only with a positive probability.

Definition 211 (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 [1222, 1223, 1224, 1225].

37.2 Distributed Systems and Distributed Algorithms

A distributed system is a system of autonomous computers that are connected loosely by a network and communicate by the exchange of messages in order to together perform a common functionality. *(Gero Mühl)*

Distributed algorithms [1226, 967, 966] are algorithms that are performed by multiple computers in such a distributed systems. The instances of the algorithm, running on different computers, do not share the same view on the global state. They exchange information by the means of communication. The differences in the view on the global state may result from the fact that in most cases no common, global time exists. It is also due to the fact that communication involves usually latency discussed in Section 37.2.3 on page 612 – one node sends a message to another one and it takes some time t for the message to reach that node. In the mean time, the message is regarded as *sent* by the first node and not yet known to the second one. Furthermore, networks may induce arbitrary errors into the message's content and messages can even get lost. The nodes in a distributed system are not necessarily homogeneous and thus can provide different computational power. This in turn will lead to different speed of progression of the single instances of the distributed algorithm.

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 parallel algorithms since on every node runs an instance of the same algorithm with a different view on the global state.

Distributed algorithms can provide the following advantages depending on their respective design:

1. modularity

[19] `http://en.wikipedia.org/wiki/Monte_carlo_algorithm` [accessed 2007-07-03], see also Definition 211

[20] `http://en.wikipedia.org/wiki/Monte_Carlo_method` [accessed 2007-07-03]

2. flexibility
3. resource-sharing
4. no central point of failure because of decentralization
5. scalability because of decentralization
6. robustness
7. availability
8. fault-tolerance

Distributed algorithms may come with the following drawbacks depending on their respective design:

1. high complexity
2. no common view on the global state
3. no global time
4. processes may fail
5. latency in communication (see Section 37.2.3 on page 612)
6. faults in communication (Section 37.2.3 on page 610 and Section 37.2.3)
7. problems in termination detection
8. phantom/pseudo-deadlocks
9. race conditions

The quality of a distributed algorithm can be determined by its communications complexity, i. e. how many messages need to be exchanged, or by its time complexity, i. e. how many computational steps need to be performed on the single nodes.

Definition 212 (Scalability). Scalability is a measure describing how good a system can grow or be extended for processing a higher computational load.

Definition 213 (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 214 (Bottleneck). The bottleneck[21] of a distributed application the part that has the most limiting influence on its performance. In hourglass, the *bottleneck* is the dilution in its center that limits the amount of sand that can fall down per time unit.

37.2.1 Network Topologies

Definition 215 (Network Topology). Network topology[22] is the study of arrangement and mapping of the components of a network such as connections and nodes. One may also refer to a network arrangement as a topology, i. e. you can say "The topology of our network is a star."

[21] `http://en.wikipedia.org/wiki/Bottleneck` [accessed 2007-07-03]
[22] `http://en.wikipedia.org/wiki/Network_topology` [accessed 2007-07-03]

A computer network has exactly one physical topology which is the layout of its physical components (computers, cables). On that, several virtual/overly topologies may be built. 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 topology is closely related to graph theory.

Definition 216 (Overlay Network). An overlay network[23] 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 [1227].

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.

When speaking of topology, one would normally think about a hard-wired network of computers, connected with each other through Ethernet cabling and such and such. If we take a wireless sensor network, as described in Definition 220 on page 599, 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 instead may be able to directly talk 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.

Unrestricted

In an unrestricted network topology as the one sketched in Figure 37.3a, 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 Figure 37.3b) are connected to the same transmission medium in a linear arrangement. All messages send over the medium can be regarded as broadcasts that potentially can be received by all nodes more or less simultaneously. The transmission medium has exactly two ends.

[23] `http://en.wikipedia.org/wiki/Overlay_network` [accessed 2007-07-03]

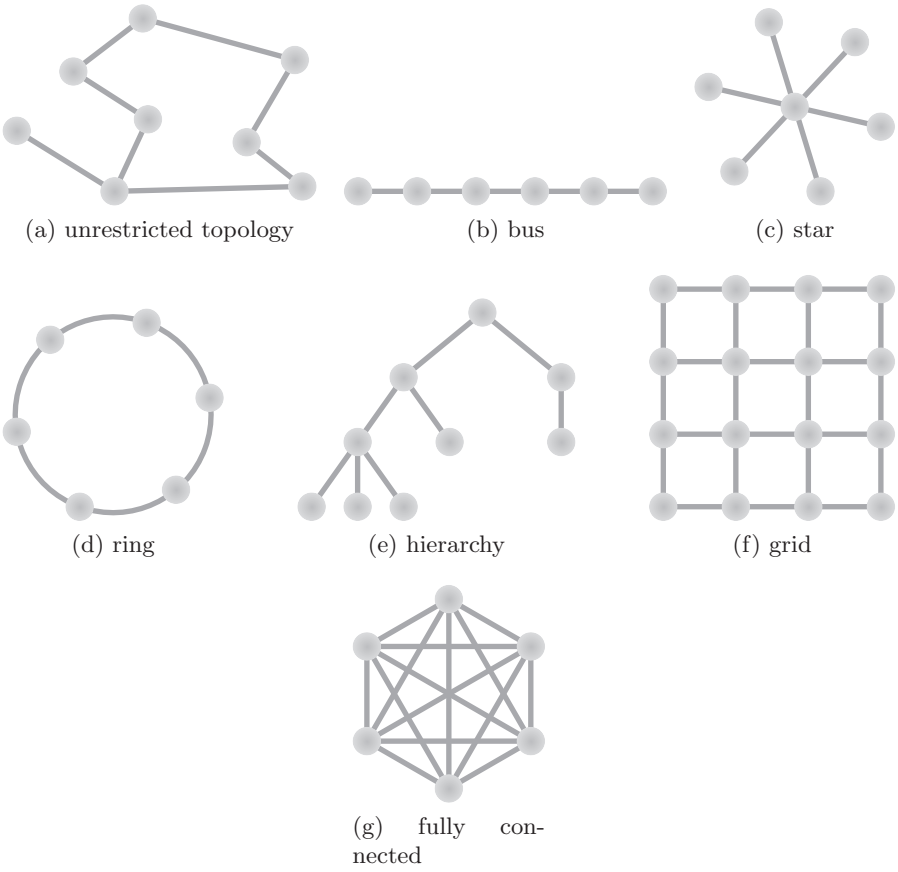


Fig. 37.3: Some simple network topologies.

Star

Figure 37.3c 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[24] or switch[25] 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 37.2.2 on the following page.

[24] `http://en.wikipedia.org/wiki/Ethernet_hub` [accessed 2007-07-03]
[25] `http://en.wikipedia.org/wiki/Ethernet_switch` [accessed 2007-07-03]

Ring

In this topology, each node is connected to exactly two other nodes in a way that no partition exists. Thus, it is like a bus where the first and the last node are connected with each other. An instance of the ring topology is illustrated in Figure 37.3d.

Hierarchy

Figure 37.3e 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 on the border of the grid, is connected with four neighbors: one to the left, one to the right, one above and one below. Figure 37.3f is an instance of such a topology.

Fully Connected

In a fully connected network, as outlined in Figure 37.3g, each node is directly connected with each other node.

37.2.2 Some Architectures of Distributes Systems

Client-Server

Definition 217 (Client-Server). Client-server[26] is a network architecture that separates two types of nodes: the client(s) and the server(s). A client[27] utilizes a service provided by the server[28]. It does so by sending a request to the server. This request contains details of the task to be carried out by the server, for example an 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.

The client-server architecture illustrated in Figure 37.4 is the most basic and the most common application logical architecture in distributed computing [1228, 1226, 1229]. It is part of almost all internet applications like:

[26] `http://en.wikipedia.org/wiki/Client_server` [accessed 2007-07-03]
[27] `http://en.wikipedia.org/wiki/Client_%28computing%29` [accessed 2007-07-03]
[28] `http://en.wikipedia.org/wiki/Server_%28computing%29` [accessed 2007-07-03]

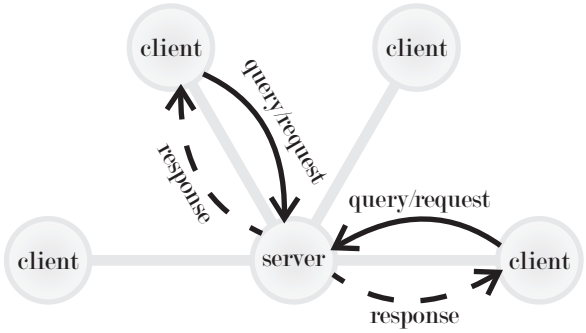


Fig. 37.4: Multiple clients connected with one server

- Websites[29] in the world wide web[30] are obtained by using the HTTP[31] protocol for communication between a web browser[32] and a web server[33].
- Application servers[34] contain the business logic of corporations. They for example support online shops[35] with an underlying business model.
- Database servers[36] 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.
- ...

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 more difficult and error-prone. The heaviest weakness of the client-server scheme is that the server represents a bottleneck (see Definition 213) and a single point of failure (see Definition 213 on page 593).

Peer-to-Peer Networks

Definition 218 (Peer-to-Peer Network). Instead of being composed of client and server nodes, a peer-to-peer[37] network consists only of equal peer nodes. A peer node functions as a server for its fellow peers by providing certain functionality and simultaneous acts as client utilizing an similar service

[29] http://en.wikipedia.org/wiki/Website [accessed 2007-07-03]
[30] http://en.wikipedia.org/wiki/Www [accessed 2007-07-03]
[31] http://en.wikipedia.org/wiki/Http [accessed 2007-07-03]
[32] http://en.wikipedia.org/wiki/Web_browser [accessed 2007-07-03]
[33] http://en.wikipedia.org/wiki/Web_server [accessed 2007-07-03]
[34] http://en.wikipedia.org/wiki/Application_server [accessed 2007-07-03]
[35] http://en.wikipedia.org/wiki/Online_shop [accessed 2007-07-03]
[36] http://en.wikipedia.org/wiki/Database_server [accessed 2007-07-03]
[37] http://en.wikipedia.org/wiki/Peer-to-peer [accessed 2007-07-03]

from its peers [1228, 1226, 1229, 1230, 1231]. Therefore, a peer node is often also called *servent*[38], a combination of the words server and client. The expression peer-to-peer is often abbreviated by P2P.

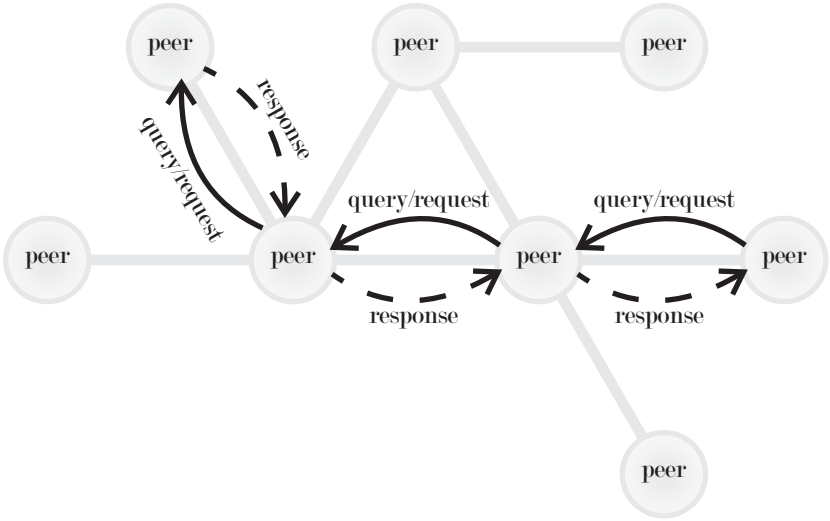


Fig. 37.5: A peer-to-peer system in an unstructured network

Peer-to-peer networks may have an arbitrary structure like the one sketched in Figure 37.5. 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 failures and can be constructed to be very robust against bottlenecks. They furthermore often are ad hoc, i. e. new peers may join the network at any time and leave it whenever they decide to. 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 clients for example register by a server which keeps track on the users online. Also, there may exist different hierarchical or non-hierarchical overlay networks.

Important peer-to-peer-based applications are

[38] `http://en.wikipedia.org/wiki/Servent` [accessed 2007-07-03]

- File and content sharing systems [1232, 1233] are the most influential and wide-spread p2p systems. Millions of users today share music, videos, documents and software over networks like Gnutella[39], Bittorrent[40], appleJuice[41] and the famous but shut-down Napster[42] network.
- Many scientific applications like Seti@home[43], Einstein@home[44], and Folding@home[45] rely on users all over the world that voluntarily provide their unused computational power. They 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.
- Many instant messaging[46] systems like talk[47] 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 chosen.
- ...

Sensor Networks

Definition 219 (Sensor Network). A sensor network[48] [1234, 1235, 1236, 1237] is a network of autonomous devices which are equipped with sensors and together measure an physical entity like temperature, sound, vibrations, pressure, or motion.

Definition 220 (Wireless Sensor Network). A wireless sensor network (WSN) [1238, 1239, 1240, 1241, 1242] is a sensor network where the single nodes are connected wireless, using techniques like wireless LAN[49], Bluetooth[50], or radio[51].

Figure 37.6 sketches the building blocks of a sensor node. Since they are autonomous devices, sensor nodes have to be equipped with some sort of energy source. For communication with other nodes, bare short range radios, Bluetooth, or wireless LAN adapters are often added. The core of a sensor

[39] http://en.wikipedia.org/wiki/Gnutella [accessed 2007-07-03]
[40] http://en.wikipedia.org/wiki/BitTorrent [accessed 2007-07-03]
[41] http://www.applejuicenet.de/ [accessed 2007-07-03]
[42] http://en.wikipedia.org/wiki/Napster [accessed 2007-07-03]
[43] http://en.wikipedia.org/wiki/Seti_at_home [accessed 2007-07-03]
[44] http://en.wikipedia.org/wiki/Einstein%40Home [accessed 2007-07-03]
[45] http://en.wikipedia.org/wiki/Folding%40home [accessed 2007-07-03]
[46] http://en.wikipedia.org/wiki/Instant_messaging [accessed 2007-07-03]
[47] http://en.wikipedia.org/wiki/Talk_%28Unix%29 [accessed 2007-07-03]
[48] http://en.wikipedia.org/wiki/Sensor_network [accessed 2007-07-03]
[49] http://en.wikipedia.org/wiki/Wireless_lan [accessed 2007-07-03]
[50] http://en.wikipedia.org/wiki/Bluetooth [accessed 2007-07-03]
[51] http://en.wikipedia.org/wiki/Radio [accessed 2007-07-03]

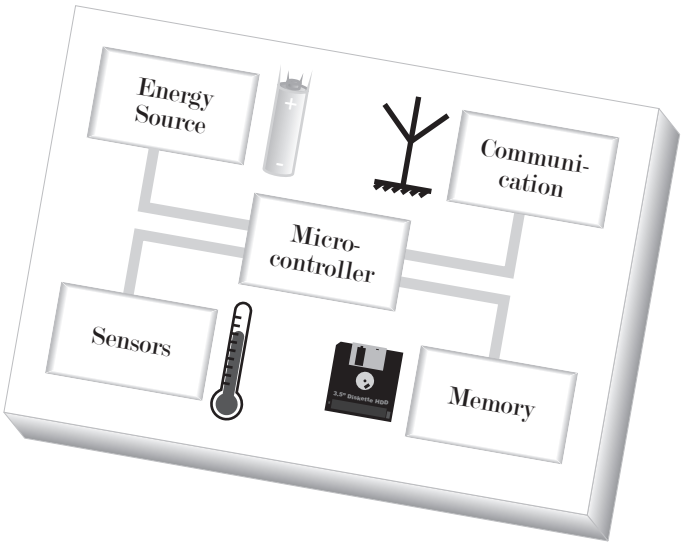


Fig. 37.6: A block diagram outlining building blocks of a sensor node.

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, a sensor node has always one or multiple sensors attached.

In order to allow free or even random deployment of sensor networks there is usually no cabling. Wireless communication and an independent power supply are hence part of many sensor node designs. Chemical batteries are used to store energy, but often power scavenging units [1243, 1244, 1245, 1246] like, for example, solar cells [1247, 1248, 1249], thermal [1250] or kinetic energy harvesters [1251, 1252, 1253] are added. The field of energy supply of sensor nodes is critical and subject to active research [1254, 1255, 1256, 1257]. 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 to matchbox dimensions. There is a strong affinity for smaller nodes. Small sensors are recognized less obviously and blend better in their environment. Since they require less raw material, they might become much cheaper then 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. It also limits the dimensions of the memory and the sensors of the node [1258].

Other important research topics are data fusion and transportation in a WSN [1259, 1260] as deployment and maintenance [1261, 1262].

Widespread sensor node architectures are:

- *BTNodes*[52] 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. Figure 37.7a shows a BTNode.
- Crossbow's *MICA2*[53] motes are multipurpose nodes. These systems are applied widely 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 Figure 37.7b.
- Scatterweb[54] provide both, a research platform (*MSB* nodes, illustrated in Figure 37.7c) and an industrial sensor network (*ScatterNodes*).
- Dust Networks[55] provide their *SmartMesh* for building wireless solutions for the global market. Their nodes provide the Time Synchronized Mesh Protocol and middle-range radio to provide the reliability of a typical WLAN in their sensor networks. Figure 37.7d shows a Dust Networks Evaluation Mote.
- ...

A small example application demonstrating the use of sensor networks is discussed in Section 20.1 on page 337.

Properties of Peer-To-Peer and Sensor Networks

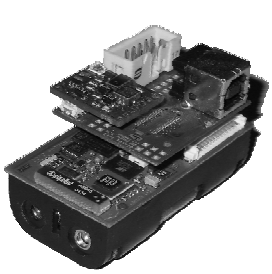
- Current peer-to-peer networks are often large-scale, with tens of thousands [1233] up to millions [1263] of users/nodes online. Although networks of thousands of sensors are a future goal, the number of nodes in sensor networks has not yet reached this extent. However, systems of several hundreds of nodes are already deployed [1264, 1265].
- 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 is solved and made transparent by routers. In sensor networks however, no such thing as dedicated hardware routers exist (since the sensors are uniform). Therefore, special

[52] `http://www.btnode.ethz.ch/` [accessed 2007-07-03]

[53] `http://www.xbow.com/Products/productdetails.aspx?sid=156` [accessed 2007-07-03]

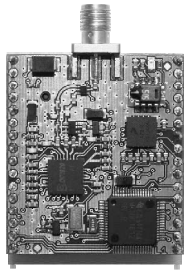
[54] `http://www.inf.fu-berlin.de/inst/ag-tech/scatterweb_net/` [accessed 2007-07-03] and `http://www.scatterweb.com/` [accessed 2007-07-03]

[55] `http://www.dustnetworks.com/` [accessed 2007-07-03]



(a) BTNode

(b) Mica2Dot



(c) MSB Mote

(d) Dust Networks Evaluation Mote

Fig. 37.7: Images of some sensor network platforms.

routing protocols [1266, 1267, 1268] are applied. Here we see a strong relation between sensor networks and peer-to-peer systems: each sensor may act as sender of a message as well as router, there is no generic hierarchy or division between senders or routers.

- 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 becomes member of the network.
- Since sensor networks utilize sleep cycles in order to reduce energy consumption, messages that are routed my arbitrarily be delayed or can get lost.
- P2P networks often represent very heterogeneous environments, consisting of computers of different architectures and operating systems.

37.2.3 Modeling Distributed Systems

In this section we will discuss how to model the features of distributed systems. Models are needed to prove the properties of distributed algorithms and are the foundation for any form simulation.

Communication Operations

We start with modeling by defining some basic communication operations to be used in the definition of distribution algorithms. In these definitions, we treat nodes and messages as objects with no further descriptions or contents.

Definition 221 (sendTo). The $sendTo(n, m)$ operation sends a message m to the node n. $sendTo$ incorporates routing if needed. $sendTo$ does not block the sender asynchronously transmits the message m according to the model parameters to n. n can obtain it with $receiveFrom$ or $receiveAny$.

Definition 222 (broadcast). The operation $broadcast(m)$ sends the message m to all nodes directly attached and reachable by the invoking node. Like $sendTo$, $broadcast$ does not block.

Definition 223 (receiveFrom). With the statement $m = receiveFrom(n)$, the invoking node waits until a message is received from the node n. The first message from n that comes in is returned in m. This operation is blocking, it will not return until a message from n has been received.

Definition 224 (receiveAny). With the statement $m = receiveAny()$, the invoking node waits until a message is received, not caring about the sender. The first message that comes in from any node is returned in m. This operation is blocking, it will not return until a message has been received.

Definition 225 (getSender). The function $getSender(m)$ returns the note from that has sent the message m. If, for example, the node n has broadcasted m with $broadcast(m)$, each node that received ($m = receiveAny()$) will be able to determine its source ($n = getSender(m)$).

Definition 226 (getReceivers). The function $getReceivers(m)$ returns the set of nodes to which the message m was sent. Each node that received a message using $receiveFrom$ or $receiveAny$ will be part of this set.

In order to allow us to make general statements about the order of messages, let us further define the global time where messages are sent and received. These times are normally not available to the nodes and just serve us as aid.

Definition 227 (getSendTime). The function $getSendTime(m)$ returns the global time where the message m was sent, i. e. handed down from the sending application to the operating system or communication middleware. This is the instant where $sendTo(x, m)$ or $broadcast(m)$ was called.

Definition 228 (getReceiveTime). The function $getReceiveTime(n, m)$ returns the global time where the node n has received the message m, i. e. the moment where the message m is handed up from the communication middleware or the operating system. This is the same instant where $receiveFrom(x)$ or $receiveAny$ have returned m. The $getReceiveTime(\varepsilon, n)$ of a message ε not received by n is positive infinite.

By the laws of logic, a message can only be received after it has been sent. If we define a model with instant communication, the send and the receive time can at least be equal.

$$\forall messages\, m : \forall\, n \in getReceivers(m) \Rightarrow \\ getReceiveTime(n, m) \geq getSendTime(m) \quad (37.8)$$

Equation 37.8 subsumes these axioms which have currently not yet been refuted by physicists, although they are trying.

Modeling Parallelism

In reality, the distributed algorithms on the nodes of a network all run in parallel in an unpredictable manner. We now need to find a model that allows us to explore this parallel behavior in a formalized fashion.

In simulations, it is complicated to model time continuous and thus, physically correct. A discretization of time simplifies many aspects of a parallel system. It can be justified by the fact that time, as a continuum, can be split into infinite small units. Analogical to our elaborations on the Poisson process in Section 35.3.2 on page 532 for Δt approaching zero, we make the assumptions that in such a time step either

1. nothing happens or
2. exactly one node of the modeled systems performs exactly one elementary action of the distributed algorithm.

With this simplification we also indirectly have sequenced in our model all elementary/atomic actions in the running system. Furthermore, if the data exchange of the distributed algorithms can be modeled as instantaneous with discrete time, we can disregard the time steps where nothing happens in our considerations.

Each distributed algorithm to be analyzed now needs to be broken up into atomic actions. How we do this has severe impact on the complexity of the analysis. Let us take Algorithm 37.1 for example, a very simple and naïve graph coloring algorithm. The graph coloring problem is defined on unrestricted graphs of arbitrary structure. The objective is to find a coloring where no node has the same color as any other node it is directly connected with. Our sample algorithm tries to solve this by first selecting a random color for the node it runs on and broadcasting this color to each of its neighbors. If

a node now receives a message (containing the color of one of its neighbors), it checks if it has the same color set. If so, it randomly selects another one and again, informs its neighbors about its choice. From the looking, we could divide this algorithm into at least eight elementary decisions or actions. Since these consist of high-level constructs like *if* or *while*, we probably would have to further split them up and transform them into jumps in some sort of pseudo-assembler that we can simulate properly. Making formal statements about such aggregations of instructions in the context of parallelism however gets more and more cumbersome the more instructions have to be taken into account.

Algorithm 37.1: *distributedGraphColoring*

Input: *colors* a set of colors to choose from
Data: *color* the color of the node
Data: *msg* a message received from another node, containing the other node's color

```
1 begin
2     color ⟵ colors[random_u(|colors|)]
3     broadcast(color)
4     while true do
5         msg = receive()
6         if msg = color then
7             while msg = color do color ⟵ colors[random_u(|colors|)]
8             broadcast(color)
9 end
```

Reasoning about the features of our sample algorithm becomes way simpler when recognizing that we only need to decompose it into two pieces. The initialization part (line 2 and 3) and the update part (lines 5 to 8). We can do this because only the sequence of those algorithm steps where a node receives or sends a message do matter in global sequence. Everything in between is local to the single node and has no influence on the progress of the algorithm instances on the other nodes. Therefore, we can assume the message sending/receiving actions together with all the actions in between them as elementary and atomic from the global standpoint.

Reducing the atomic pieces of a distributed algorithm also reduces the number of relations to be taken into consideration and thus, eases its analysis significantly. The same goes for its simulation, since it is much more cumbersome to build interpreters for assembler-style elementary instructions than simple simulating a few statements in form of compound, high-level expressions.

After splitting up the algorithms into their elementary building blocks, we can classify parallelism models according to their degree of concurrency.

Synchronous Models

In synchronously running distributed algorithms, all nodes proceed with exactly the same speed. We model this by executing elementary algorithm steps according to the round robin principle[56]. From the view of a single node, all its neighbors are always as fast as it is itself. This becomes clear when visualizing that the time where a node "does nothing", where none of its atomic instructions is executed, does not exist for the node. From the global perspective, runtime is assigned to the nodes as illustrated in Figure 37.8.

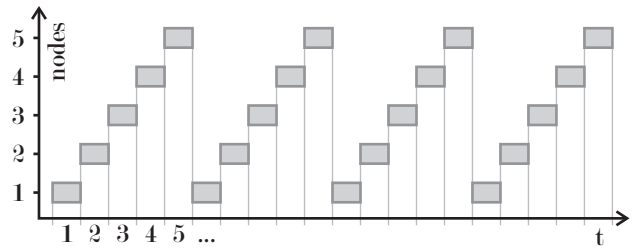


Fig. 37.8: Synchronous parallelism in a model of a network of five nodes.

Asynchronous Models

In some cases it may be sufficient to regard a distributed system as synchronous – most real networks however are not. They consist of computers that run highly asynchronous, maybe even at different speeds that can change over time. Such behavior can be modeled in a surprisingly simple fashion. Instead of assigning the runtime to the nodes using to the round robin principle, in each time step one node is picked randomly according to the uniform distribution (see Section 35.3.1 on page 527). Thus, in each time step each node has exactly the same probability of executing one action. In average over infinite time, all nodes will execute the same amount of algorithm steps. For a given time period however, it is possible that one node can execute five steps while another one just proceeds by two, as outlined in Figure 37.9. It is obvious that in this time period, node one progresses with more than double the speed than node four. On the other hand, during the following twenty time units, this ratio may as well be exactly reverted. We hence can model different and changing execution speeds. All nodes still have the same average speed, as it would be in a homogeneous network of computers of the same type. In case this is not wanted, we simple need to adjust the probability with which the nodes are picked and deviate from the uniform distribution.

[56] `http://en.wikipedia.org/wiki/Round-robin_scheduling` [accessed 2007-07-03]

The single nodes start at different points in time with their execution, one node can be done with its work before another one gets assigned that first time step. If the distributed algorithms furthermore do not contain infinite loops or similar constructs and thus, finish at some point of time, this will result in a dynamic network topology where nodes join and leave the computation in an arbitrary manner.

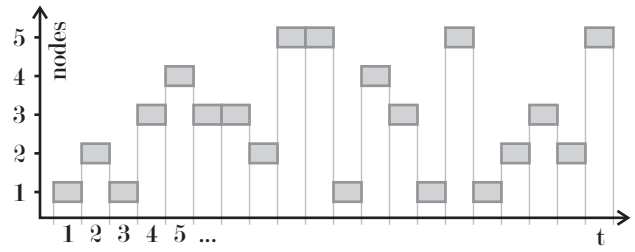


Fig. 37.9: Asynchronous parallelism in a model of a network of five nodes.

Modeling Topology

If we want to find out about the features of a distributed algorithm, we need to define a certain topology in which the nodes are arranged. There are algorithms that only work on specific topologies while others may work on any (partition-free) network layout. In most cases however, the network topology has severe impact on the convergence/progression speed of the distributed algorithms. In terms of modeling or simulation, we usually only pay attention to the top-level overlay topology, if there is any, and to the network layout otherwise.

Static Topology

If our topology is static and thus, does not change by time, we can for example commit our model to one specific topology mentioned in Section 37.2.1 on page 593. Under the premise that the algorithms that we want to examine do not depend on a special arrangement of the nodes, we basically have three choices for a proper topological model.

- If the algorithms are topology-independent, we can define the topology of our model to be unrestricted (see Section 37.2.1). The strength of the unrestricted topology is that properties of a distributed algorithm observed will be valid for all other topologies. We do not assume that all nodes are able to communicate directly with each other. Therefore, this model is optimal to study algorithms that are used to spread information over a

network since it allows us to study the data dissemination behavior. On the other hand, this topology requires some inherent routing of information-spreading technique in the algorithms. For protocols that do not primarily deal with this issue, the unspecified topology is not suitable.

- For such algorithms, a fully connected network layout (see Section 37.2.1 on page 596) should be used. Here, we can obtain information about their behavior without interferences possible induced by additional routing functionality.
- If we need to perform really fast simulations, unspecified topologies have the drawback of being arbitrary graph structures which complicates the computations performed by the simulation environment. Therefore, it should be replaced by a two-dimensional grid topology discussed Section 37.2.1 on page 596. Here we can arrange the simulated nodes in regular lattice which is simpler to access and to simulate.

Dynamic Topology

A dynamic topology like it could occur in a peer-to-peer network (see Section 37.2.2 on page 597) is implicitly be modeled by parallelism (see Section 37.2.3). In Figure 37.10 six nodes, granted the same runtime, are modeled/simulated. Since the time unites are assigned randomly to them, node 1 has spent up all his ticks before nodes 4 to 6 did even start. The network hence first consisted only of the node 1, then later of the nodes 1 to 3 and transcends over consisting of the nodes 4 to 6 to its final state where only the node 6 remains. Dynamic topologies should, in general, be regarded as unrestricted

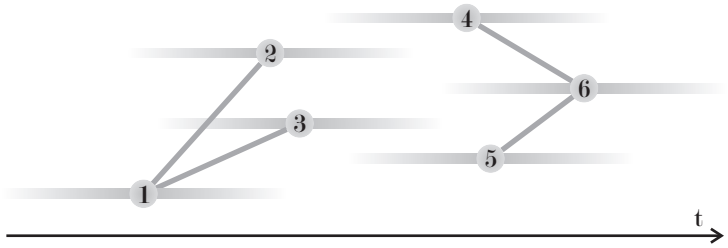


Fig. 37.10: Dynamic topology due to overlapping active times of nodes.

topologies. For the ease of simulation implementation, we can however define a grid topology as basis where nodes are considered as switched on and off according to the random consumption of their runtime.

Static and Dynamic Partitions

Definition 229 (Static Partition). A static partition in a network $\mathcal{N} = (p_1, p_2, \ldots)$ containing the nodes p_i exists if we can divide the network in at

least two disjoint subsets $\mathcal{N}_1 \cup \mathcal{N}_2 = \mathcal{N}$ in way that there does not exist a connection between any node in $\mathcal{N}_1$ and a node in $\mathcal{N}_2$.

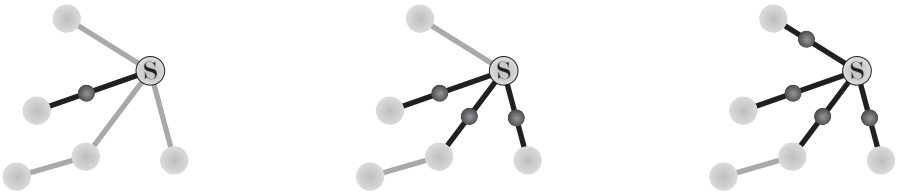
If a network is statically partitioned, there exist at least two nodes that, even with routing over arbitrary many stations, will never be able to exchange data in any way. Such configurations are not interesting for simulations and models, since distributed algorithms cannot work on them properly. The modeler or simulation designer thus has to take care that network partitions do not occur.

Communication

Every distributed algorithm requires some form of communication.

Number of Receivers

In a network, a node may be connected to n other nodes. Communication forms can be divided into three categories according to how many receivers are reached with one transmission.



(a) Unicast (b) Multicast (c) Broadcast

Fig. 37.11: The three different message transmission types.

Definition 230 (Unicast). Unicast[57] means sending a message to one single recipient only, as illustrated in Figure 37.11a.

Definition 231 (Multicast). Figure 37.11b outlines the multicast[58] transmission scheme. A multicast message is sent to a subset of m recipients (in a group of n possible receivers), where $1 \leq m \leq n$.

Definition 232 (Broadcast). To broadcast[59] a message means to send it to all n possible recipients at once. Figure 37.11c sketches a broadcast message that is received by all destinations that are directly linked with the sender.

[57] http://en.wikipedia.org/wiki/Unicast [accessed 2007-07-03]
[58] http://en.wikipedia.org/wiki/Multicast [accessed 2007-07-03]
[59] http://en.wikipedia.org/wiki/Broadcasting_%28networks%29 [accessed 2007-07-03]

If the group of possible recipients only has $n \leq 1$ members, there is no distinctions between unicast, multicast, or broadcast. In general, we could regard broadcasts as a special case of multicast that is send to all possible destinations. Furthermore, one could model unicasts as multicasts with only one recipient.

Reliability of Transmissions

The reliability of transmissions is another aspect that should be thought of when modeling the environment of a distributed algorithm. We can distinct between two different types of faults in communication, message loss and message modification. In addition, there is the message delay/latency which can also take on the characteristics of an error.

When modeling a network in order to derive properties of a distributed algorithm, we have to decide which of these faults are relevant and which are not. If we, for example, examine an algorithm that runs in an environment where communication is secured by underlying protocol levels, message loss may be irrelevant. Also, message modification can be omitted from our model if our algorithm is not security-related and runs on top of a protocol that uses checksums and such and such to secure data integrity later in the real implementation.

If our model bases on broadcast or multicast, we also have to decide if possible faults occur per message or per packet on a connection. As already discussed before, we can think of a network as a graph where each node is represented by a vertex. A node is connected to all n nodes that it can directly broadcast to by an edge. A broadcast message would be split up into n identical packets, each traveling on one connection. Here we should distinct if we model a hard-wired network, where errors would occur packet-wise since they are usually bound to a single connection, or if we have a wireless network where an error would probably influence the whole transmission. The following considerations may be applied in either way.

Message Loss Messages may vanish on the way to their recipient.

- This is most often caused by a collision[60] with another transmission on the same medium.
- The sending node may not be directly connected to the receiving one. Then, its messages have to be routed over some intermediate nodes. If one of these stations gets congested and its in or output buffers are overflowing, it may discard the message.
- Messages can also be caused to disappear by a third party that intrudes the communication channel with malicious intend of deprive the receiver of information.

[60] `http://en.wikipedia.org/wiki/Collision_%28telecommunications%29` [accessed 2007-07-03]

- A connection between two nodes may be broken and all messages on this connection will be lost.

The loss of a message can be detected on the receiving side if the communication source sending a strictly increasing sequence number along with the payload. Well known protocols like TCP[61] utilize this mechanism.

If we need to model this mechanism, we can do so by simple determining a probability $0 \leq \varepsilon \leq 1$ with which transmissions fail. Then, whenever a message is sent by node, we use draw a random number e uniformly distributed in $[0, 1)$. If $e < \varepsilon$ then the transmission is lost, otherwise it will get through. This is a rather crude approach but has the advantage that it can easily be understood and analyzed using mathematical methods.

In order to add the possibility of connection break down, we can also draw such a random number at each time step for each connection in order to determine if it will fail or not. A failing connection then will stay broken for a time determined using the exponential distribution.

Message Modification There are two possible causes why messages could be modified in transmission.

- Accidental modification due to physical interference like power surges or other signals.
- Again, a third party may be responsible for the change of the message's content. The intention could be to deceive the receiver.

If we can assume that the distributed algorithms can rely on an underlying protocol with error detection capabilities, like the IP[62] that therefore uses a checksum[63], message modification does not need to be modeled.

Otherwise, we can regard a message m as a sequence of n bits $m = (b_1, b_2, \ldots, b_n)$.

Definition 233 (Error Burst). An error burst[64] is a very common error scheme in telecommunications. It denotes a continuous sequence of symbols (in our case, bits) over a data transmission channel or part of a message such that this sequence contains not a single correct (i. e. unaltered) symbol.

We can simulate such errors with Algorithm 37.2 by first defining a fault probability $0 \leq \varepsilon \leq 1$. For each message we draw a uniformly distributed random number from $[0, 1)$. If this number is smaller than ε, the message will be modified. Therefore we first randomly determine the length of the error burst in line 3 and then its position. In Algorithm 37.2, we simple draw the error burst length from a normally distributed random variable with mean μ

[61] `http://en.wikipedia.org/wiki/Transmission_Control_Protocol` [accessed 2007-07-03]

[62] `http://en.wikipedia.org/wiki/Internet_Protocol` [accessed 2007-07-03]

[63] `http://en.wikipedia.org/wiki/Checksum` [accessed 2007-07-03]

[64] `http://en.wikipedia.org/wiki/Error_burst` [accessed 2007-07-03]

and variance σ^2 which have to be chosen carefully. In order to limit the burst length to a positive, natural number smaller or equal to the message length, we apply the cut-off function $random_l$[65] and round down.

Algorithm 37.2: $m_\varepsilon = errorBurst(m)$

Input: m the original message
Data: l the random length of the error burst
Data: p the position where the error burst occurs
Output: m_e the message after passing the transmission channel

```
1  begin
2      m_ε ⟵ m if random_u() < ε then
3          l = ⌊random_l(random_n(μ, σ²), 1, |m| + 1))⌋
4          p = ⌊random_u(0, |m| − l + 1)⌋ + l − 1
5          while l > 0 do
6              m_ε[p] ⟵ ¬m[p]
7              l ⟵ l − 1
8              p ⟵ p − 1
9      return m_ε
10 end
```

Message Latency

Definition 234 (Latency). Latency[66] is the time difference between the moment where something is initiated and the moment of its effects becoming observable [1269].

In a real distributed application, messages are constructed and sent by a software process running on a node. We consider the moment when they are passed down to the operating system or to the middleware as the moment when they are sent. From there, however, they have to be passed to the communication hardware and from there they are transmitted over a medium. Now physical effects will delay the message from arriving instantly at the receiver side – the speed of light still cannot efficiently surpassed. This delay induced by physical laws is however normally negligible. Yet it is observable in satellite communication, for example when a host of a news show talks with a reporter on the other side of the globe.

If the node sending a message is not directly connected to the recipient, the message will be routed over some intermediate nodes. Each of these nodes needs to analyze the message's destination in order to find out where to send

[65] see Section 35.7.3 on page 566

[66] `http://en.wikipedia.org/wiki/Network_Latency` [accessed 2007-07-03], `http://en.wikipedia.org/wiki/Lag` [accessed 2007-07-03]

it next needing some processing time. Furthermore, congestion may lead to additional delay. After the message arrives on its destination, it has again to be dealt with by the hardware, operating system, and middleware before being passed to the application, which marks the arrival time.

The time a message needs to travel will of course also depend on its size. We may however assume that message size is negligible in our model by simple defining that large messages will be broken down into chunks of equal size before transmission.

The most problematic fact is that there is no upper limit for message latency, i. e. we cannot determine if a message was lost or is still delayed in transmission.

Since the latency in a real network depends on so many factors [1269], it is hard to simple approximate it with a simple probability distribution. In principle, we can choose between three candidates: the exponential, the normal, and the uniform distribution. Additionally, latency can also be modeled deterministically as a function of the distance between the sender and the receiver [1270].

- Many experiments indicate that network latency is loosely exponentially distributed. On page three in [1271], the latency/probability diagram (Figure 2) remotely resembles the exponential distribution. It is therefore often used in models and simulations [1272]. Here, a positive cut-off should be applied to prevent messages from traveling with zero delay (see Section 35.7.3 on page 566).
- Since the exact form of the distribution of latency may vary from application to application, it also makes sense to approximate it with a normal distribution. A normally distributed random number can take on zero or negative values with a non-zero probability. Therefore, again a cut-off should be applied in the model or simulation according to Section 35.7.3.
- Drawing uniformly distributed random numbers is the most crude and simple way to determine message latency. The idea of choosing it is that in most models, the exact distribution of the message delays plays no role. The only thing that counts is that messages can be delayed in a way that allows them to outpace each other or to allow some nodes to receive them early than others. These are the most critical scenarios for distributed algorithms and we can create them with the uniform distribution as well as with every other one.
- Another attractive approach to modeling latency is to make it a function of the distance between the sender and the receiver [1270]. If we consider a routed network and assume a constant delay per router passed by the message, this method is very elegant. Additionally, one could allow a bit of deviation by adding a small random number to each deterministically computed delay.

Reliability of Nodes

So far we have considered errors that may happen during message transmission. Now we want to discuss the errors that may happen before – the possible faults on the single nodes. Especially important in the context of distributed systems are Byzantine faults.

Definition 235 (Byzantine Fault). Byzantine faults[67] are errors that occur during the execution of a distributed algorithm in a network due to one or multiple nodes deviating from the prescribed flow of the algorithm [1273, 1274, 1275].

Constructing systems that are able to deal with such problems is subject to research since the 1980s [1276, 1277, 1278, 1279, 1280, 1281, 1282, 1283].

Byzantine faults cover node crashes, incorrect execution of algorithm steps, and the execution of wrong steps as well as nodes that *intentionally* send incorrect and misleading messages. Including Byzantine faults in the model of a distributed system in principle only needed in the last case, since the effects of the others can as well be reached by modeling communication faults.

37.3 Grammars and Languages

Languages are used for communication between higher animals [68]. They also 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 236 (Syntax). The syntax[69] 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 237 (Semantic). The semantic[70] 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.

37.3.1 Syntax and Formal Languages

Let us now take a closer look on the syntax of formal languages [1284, 1285].

[67] `http://en.wikipedia.org/wiki/Byzantine_fault_tolerance` [accessed 2007-07-03]
[68] `http://en.wikipedia.org/wiki/Language` [accessed 2007-07-04]
[69] `http://en.wikipedia.org/wiki/Syntax` [accessed 2007-07-03]
[70] `http://en.wikipedia.org/wiki/Semantics` [accessed 2007-07-03]

Definition 238 (Alphabet). A finite set Σ of symbols (characters) $\alpha \in \Sigma$ with a total order (see Section 34.6.1 on page 509) defined on it is called an alphabet.

Definition 239 (Character String). A character string[71] (or word) over 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 Σ if and only if it can be created using the two rules above.

Definition 240 (Concatenation). The concatenation[72] $\alpha \circ \beta$ 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 $\alpha \circ \beta = \alpha_1\alpha_2\alpha_3 \ldots \alpha_n\beta_1\beta_2\beta_3 \ldots \beta_m$ which begins with α immediately followed (and ended by) β.

The set of all strings of length l over 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[73] (or Kleene closure).

Definition 241 (Lexeme). A lexeme[74] is the lowest level of syntactical unit of a language [1284]. 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 string of characters over some alphabet.

Definition 242 (Language). A language L over the alphabet Σ is a subset of Σ^* [1285]. L is the set of all sentences over an alphabet Σ that are valid according to its rules in syntax (the grammar) [1286].

When describing the formal syntax of a language, there are two possible approaches:

1. Recognizers that determine the structure of a sentence and can decide if it belongs to the language or not. Recognizers are, for instance, used in compilers [1287].
2. A generative grammar can build all sentences of a language.

[71] http://en.wikipedia.org/wiki/Character_string [accessed 2007-07-03]
[72] http://en.wikipedia.org/wiki/Concatenation [accessed 2007-07-10]
[73] http://en.wikipedia.org/wiki/Kleene_star [accessed 2007-07-03]
[74] http://en.wikipedia.org/wiki/Lexeme [accessed 2007-07-03]

37.3.2 Generative Grammars

A generative grammar G of a language L is able to construct every single sentence in 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 37.1: A simple generative grammar.

Here we have four productions, the terminal symbols `Alice`, `Bob`, `writes`, `reads`, `cipher-text`, and `plain-text`, and five non-terminal symbols (`sentence`, `subject`, `verb`, and `object`).

Definition 243 (Formal Grammar). A formal grammar[75] $G = (N, \Sigma, P, S)$ is a 4-tuple consisting of:

- a finite set N of non-terminal symbols (variables),
- the alphabet Σ, a finite set of terminal symbols,
- a finite set P of productions (also called rules), and
- at least one start symbol $S \in N$ which belongs to the set of non-terminal symbols N.

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 Noam Chomsky in 1956 [1288, 1289, 1290] 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.

In Table 37.2 illustrates the Chomsky hierarchy, V is the set of all terminal and non-terminal symbols ($V = N \cup \Sigma$) and V^* is its Kleene closure.

37.3.3 Derivation Trees

A derivation tree[76] 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

[75] `http://en.wikipedia.org/wiki/Formal_grammar` [accessed 2007-07-03]

[76] `http://en.wikipedia.org/wiki/Context-free_grammar#Derivations_and_syntax_trees` [accessed 2007-07-16]

Table 37.2: The Chomsky Hierarchy

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

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 (Σ). Each edge constitutes one expansion according to a production of the grammar.

Assume an example grammar $G = (N, \Sigma, P, S)$ with $N = \{\texttt{T}\}$, $\Sigma = \{\texttt{1}, \texttt{+}, \texttt{a}\}$, $S = \texttt{T}$, and the productions P defined the below.

```
1 T ⟶ T+T
2 T ⟶ 1
3 T ⟶ a
```

Listing 37.2: An example context-free generative grammar G.

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 37.3: An example expansion of G.

Figure 37.12 illustrates the derivation tree that belongs to this example expansion of the example grammar G.

37.3.4 Backus-Naur Form

The Backus-Naur (BNF) form[77] is a metasyntax used to express context-free grammars [1291, 1292]. Such Chomsky Type-2 grammars are the theoretical basis of most common programming languages and data formats, like for example C and XML[78]. It allows specifying production rules in simple, human and machine-understandable manner.

[77] http://en.wikipedia.org/wiki/Backus%E2%80%93Naur_form [accessed 2007-07-03]
[78] http://www.w3.org/TR/2006/REC-xml-20060816/ [accessed 2007-07-03]

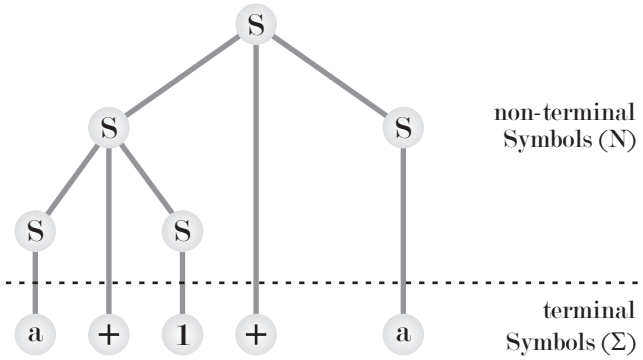


Fig. 37.12: The derivation of the example expansion of the grammar G.

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 "`|`".

Starting with `S`, the example below allows us to generate natural numbers $\mathbb{N}$. A `nonZero` is either 1,2,..., or 9 and a normal `number` may also be zero. A natural number is either a `nonZero` or a natural number with a `number` at the end. Notice that expanding `nonZero` will always lead to the first digit always being a non-zero digit since a fully expanded rule cannot contain any variables (non-terminal symbols). As start symbol `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 37.4: Natural numbers – a small BNF example.

37.3.5 Extended Backus-Naur Form

The extended Backus-Naur form[79] is an extension of the BNF metasyntax that provides additional operators and simplifications [1293, 1294, 1285].

Unlike in 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 those may either occur zero or one time

[79] `http://en.wikipedia.org/wiki/Ebnf` [accessed 2007-07-03]

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 $\mathbb{Z}$ 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 37.5: Integer numbers – a small EBNF example.

The ISO norm ISO/IEC 14977 [1293] for EBNF defines additional extension mechanisms which we will not discuss here.

37.3.6 Attribute Grammar

An attribute grammar[80] (AG) is a context-free grammar enriched with attributes, rules, and conditions for these attributes [1295, 1296, 1297, 1298]. With attributes attached to non-terminal symbols, it becomes possible to provide context-sensitive information. They are 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 243 on page 616,
2. a finite set of attributes A where each attribute $a \in A$ has a set of possible values $a = \{a_1, a_2, \ldots, a_n\}$, 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 $T(X) \subseteq A(X)$. A synthesized attribute gets its value from the attributes attached to the children of the symbol it is assigned to. Inherited attributes get their value from the parent or siblings of the symbols they belong to. The start symbol $S \in N$ and the terminal symbols Σ do not have inherited attributes ($T(S) = \emptyset, \forall\ \sigma \in \Sigma \Rightarrow T(\sigma) = \emptyset$). In Knut's original definition [1295], this was the other way round but the here discussed form has prevailed [1298].

[80] `http://en.wikipedia.org/wiki/Attribute_grammar` [accessed 2007-07-03]

A good example for synthesized attributes is given in [1299] 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.

Let us consider a simple grammar for integer mathematics with the two expressions + and ∗.

```
1 E ::= F        "+" E |
2       F
3 F ::= integer "*" F |
4       integer
```

Listing 37.6: 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.

```
1 Production                Rule
2 E ::= F        "+" E |    E.val = F.val        + E2.val
3       F                   E.val = F.val
4 F ::= integer "*" F |     F.val = integer.val * F2.val
5       integer             F.val = integer.val
```

Listing 37.7: A small example for attribute grammars.

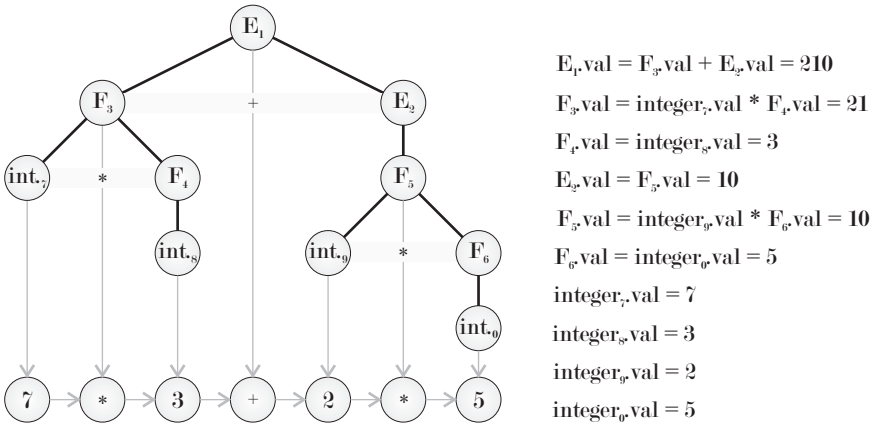


Fig. 37.13: An instantiation of the grammar from listing 37.7.

Figure 37.13 illustrates a sentence of the simple attribute grammar from listing 37.7. The non-terminal symbols are sometimes annotated with subscript numbers (like $\texttt{E}_2$) which have no meaning and only serve for clearity. While this listing 37.7 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 4.5.6 on page 169.

L-Attributed Grammars

L-attributed grammars[81] 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 on page 141). Such grammars are the foundations for many programming languages and allow convenient top-down parsing[82].

S-Attributed Grammars

An attribute grammar is called S-attributed[83] if it allows only synthesized attributes [1300]. Because of this restriction, such grammars can be parsed top-down as well as directly bottom-up[84] and are supported by various tools like Bison[85] and Flex[86].

37.3.7 Extended Attribute Grammars

Extended attribute grammars developed by Watt and Madsen (EAGs) [1301, 1302, 662] are a form of attribute grammars where the semantic (attribute-concerning) rules are no longer separated from the syntax productions. Instead, both are combined into 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 ...>
```

While $n \in N$ is a non-terminal symbol and a, b, and c, are values of attributes α, β, and γ represented as expressions over their respective attribute value domain. In an extended attribute grammar, we can define a set of inherited attributes $I(n)$ and a set of synthesized attributes $T(n)$ for each non-terminal symbol n. In the initial blueprint, $\updownarrow$ therefore has to be replaced with

[81] http://en.wikipedia.org/wiki/L-attributed_grammar [accessed 2007-07-04]
[82] http://en.wikipedia.org/wiki/Top-down_parsing [accessed 2007-07-04]
[83] http://en.wikipedia.org/wiki/S-attributed_grammar [accessed 2007-07-04]
[84] http://en.wikipedia.org/wiki/Bottom-up_parsing [accessed 2007-07-04]
[85] http://en.wikipedia.org/wiki/GNU_Bison [accessed 2007-07-04]
[86] http://en.wikipedia.org/wiki/Flex_lexical_analyser [accessed 2007-07-04]

either ↓ which means that the following attribute is inherited ($\downarrow a \Leftrightarrow \alpha \in I(n)$) or ↑ denoting a synthesized attribute ($\uparrow a \Leftrightarrow \alpha \in T(n.parent)$). 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 using a simple example borrowed from [662]. Assume the grammar $G_1 = (N, \Sigma, P, S)$ with the non-terminal symbols $N = \{S, X, Y, Z\}$, the alphabet $\Sigma = \{x, y, z, \varepsilon\}$, productions P as defined below and the start symbol S. Additionally, X,Y, and Z are equipped with one synthesized attribute $v \in \mathbb{N}_0$.

```
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 37.8: 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 occurs in 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 37.8 thus defines sentences in the form $\mathtt{x}^n\mathtt{y}^n\mathtt{z}^n$.

```
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 37.9: A typical expansion of G_1.

Another example for extended attribute grammars, again borrowed from [662], are the binary numbers. We can define a grammar $G_2 = (N, \Sigma, P, S)$ for all binary numbers where 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 $N = \{S, T, B\}$ and only two terminal symbols $\Sigma = \{0, 1\}$. 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 37.10: An extended attribute grammar G_2 for binary numbers.

Figure 37.14 illustrates one possible expansion of the start symbol 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.

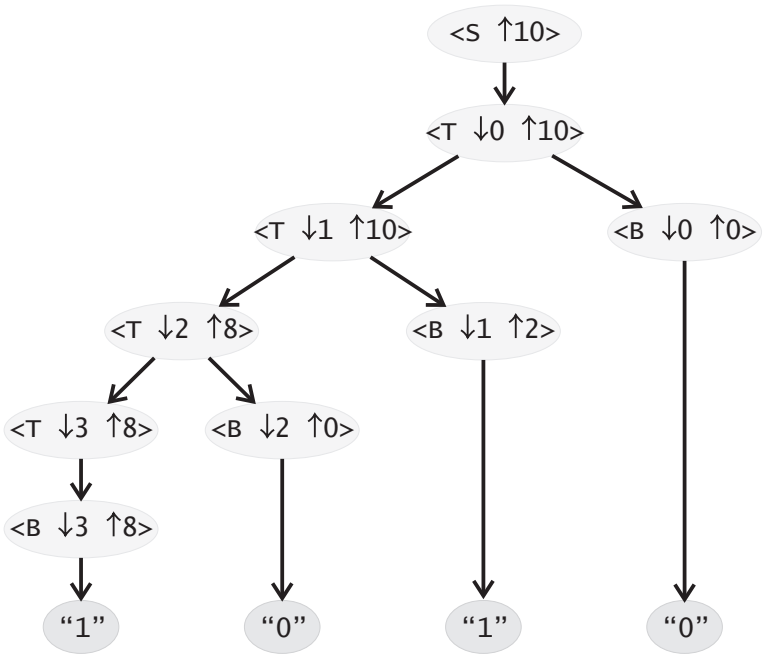


Fig. 37.14: One possible expansion of the example grammar G_2.

Extended attribute grammars are sufficient to specify the syntax and semantics of many programming languages [1303].

37.3.8 Adaptable Grammar

Definition 244 (Adaptable Grammar). An adaptable grammar[87] $G = (N, \Sigma, P, S)$ is a formal grammar 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 [662].

[87] See `http://en.wikipedia.org/wiki/Adaptive_grammar` [accessed 2007-07-13], but notice that they call it "adaptive grammars" and even refer to John Shut's masters thesis [662] where they are called "adaptable grammars".

Shutt furthermore discusses recursive adaptable grammars (RAG) which are a turning powerful formalism but yet retain the elegance of context-free grammars.

37.3.9 Christiansen Grammars

Christiansen introduces an adaptable grammar model that combines extended attribute grammars with the ability to adapt according to Definition 244 [665, 1304, 1305].

Unfortunately, Christiansen calls his adaptable attribute grammars "generative grammars" [1306, 1307] which has already another meaning (see Section 37.3.2 on page 616). We therefore resort to the term "Christiansen Grammars" coined by Shutt [662] from whom we again will borrow the examples. As described in [665, 1305], a Christiansen grammar is an extended attribute grammar where the first attribute of each non-terminal symbol $n \in N$ is inherited and a 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 ↕a ...>Z ::= XYZ` (with $\texttt{X}, \texttt{Y}, \texttt{Z} \in V$ and $n \in N$) hence only holds if `<n ↓g ↕a ...>::= Y` according to the grammar attribute `g`.

Let us start with a simple example grammar $G_1 = (N, \Sigma, P, S)$ 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 37.11: 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` which 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 $G_2 = (N, \Sigma, P, S)$ 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
```

```
<decl-list ↓g0 ↑g2>   ::= <decl ↓g0 ↑g2><decl-list ↓g1 ↑g2>
<decl ↓g ↑g+new-rule> ::= "int" <alpha-list ↓g ↑w> ";"
            where new-rule is <id ↓h> ::= w
<stmnt-list ↓g>       ::= ε
<stmnt-list ↓g>       ::= <stmnt ↓g><stmnt-lst ↓g>
<stmnt ↓g>            ::= <id ↓g> "=" <id ↓g> ";"
```

Listing 37.12: 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 grammars in the context of genetic programming can be found in listing 4.7.

37.3.10 Tree-Adjoining Grammar

Tree-adjoining grammars[88] (TAG, also called tree-adjunct grammars) are another method for defining formal grammars developed by Aravind Joshi [1308, 1309, 1310]. Different than 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 leaf to terminal symbols.

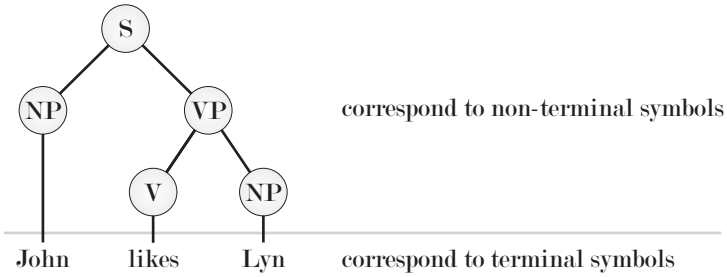


Fig. 37.15: An example TAG tree.

The simple TAG tree illustrated in Figure 37.15 is borrowed from [1310] as well as some of the following examples.

The tree structure of tree-adjoining grammar has one striking advantage compared to the flat rules in context-free grammars: an increased *domain of locality* [1309]. If we process for example an EBNF rule, we can only expand the non-terminal symbols at our current "level". Below we show that a text in an EBNF grammar similar to the one of Figure 37.15 could be resolved step

[88] `http://en.wikipedia.org/wiki/Tree-adjoining_grammar` [accessed 2007-07-03]

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 37.13: Another simple context-free grammar.

The extended domain of locality () 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 37.16 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 burried 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 37.17 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 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 245 (Lexicalized Tree-adjoining Grammar). A lexicalized tree-adjoining grammar (LTAG) is a tree-adjoining grammar where each el-

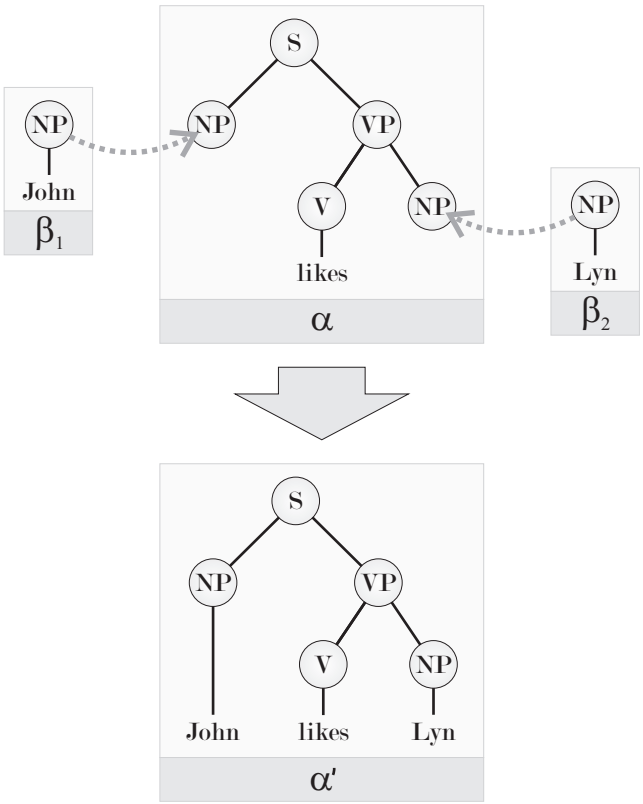


Fig. 37.16: An example for the substitution operation.

ementary 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 4.5.8 on page 174.

37.3.11 S-Expressions

S-expressions[89] (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[90] [1311, 1312, 1313] and Scheme[91] [1314] programming languages. Their

[89] `http://en.wikipedia.org/wiki/S-expression` [accessed 2007-07-03]
[90] `http://en.wikipedia.org/wiki/Lisp_programming_language` [accessed 2007-07-03]
[91] `http://en.wikipedia.org/wiki/Scheme_%28programming_language%29` [accessed 2007-07-03]

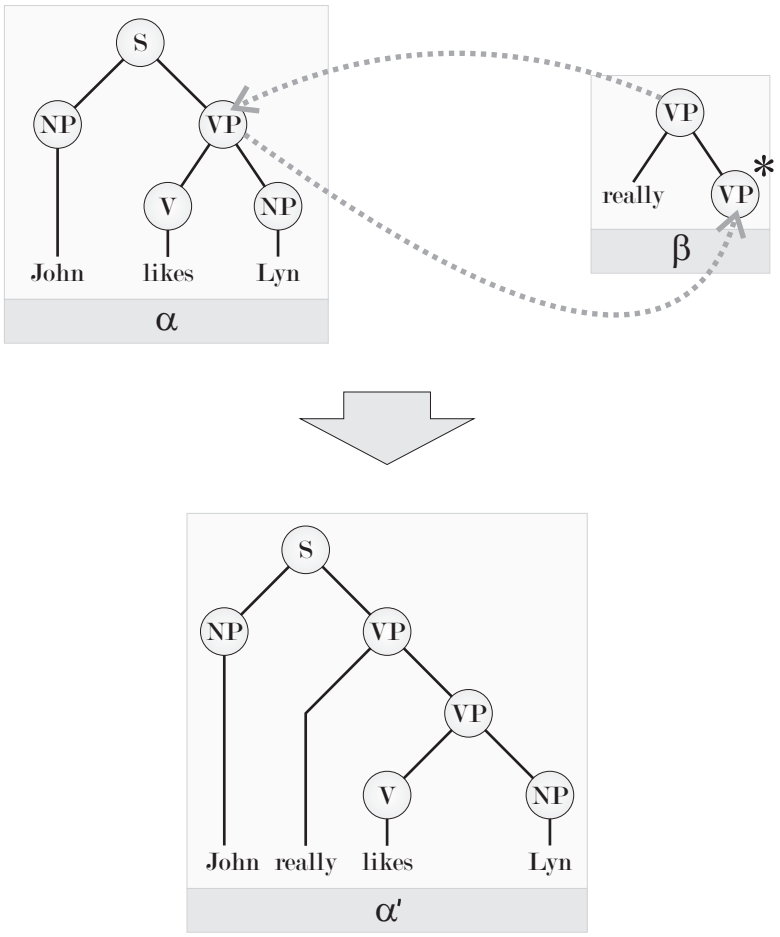


Fig. 37.17: An example for the adjunction operation.

most common feature is that they are parenthesized prefix notations (often also known as Polish notation[92]).

In 1997, Ron Rivest handed in a standardization draft [1315] for S-expressions to be considered for publication as RFC. It was however never approved but is still the foundation for many other publications and RFCs.

```
(defun fibonacci (N)
  (if (or (zerop N) (= N 1))
      1
```

[92] http://en.wikipedia.org/wiki/Polish_notation [accessed 2007-07-04]

```
4       (+ (fibonacci (- N 1)) (fibonacci (- N 2)))))
```

Listing 37.14: A small Lisp-example: How to compute Fibonacci numbers.

Part V

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. So its props to:

Stefan Achler
For working together at the 2007 DATA-MINING-CUP Contest.
see Section 18.1.2 on page 300

Steffen Bleul
For working together at the 2006 and 2007 Web Service Challenge.
see Section 18.2 on page 312

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.

Gan Min
For careful reading and pointing out inconsistencies and spelling mistakes.
Especially in the Pareto optimization area Section 1.3.2 on page 14.

Kurt Geihs
For being supportive and contributing to many of my research projects.
see for example Section 18.1.2 on page 300 and [551, 552, 548, 1055, 1056]

Martin Göb
For working together at the 2007 DATA-MINING-CUP Contest.
see Section 18.1.2 on page 300

Ibrahim Ibrahim
For careful proofreading.
see Chapter 1 on page 3

Roland Reichle

For discussing with me the relation of maximum likelihood estimation and symbolic regression.
see Section 35.6.1 on page 552

Richard Seabrook
For pointing out mistakes in some of the set theory definitions.
see Chapter 34 on page 501

Christian Voigtmann
For working together at the 2007 DATA-MINING-CUP Contest.
see Section 18.1.2 on page 300

Wibul Wongpoowarak
For valuable suggestions in different areas, like including random optimization.
see Section 35.7.1 on page 560 [880], Chapter 9 on page 227

Michael Zapf
For helping me to make many sections more comprehensible.
See for instance Section 20.1.1 on page 337, Section 18.1.2 on page 300, and Section 4.5 on page 159.

D

Citation Suggestion

```
@book{W2008GOEB,
    author       = {Thomas Weise},
    title        = {Global Optimization Algorithms – Theory and Application},
    year         = {2008},
    month        =  jan # {~4,},
    howpublished = {Online as e-book},
    edition      = {2008-1-4},
    institution  = {University of Kassel, Distributed Systems Group},
    organization = {University of Kassel, Distributed Systems Group},
    publisher    = {Thomas Weise},
    copyright    = {Copyright (c) 2006-2008 Thomas Weise, licensed under GNU
                    FDL},
    keywords     = {global optimization, evolutionary computation, evolutionary
                    algorithms, genetic algorithms, genetic programming,
                    evolution strategy, learning classifier systems, 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    2008-1-4
%8    2008-1-4
%7    2008-1-4
%9    Online available as e-book at http://www.it-weise.de/
```

References

[1] 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].

[2] Wikipedia. Online available at `http://en.wikipedia.org/` [accessed 2008-1-4].

[3] Linus Pauling. *The Nature of the Chemical Bond*. Cornell Univ. Press, Ithaca, New York, ISBN: 0-8014-0333-2, 1960. Online available at `http://osulibrary.oregonstate.edu/specialcollections/coll/pauling/` [accessed 2007-07-12].

[4] 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].

[5] 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].

[6] 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].

[7] P. M. Pardalos, Nguyen Van Thoai, and Reiner Horst. *Introduction to Global Optimization*. Nonconvex Optimization and Its Applications. Springer, second edition, ISBN: 978-0792367567, December 31, 2000. First edition: June 30, 1995, ISBN: 978-0792335566.

[8] Zbigniew Michalewicz and David B. Fogel. *How to Solve It: Modern Heuristics*. Springer, second, revised and extended edition, ISBN: 978-3540224945, December 2004.

[9] V. J. Rayward-Smith, I. H. Osman, C. R. Reeves, and G. D. Smith, editors. *Modern Heuristic Search Methods*. Wiley, ISBN: 978-0471962809, December 1996.

[10] Judea Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. The Addison-Wesley series in artificial intelligence. Addison-Wesley Pub (Sd), ISBN: 978-0201055948, April 1984.

[11] John R. Koza. *Genetic Programming, On the Programming of Computers by Means of Natural Selection*. A Bradford Book, The MIT Press, Cambridge, Massachusetts, 1992 first edition, 1993 second edition, ISBN: 0262111705, 1992.

[12] Sewall Wright. The roles of mutation, inbreeding, crossbreeding and selection in evolution. In *Proceedings of the Sixth Annual Congress of Genetics*, 1932, volume 1, pages 356–366. Online available at `http://www.blackwellpublishing.com/ridley/classictexts/wright.pdf` [accessed 2007-08-11].

[13] Stuart Alan Kauffman. *The Origins of Order: Self-Organization and Selection in Evolution*. Oxford University Press, ISBN: 0195079515, May 1993.

[14] Sergey Gavrilets. *Fitness Landscapes and the Origin of Species*. Number MPB-41 in Monographs in Population Biology. Princeton University Press, ISBN: 069111983X, July 2004.

[15] Niko Beerenwinkel, Lior Pachter, and Bernd Sturmfels. Epistasis and shapes of fitness landscapes, 2006. Eprint arXiv:q-bio/0603034. Online available at `http://arxiv.org/abs/q-bio.PE/0603034` [accessed 2007-08-05].

[16] Richard Dawkins. *Climbing Mount Improbable*. Penguin Books (UK) and W.W. Norton & Company (USA), London, 1 edition, ISBN: 0670850187 and 0140179186, ASIN: 0393039307 and 0393316823, 1996.

[17] Melanie Mitchell. *An Introduction to Genetic Algorithms*. Complex Adaptive Systems. The MIT Press, reprint edition, ISBN: 0-2626-3185-7, February 1998.

[18] William B. Langdon and Riccardo Poli. *Foundations of Genetic Programming*. Springer, Berlin, first edition, ISBN: 3540424512, January 2002.

[19] Agoston E. Eiben and James E. Smith. *Introduction to Evolutionary Computing*. Natural Computing Series. Springer, first edition, ISBN: 978-3540401841, November 2003.

[20] Kalyanmoy Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. Wiley Interscience Series in Systems and Optimization. John Wiley & Sons, Inc., New York, NY, USA, ISBN: 978-0471873396, May 2001.

[21] 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*, 1999, pages 3–13. See proceedings [249]. Online available at `http://citeseer.ist.psu.edu/coellocoello99updated.html` [accessed 2007-08-25].

[22] 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 [42].

[23] Kalyanmoy Deb. Evolutionary algorithms for multi-criterion optimization in engineering design. In Kaisa Miettinen, Marko M. Mäkelä, Pekka Neittaanmäki, and Jacques Periaux, editors, *Evolutionary Algorithms in Engineering and Computer Science*, pages 135–161. John Wiley & Sons, Ltd, Chichester, UK, 1999. Online available at `http://citeseer.ist.psu.edu/deb99evolutionary.html` and `http://www.lania.mx/~ccoello/EMOO/deb99.ps.gz` [accessed 2007-08-25].

[24] Martin J. Osborne and Ariel Rubinstein. *A Course in Game Theory*. The MIT Press, ISBN: 0-2626-5040-1, July 1994.

[25] Drew Fudenberg and Jean Tirole. *Game Theory*. The MIT Press, ISBN: 0-2620-6141-4, 978-0-262-06141-4, August 1991.

[26] Vira Chankong and Yacov Y. Haimes. *Multiobjective Decision Making Theory and Methodology*. North-Holland, Elsevier, Dover Publications, New York, ISBN: 978-0444007100, 978-0486462899, January 1983.

[27] Ralph E. Steuer. *Multiple Criteria Optimization: Theory, Computation and Application*. Krieger Pub Co, reprint edition, ISBN: 978-0894643934, August 1989.

[28] 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)*, University of Virginia, Charlottesville, Virginia, USA, June 12-18, 1998, volume 487 of *Lecture Notes in Economics and Mathematical Systems*. Springer, ISBN: 978-3540672661. See `http://www.virginia.edu/~risk/mcdm98.html` [accessed 2007-09-10]. Published June 15, 2000.

[29] 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. Online available at `http://www.springerlink.com/content/m71638106736h581/fulltext.pdf` and `http://dx.doi.org/10.1023/A:1022639028824` [accessed 2007-09-10].

[30] Aharon Ben-Tal. Characterization of pareto and lexicographic optimal solutions. In *Proceedings of the Third Conference on Multiple Criteria Decision Making: Theory and Application*, 1979, pages 1–11. See proceedings [153].

[31] T. Satoh, Tadashi Ishihara, and H. Inooka. Systematic design via the method of inequalities. *Control Systems Magazine, IEEE*, 16:57–65, October 1996.

[32] J. F. Whidborne, D.-W. Gu, and I. Postlethwaite. Algorithms for the method of inequalities – a comparative study. In *Procedings of the 1995 American Control Conference*, Seattle, Washington, 1995, pages 3393–3397.

[33] 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*, 15:811–816, May 2000.

[34] 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*, Chelmsford, UK, 1993, pages 1–8.

[35] Abraham Charnes and William Wager Cooper. *Management Models and Industrial Applications of Linear Programming*. John Wiley & Sons Inc, New York, ISBN: 978-0471148500, December 1961.

[36] 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].

[37] 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` [accessed 2007-08-25].

[38] Kalyanmoy Deb. Solving goal programming problems using multi-objective genetic algorithms. In *Proceedings of Congress on Evolutionary Computation*, 1999, volume 1, pages 77–84. See proceedings [249].

[39] 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)*, The University of Otago, Dunedin, New Zealand, November 2001, pages 113–122. Online available at `http://www.lania.mx/~ccoello/EMOO/sarker01a.pdf.gz` [accessed 2007-08-25].

[40] Hartmut Pohlheim. Geatbx introduction – evolutionary algorithms: Overview, methods and operators. Technical report, 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].

[41] 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].

[42] 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*, 28(1):38–47, 1998. Online available at http://citeseer.ist.psu.edu/27937.html [accessed 2007-09-19]. See also [22].

[43] Carlos M. Fonseca. Decision making in evolutionary optimization (abstract of invited talk). In *4th International Conference on Evolutionary Multi-Criterion Optimization*, 2007. See proceedings [265].

[44] Carlos M. Fonseca and Peter J. Fleming. Multiobjective optimization and multiple constraint handling with evolutionary algorithms. In *Practical Approaches to Multi-Objective Optimization*, 2004. See proceedings [256]. Online available at http://drops.dagstuhl.de/opus/volltexte/2005/237/ [accessed 2007-09-19].

[45] 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. See proceedings [132].

[46] Eik Fun Khor, Kay Chen Tan, Tong Heng Lee, and C. K. Goh. A study on distribution preservation mechanism in evolutionary multiobjective optimization. *Artificial Intelligence Review*, 23(1):31–33, March 2005. Online available at http://www.springerlink.com/content/h3w111g418gn1045/fulltext.pdf [accessed 2007-08-25].

[47] J. Shekel. Test functions for multimodal search techniques. In *Proceedings of the Fifth Annual Princeton Conference on Information Science and Systems*, Princeton, NJ, USA, 1971. 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].

[48] Anatas Žilinskas. Algorithm as 133: Optimization of one-dimensional multimodal functions. *Applied Statistics*, 27(3):367–375, 1978.

[49] 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 2003. 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].

[50] J. David Schaffer, Larry J. Eshelman, and Daniel Offutt. Spurious correlations and premature convergence in genetic algorithms. In *Pro-*

ceedings of the First Workshop on Foundations of Genetic Algorithms (FOGA), June 1990, pages 102–112. See proceedings [439].

[51] Günter Rudolph. Self-adaptive mutations may lead to premature convergence. *IEEE Transactions on Evolutionary Computation*, 5(4):410–414, 2001. Also: technical report CI–73/99 of Fachbereich Informatik, Universität Dortmund, 44221 Dortmund, Germany. Online available at `http://citeseer.ist.psu.edu/444363.html` [accessed 2007-07-28].

[52] 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*, 1991, pages 115–122. See proceedings [445].

[53] Shana Shiang-Fong Smith. Using multiple genetic operators to reduce premature convergence in genetic assembly planning. *Computers in Industry*, 54(1):35–49, 2004.

[54] 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].

[55] Agoston 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].

[56] Nitin Muttil and Shie-Yui Liong. Superior exploration–exploitation balance in shuffled complex evolution. *Journal of Hydraulic Engineering*, 130(12):1202–1205, December 2004.

[57] 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*, 2005, pages 1531–1538. See proceedings [299]. Online available at `http://doi.acm.org/10.1145/1068009.1068250` [accessed 2007-08-29].

[58] Kalyanmoy Deb. Genetic algorithms for optimization. Technical report, Kanpur Genetic Algorithms Laboratory (KanGAL), Kanpur, PIN 208 016, India, KanGAL, Kanpur, PIN 208 016, India, 2001. KanGAL Report Number 2001002, 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].

[59] 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*, 2006, pages 1305–1312. See proceedings [298]. Online available at `http://doi.acm.org/10.1145/1143997.1144200` [accessed 2007-09-10].

[60] 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].

[61] Simon Ronald. Preventing diversity loss in a routing genetic algorithm with hash tagging. *Complexity International*, 2, 1995. Online available at http://www.complexity.org.au/ci/vol02/sr_hash/ [accessed 2007-07-28].

[62] 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.

[63] 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 and 2007-08-20 [accessed .]

[64] Krasimir Kolarov. Landscape ruggedness in evolutionary algorithms. In *Proceedings of The IEEE Conference on Evolutionary Computation*, 1997, pages 19–24. See proceedings [251]. Online available at http://citeseer.ist.psu.edu/14650.html and http://de.scientificcommons.org/27191 [accessed 2007-08-19].

[65] Henrick Flyvbjerg and Benny Lautrup. Evolution in a rugged fitness landscape. *Physical Review*, pages 6714–6723, March 1992. 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].

[66] Edward D. Weinberger. Correlated and uncorrelatated fitness landscapes and how to tell the difference. *Biological Cybernetics*, 63:325–336, 1990.

[67] S. Verel, P. Collard, and M. Clergue. Measuring the evolvability landscape to study neutrality. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, 2006, pages 613–614. See proceedings [298]. Online available at http://portal.acm.org/citation.cfm?id=1144107 [accessed 2007-08-13].

[68] Albert Donally Bethke. *Genetic algorithms as function optimizers.* PhD thesis, 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].

[69] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, first edition, ISBN: 0201157675, January 1989.

[70] Gunar E. Liepins and Michael D. Vose. Deceptiveness and genetic algorithm dynamics. In *Proceedings of the First Workshop on Foundations of Genetic Algorithms (FOGA)*, 1991, pages 36–50. See proceedings [439]. Online available at http://www.osti.gov/bridge/servlets/purl/6445602-CfqU6M/ [accessed 2007-11-05].

[71] Andreas Wagner. Robustness, evolvability, and neutrality. *FEBS Lett*, 579(8):1772–1778, March 2005. 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].

[72] Andreas Wagner. *Robustness and Evolvability in Living Systems*. Princeton Studies in Complexity. Princeton University Press, ISBN: 0691122407, August 2005.

[73] Lionel Barnett. Tangled webs: Evolutionary dynamics on fitness landscapes with neutrality. Master's thesis, School of Cognitive Science, University of East Sussex, Brighton, Brighton, 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].

[74] 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].

[75] Günter P. Wagner and Lee Altenberg. Complex adaptations and the evolution of evolvability. *Evolution*, 50(3):967–976, 1996. Online available at http://citeseer.ist.psu.edu/437225.html [accessed 2007-08-05].

[76] R. J. Riedl. A systems-analytical approach to macroevolutionary phenomena. *Quarterly Review of Biology*, pages 351–370, 1977.

[77] M. Kirschner and J. Gerhart. Evolvability. *Proceedings of the National Academy of Science of the USA (PNAS)*, 95(15):8420–8427, July 1998. Online available at http://www.pnas.org/cgi/content/full/95/15/8420 [accessed 2007-08-05].

[78] Richard Dawkins. The evolution of evolvability. In Christopher G. Langton, editor, *ALIFE*, Los Alamos, New Mexico, USA, 1987, pages 201–220, Redwood City, CA. Addison-Wesley.

[79] Lee Altenberg. Genome growth and the evolution of the genotype-phenotype map. In *Evolution and Biocomputation, Computational Models of Evolution*, 1992, pages 205–259, ISBN: 3-540-59046-3, 0-387-59046-3. See proceedings [335]. Online available at http://citeseer.ist.psu.edu/254722.html [accessed 2007-07-29].

[80] Tom Smith, Phil Husbands, Paul Layzell, and Michael O'Shea. Fitness landscapes and evolvability. *Evolutionary Computation*, 10(1):1–34, Spring 2002. Online available at `http://wotan.liu.edu/docis/dbl/evocom/2002_10_1_1_FLAE.html` [accessed 2007-07-28].

[81] Rob Shipman, Marc Shackleton, Marc Ebner, and R. Watson. Neutral search spaces for artificial evolution: a lesson from life, 2000. Submitted to Artificial Life, VII. Online available at `http://citeseer.ist.psu.edu/shipman00neutral.html` and `http://www.demo.cs.brandeis.edu/papers/alife7_shipman.ps.gz` [accessed 2007-07-29].

[82] Vesselin K. Vassilev and Julian F. Miller. The advantages of landscape neutrality in digital circuit evolution. In *ICES '00: Proceedings of the Third International Conference on Evolvable Systems*, Edinburgh, Scotland, UK, April 17-19, 2000, volume 1801/2000, pages 252–263. Springer-Verlag, London, UK, ISBN: 3-540-67338-5, ISSN: 0302-9743 (Print) 1611-3349 (Online). Online available at `http://citeseer.ist.psu.edu/vassilev00advantages.html` [accessed 2007-11-02].

[83] 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*, 2002, pages 13–25. See proceedings [573]. 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].

[84] 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*, 2005, pages 1433–1440. See proceedings [299]. Online available at `http://doi.acm.org/10.1145/1068009.1068237` and `http://www.citeulike.org/user/denizinho/article/995596` [accessed 2007-08-19].

[85] 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*, 1999, volume 3, pages 2047–2053. See proceedings [249].

[86] 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*, 2004, pages 111–120. See proceedings [324].

[87] Jürgen Branke. The moving peaks benchmark. Technical report, Institute AIFB, University of Karlsruhe, D-76128 Karlsruhe, Germany, 1999. Online available at `http://www.aifb.uni-karlsruhe.de/~jbr/MovPeaks/` [accessed 2007-08-19].

[88] Shengxiang Yang, Yew-Soon Ong, and Yaochu Jin, editors. *Evolutionary Computation in Dynamic and Uncertain Environments*, volume 51 of *Studies in Computational Intelligence*. Springer, ISBN: 978-3-540-49772-1, 2007. Presentation online available at `http://www.soft-computing.de/Jin_CEC04T.pdf.gz` [accessed 2007-08-19].

[89] 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.

[90] Claus O. Wilke. *Evolutionary Dynamics in Time-Dependent Environments*. PhD thesis, Fakultät für Physik und Astronomie, Ruhr-Universität Bochum, July 1999, ISBN: 978-3826561993. Online available at `http://wlab.biosci.utexas.edu/~wilke/ps/PhD.ps.gz` [accessed 2007-08-19].

[91] 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 [92].

[92] Jürgen Branke. *Evolutionary Optimization in Dynamic Environments*, volume 3 of *Genetic Algorithms and Evolutionary Computation*. Kluwer Academic Publishers, ISBN: 978-0792376316, December 2001. See also [91].

[93] Ronald Walter Morrison. *Designing Evolutionary Algorithms for Dynamic Environments*. PhD thesis, George Mason University, USA, 2002. Advisor: Kenneth Alan De Jong. See also [94].

[94] Ronald Walter Morrison. *Designing Evolutionary Algorithms for Dynamic Environments*, volume 24 of *Natural Computing*. Springer, Berlin, ISBN: 978-3540212317, ISSN: 0263-5747, June 2004. See also [93].

[95] 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*, 4(3), October 2004. Online available at `http://journal.info.unlp.edu.ar/Journal/journal12/papers.html` [accessed 2007-08-19].

[96] 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].

[97] 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*, Orlando, Florida, USA, June 9-13, 2002, volume 13, pages 265–270. Online available at `http://antho.huntingdon.edu/publications/default.html` [accessed 2007-08-19].

[98] 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, 2006, pages 51–58. See proceedings [298]. Online available at http://doi.acm.org/10.1145/1143997.1144005 [accessed 2007-08-19].

[99] 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)*, Jinan, China, 2006, volume 2, pages 1024–1028, Washington, DC, USA. IEEE Computer Society, ISBN: 0-7695-2528-8.

[100] Tim Blackwell. Particle swarm optimization in dynamic environments. In Shengxiang Yang, Yew-Soon Ong, and Yaochu Jin, editors, *Evolutionary Computation in Dynamic and Uncertain Environments*, volume 51 of *Studies in Computational Intelligence*, ISBN: 978-3-540-49772-1, chapter 2, pages 29–52. Springer, 2007. See [88]. Online available at http://igor.gold.ac.uk/~mas01tb/papers/PSOdynenv.pdf [accessed 2007-08-19].

[101] Rui Mendes and Arvind S. Mohais. Dynde: a differential evolution for dynamic optimization problems. In *Proceedings of 2005 IEEE Congress on Evolutionary Computation*, 2005, volume 3, pages 2808–2815. See proceedings [243]. Online available at http://www.di.uminho.pt/~rcm/publications/DynDE.pdf and http://en.scientificcommons.org/8412355 [accessed 2007-08-13].

[102] 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].

[103] 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*, 2001, pages 213–222. See proceedings [282]. Online available at http://pacosy.informatik.uni-leipzig.de/pv/Personen/middendorf/Papers/ [accessed 2007-08-19].

[104] 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)*, 2001, pages 860–867. See proceedings [310]. Online available at http://citeseer.ist.psu.edu/693476.html [accessed 2007-08-19].

[105] Tom Dietterich. Overfitting and undercomputing in machine learning. *ACM Computing Surveys (CSUR)*, 27(3):326–327, 1995. Online available at http://doi.acm.org/10.1145/212094.212114 and http://citeseer.ist.psu.edu/dietterich95overfitting.html [accessed 2007-09-13].

[106] 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.

[107] 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].

[108] 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)*, Como, Italy, July 24-27, 2000, volume 1, pages 1114–1119. IEEE Computer Society, ISSN: 1098-7576. Online available at http://citeseer.ist.psu.edu/lawrence00overfitting.html [accessed 2007-09-13].

[109] Warren S. Sarle. Stopped training and other remedies for overfitting. In *Proceedings of the 27th Symposium on the Interface: Computing Science and Statistics*, Pittsburgh, Pennsylvania, June 23, 1995. Online available at http://citeseer.ist.psu.edu/71530.html and ftp://ftp.sas.com/pub/neural/inter95.ps.Z [accessed 2007-09-13].

[110] Paul L. Rosin and Freddy Fierens. Improving neural network generalisation. In *Proceedings of the International Geoscience and Remote Sensing Symposium, IGARSS'95*, Florenz, Italy, July 10-14, 1995, volume 1. 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].

[111] 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].

[112] 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].

[113] 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*, 2006, pages 45–54. See proceedings [239] and (for an extended version) [114]. Online available at http://www.item.uni-bremen.de/staff/zilli/zielinski06stopping_PSO.pdf [accessed 2007-09-13].

[114] Karin Zielinski and Rainer Laur. Stopping criteria for a constrained single-objective particle swarm optimization algorithm. *Informatica*, 31:51–59, 2007. Extended version of [113]. 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].

[115] Joel N. Morse. Reducing the size of the nondominated set: Pruning by clustering. *Computers & Operations Research*, 7:55–66, 1980.

[116] 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. Online available at http://www.rci.rutgers.edu/~coit/RESS_2007.pdf [accessed 2007-09-10].

[117] 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)*, Gdansk, Poland, June 2005, pages 391–398. Online available at http://www.engr.rutgers.edu/ie/research/working_paper.html [accessed 2007-07-29].

[118] 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 [381]. Online available at http://citeseer.ist.psu.edu/543052.html and http://www.lania.mx/~ccoello/EMOO/fieldsend03.pdf.gz [accessed 2007-08-25].

[119] Heidi A. Taboada and David W. Coit. Data clustering of solutions for multiple objective system reliability optimization problems. *Quality Technology and Quantitative Management Journal*, 4:35–54, June 2007. Online available at http://www.soe.rutgers.edu/ie/research/working_paper/paper05-019.pdf [accessed 2007-09-10].

[120] 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.

[121] V. Bhaskar, Santosh K. Gupta, and A.K. Ray. Applications of multi-objective optimization in chemical engineering. *Reviews in Chemical Engineering*, 16(1):1–54, 2000.

[122] Nikolaos V. Sahinidis and Mohit Tawarmalani. Applications of global optimization to process and molecular design. *Computers and Chemical Engineering*, 24:2157–2169, October 1, 2000. Online available at http://citeseer.ist.psu.edu/397733.html [accessed 2007-09-01].

[123] Christodoulos A. Floudas and P. M. Pardalos, editors. *Frontiers in Global Optimization*. Nonconvex Optimization and Its Applications. Springer US, ISBN: 978-1402076992, November 30, 2003.

[124] Christodoulos A. Floudas, editor. *Deterministic Global Optimization: Theory, Methods and Applications*, volume 37 of *Nonconvex Optimization and Its Applications*. Springer-Verlag GmbH, ISBN: 978-0-7923-6014-8, 1999.

[125] 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*, Vancouver, British Columbia, Canada, June 20-25, 2004. OM AE2004-51225. Online available at `http://130.149.35.79/downloads/publikationen/2004/OMAE2004-51225.pdf` [accessed 2007-09-01].

[126] G. Dzemyda, V. Saltenis, and A. Zilinskas, editors. *Stochastic and Global Optimization*. Nonconvex Optimization and Its Applications. Springer-Verlag GmbH, ISBN: 978-1402004841, March 1, 2002.

[127] Max Jerrell. Applications of public global optimization software to difficult econometric functions. In *Computing in Economics and Finance 200*, Departament d'Economia i Empresa, Universitat Pompeu Fabra, Ramon Trias Fargas, 25,27, 08005, Barcelona, Spain, 2000. Society for Computational Economics. Number 161. Online available at `http://econpapers.repec.org/paper/scescecf0/161.htm` [accessed 2007-09-01].

[128] 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*, 43:491–499, February-March 2002. Online available at `http://dx.doi.org/10.1016/S0898-1221(01)00300-5` [accessed 2007-09-01].

[129] 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*, December 1998, volume 3430, pages 175–183.

[130] 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*, 45:103001, October 2006.

[131] Yacov Y. Haimes, Warren Hall, and Herbert T. Freedman. *Multiobjective Optimization in Water Resource Systems*. Elsevier, New York, 1975. ASIN: B000UUMGXE.

[132] Andreas König, Mario Köppen, Ajith Abraham, Christian Igel, and Nikola Kasabov, editors. *7th International Conference on Hybrid Intelligent Systems (HIS 2007)*, Fraunhofer-Center, University of Kaiserslautern, Kaiserslautern, Germany, September 17-19, 2007. IEEE Computer Society, ISBN: 0-7695-2946-1. Library of Congress Number 2007936727, Product Number E2946. see `http://his07.hybridsystem.com/` [accessed 2007-09-01].

[133] *6th International Conference on Hybrid Intelligent Systems (HIS 2006)*, AUT Technology Park, Auckland, New Zealand, December 13-15, 2006. IEEE Computer Society, ISBN: 0-7695-2662-4. see `http://his06.hybridsystem.com/` [accessed 2007-09-01].

[134] Nadia Nedjah, Luiza de Macedo Mourelle, Ajith Abraham, and Mario Köppen, editors. *5th International Conference on Hybrid Intelligent Systems (HIS 2005)*, Pontifical Catholic University of Rio de Janeiro: PUC-Rio, Rio de Janeiro, Brazil, November 6-9, 2005. IEEE Computer Society, ISBN: 0-7695-2457-5. see `http://his05.hybridsystem.com/` [accessed 2007-09-01].

[135] *4th International Conference on Hybrid Intelligent Systems (HIS 2004)*, Kitakyushu, Japan, December 5-8, 2005. IEEE Computer Society, ISBN: 0-7695-2291-2.

[136] 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*, Monash Conference Centre, Level 7, 30 Collins Street, Melbourne, Australia, December 14-17, 2003, volume 105 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, ISBN: 1-58603-394-8. see `http://his03.hybridsystem.com/` [accessed 2007-09-01].

[137] Ajith Abraham, Javier Ruiz del Solar, and Mario Köppen, editors. *Soft Computing Systems – Design, Management and Applications, HIS 2002*, Santiago, Chile, December 1-4, 2002, volume 87 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, ISBN: 1-58603-297-6.

[138] Ajith Abraham and Mario Köppen, editors. *Hybrid Information Systems, Proceedings of the First International Workshop on Hybrid Intelligent Systems*, Adelaide University's main campus, Adelaide, Australia, December 11-12, 2002, Advances in Soft Computing. Physica-Verlag, ISBN: 3-7908-1480-6. see `http://his01.hybridsystem.com/` [accessed 2007-09-01].

[139] Matthias Ehrgott, editor. *Proceedings of the 19th International Conference on Multiple Criteria Decision Making: MCDM for Sustainable Energy and Transportation Systems (MCDM'2008)*, Auckland, New Zealand, June 7-12, 2008. See `https://secure.orsnz.org.nz/mcdm2008/` [accessed 2007-09-10].

[140] Constantin Zopounidis, editor. *Proceedings of the 18th International Conference on Multiple Criteria Decision Making (MCDM'2006)*, MAICh (Mediterranean Agronomic Institute of Chania) Conference Centre, Chania, Crete, Greece, June 9-13, 2006. See `http://www.dpem.tuc.gr/fel/mcdm2006/` [accessed 2007-09-10].

[141] Bill Wedley, editor. *Proceedings of the 17th International Conference on Multiple Criteria Decision Making (MCDM'2004)*, Whistler Conference Center, Whistler, British Columbia, Canada, August 6-9, 2004. See `http://www.bus.sfu.ca/events/mcdm/` [accessed 2007-09-10].

[142] Mikulas Luptacik and Rudolf Vetschera, editors. *Proceedings of the First MCDM Winter Conference, 16th International Conference on Multiple Criteria Decision Making (MCDM'2002)*, Hotel Panhans,

Semmering, Austria, February 12-22, 2002. See http://orgwww.bwl.univie.ac.at/mcdm2002 [accessed 2007-09-10].

[143] 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)*, Middle East Technical University, Ankara, Turkey, July 10-14, 2000, volume 507 of *Lecture Notes in Economics and Mathematical Systems*. Springer, ISBN: 978-3540423775. See http://mcdm2000.ie.metu.edu.tr/ [accessed 2007-09-10]. Published November 9, 2001.

[144] Theo Stewart, editor. *Proceedings of the 13th International Conference on Multiple Criteria Decision Making: Trends in Multicriteria Decision Making (MCDM'1997)*, University of Cape Town, Cape Town, South Africa, January 6-10, 1997, volume 465 of *Lecture Notes in Economics and Mathematical Systems*. Springer-Verlag Telos, ISBN: 978-3540647416. See http://www.uct.ac.za/depts/maths/mcdm97 [accessed 2007-09-10]. Published January 15, 1999.

[145] Günter Fandel, Thomás Gál, and Thomas Hanne, editors. *Proceedings of the 12th International Conference on Multiple Criteria Decision Making (MCDM'1995)*, Hagen, Germany, 1995, volume 448 of *Lecture Notes in Economics and Mathematical Systems*. Springer, ISBN: 978-3540620976. Published on March 21, 1997.

[146] João Climaco, editor. *Proceedings of the 11th International Conference on Multiple Criteria Decision Making: Multicriteria Analysis (MCDM'1994)*, Coimbra, Portugal, August 1-6, 1994. Springer, ISBN: 978-3540620747. Published in May 1997.

[147] 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)*, Taipei, Taiwan, 1992. Springer, ISBN: 978-0387942971. Published in June 1994.

[148] 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)*, Fairfax, USA, 1990. Springer, ISBN: 978-0387978055. Published in July 1992.

[149] 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)*, Manchester, UK, 1988, Lecture Notes in Economics and Mathematical Systems. Springer, ISBN: 978-0387517957. Published in December 1989.

[150] H. Nakayama and Y. Sawaragi, editors. *Proceedings of the 7th International Conference on Multiple Criteria Decision Making (MCDM'1986)*, Kyoto, Japan, 1986.

[151] Yacov Y. Haimes, editor. *Proceedings of the 6th International Conference on Multiple Criteria Decision Making: Decision Making with*

Multiple Objectives (MCDM'1984), Cleveland, Ohio, USA, June 4-6, 1984, volume 242 of *Lecture Notes in Economics and Mathematical Systems*. Springer-Verlag Berlin and Heidelberg GmbH & Co. KG, ISBN: 978-3540152231. Published December 31, 1985.

[152] Pierre Hansen, editor. *Proceedings of the 5th International Conference on Multiple Criteria Decision Making: Essays and Surveys on Multiple Criteria Decision Making (MCDM'1982)*, Mons, Belgium, 1982, volume 209 of *Lecture Notes in Economics and Mathematical Systems*. Springer, ISBN: 978-0387119915. Published in March 1983.

[153] Joel N. Morse, editor. *Proceedings of the 4th International Conference on Multiple Criteria Decision Making: Organizations, Multiple Agents With Multiple Criteria (MCDM'1980)*, Newark, Delaware, USA, 1980, Lecture Notes in Economics and Mathematical Systems. Springer, ISBN: 978-0387108216. Published in July 1981.

[154] Günter Fandel and Thomás Gál, editors. *Proceedings of the 3rd International Conference on Multiple Criteria Decision Making: Theory and Application (MCDM'1979)*, Hagen/Königswinter, (West) Germany, August 20-24, 1979, volume 17 of *Lecture Notes in Economics and Mathematical Systems*. Springer-Verlag, Berlin, Germany, ISBN: 978-0387099637. Published in May 1980.

[155] Stanley Zionts, editor. *Proceedings of the 2nd International Conference on Multiple Criteria Decision Making (MCDM'1977)*, Buffalo, New York, USA, 1977.

[156] Herve Thiriez and Stanley Zionts, editors. *Proceedings of the 1st International Conference on Multiple Criteria Decision Making (MCDM'1975)*, Jouy-en-Josas, France, May 21-23, 1975, Lecture notes in economics and mathematical systems. Springer, ISBN: 978-0387077949. Published in 1976.

[157] *Proceedings of the 13th International Conference on Soft Computing, MENDEL'07*, Prague, Czech Republic, September 5-7, 2007.

[158] *Proceedings of the 12th International Conference on Soft Computing, MENDEL'06*, Brno University of Technology, Brno, Czech Republic, May 31-June 2, 2006, ISBN: 80-214-3195-4.

[159] *Proceedings of the 11th International Conference on Soft Computing, MENDEL'05*, Brno University of Technology, Brno, Brno, Czech Republic, June 15-17, 2005, ISBN: 80-214-2961-5.

[160] *Proceedings of the 10th International Conference on Soft Computing, MENDEL'04*, Brno University of Technology Faculty of Mechanical Engeneering, Brno, Brno, Czech Republic, June 16-18, 2004, ISBN: 80-214-2676-4.

[161] *Proceedings of the 9th International Conference on Soft Computing, MENDEL'03*, Brno University of Technology, Brno, Czech Republic, 2003, ISBN: 80-214-2411-7.

[162] *Proceedings of the 8th International Conference on Soft Computing, MENDEL'02*, Brno University of Technology, Brno, Czech Republic, June 5-7, 2002, ISBN: 80-214-2135-5.

[163] *Proceedings of the 7th International Conference on Soft Computing, Evolutionary Computation, Genetic Programing, Fuzzy Logic, Rough Sets, Neural Networks, Fractals, Bayesian Methods, MENDEL 2001*, Brno University of Technology, Brno, Czech Republic, June 6-8, 2001, ISBN: 80-7248-107-X.

[164] Pavel Osmera, editor. *Proceedings of the 6th International Conference on Soft Computing, MENDEL 2000*, Brno University of Technology, Brno, Czech Republic, June 7-9, 2000, ISBN: 80-214-1609-2.

[165] *Proceedings of the 3rd International Conference on Genetic Algorithms*, Brno University of Technology, Brno, Czech Republic, June 1997, ISBN: 80-214-1131-7.

[166] *Proceedings of the 4th International Mendel Conference on Genetic Algorithms, Optimisation Problems, Fuzzy Logic, Neural Networks, Rough Sets*, Technical University of Brno, Faculty of Mechanical Engineering, Brno, Czech Republic, June 24-26, 1998, ISBN: 80-214-1131-7.

[167] *Proceedings of the 2nd International Mendel Conference on Genetic Algorithms, MENDEL 1997*, Brno University of Technology, Brno, Czech Republic, June 26-28, 1997, ISBN: 80-214-0769-7.

[168] *Proceedings of the 2nd International Mendel Conference on Genetic Algorithms, MENDEL 1996*, Brno University of Technology, Brno, Czech Republic, June 26-28, 1996, ISBN: 80-214-0769-7.

[169] *Proceedings of Mendel'95, the 1st International Mendel Conference on Genetic Algorithms*, Brno, Czech Republic, September 1995.

[170] *The Seventh Metaheuristics International Conference*, Montreal, Canada, June 25-29, 2007. See `http://www.mic2007.ca/` [accessed 2007-09-12].

[171] *6th Metaheuristics International Conference (MIC2005)*, Vienna, Austria, August 22-26, 2005. See `http://www.mic2005.org/` [accessed 2007-09-12].

[172] Toshihide Ibaraki, Koji Nonobe, and Mutsunori Yagiura, editors. *Fifth Metaheuristics International Conference (MIC2003) – Metaheuristics: Progress as Real Problem Solvers*, Kyoto International Conference Hall, Kyoto, Japan, August 25-28, 2003, volume 32 of *Operations Research/-Computer Science Interfaces*. Springer, Berlin Heidelberg New York, ISBN: 0-387-25382-3. Published in 2005. See `http://www-or.amp.i.kyoto-u.ac.jp/mic2003/` [accessed 2007-09-12].

[173] Mauricio G. C. Resende and Jorge Pinho de Sousa, editors. *4th Metaheuristics International Conference (MIC2001) – Metaheuristics: Computer Decision-Making*, Porto, Portugal, July 16-20, 2001, volume 86 of *Applied Optimization*, Boston, USA. Kluwer Academic Publishers, ISBN: 1-4020-7653-3. Published in November 30, 2003. See `http://paginas.fe.up.pt/~gauti/MIC2001/` [accessed 2007-09-12].

[174] Celso C. Ribeiro and Pierre Hansen, editors. *3rd Metaheuristics International Conference (MIC'99) – Essays and Surveys in Metaheuristics*, Angra dos Reis, Brazil, July 19-23, 1999, volume 15 of *Operations Research/Computer Science Interfaces*, Boston, USA. Kluwer Academic Publishers, ISBN: 0-7923-7520-3. Published in 2001.

[175] 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*, Sophia Antipolis, France, July 21-24, 1997, Boston, USA. Kluwer Academic Publishers, ISBN: 0-7923-8369-9. Published in 1998.

[176] H. Osman Ibrahim and James P. Kelly, editors. *1st Metaheuristics International Conference (MIC'95) – Meta-Heuristics: The Theory and Applications*, Breckenridge, Colorado, USA, July 22-26, 1995, Boston, USA. Kluwer Academic Publishers, ISBN: 0-7923-9700-2. Published in 1996.

[177] 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, ISBN: 354020637X, 978-3540206378, January 2004.

[178] David Corne, Marco Dorigo, and Fred Glover, editors. *New Ideas in Optimisation*. McGraw-Hill's Advanced Topics In Computer Science Series. McGraw-Hill Education, Maidenhead, UK, England, ISBN: 0077095065, 978-0077095062, May 1999.

[179] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, ISBN: 0195099710, January 1996.

[180] 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, Bristol, New York, ringbound edition, ISBN: 0750303921, April 1997.

[181] Thomas Bäck, Ulrich Hammel, and Hans-Paul Schwefel. Evolutionary computation: comments on the history and current state. *IEEE Transactions on Evolutionary Computation*, 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].

[182] Ernst Mayr. *The Growth of Biological Thought*. Harvard University Press, ISBN: 0-674-36446-5, 1982.

[183] 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. supervisors: Guszti Eiben and Han La Poutré. Online available at `http://citeseer.ist.psu.edu/128231.html` [accessed 2007-08-24].

[184] 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].

[185] 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].

[186] Sanaz Mostaghim. *Multi-objective Evolutionary Algorithms: Data structures, Convergence and, Diversity*. PhD thesis, Fakultät für Elektrotechnik, Informatik und Mathematik, Universität Paderborn, Deutschland (Germany), 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].

[187] Mark Ridley. *Evolution*. Blackwell Publishing Limited, eighth (october 1, 2003) edition, ISBN: 978-1405103459, 1996.

[188] 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].

[189] Carlos Artemio Ceollo Coello. A short tutorial on evolutionary multiobjective optimization. In *First International Conference on Evolutionary Multi-Criterion Optimization*, 2001, pages 21–40. See proceedings [268]. Online available at `http://citeseer.ist.psu.edu/coellocoello01short.html` [accessed 2007-07-29].

[190] Nadia Nedjah, Enrique Alba, and Luiza De Macedo Mourelle, editors. *Parallel Evolutionary Computations*, volume 22/2006 of *Studies in Computational Intelligence*. Springer Berlin / Heidelberg, ISBN: 9783540328377, ISSN: 1860-949X (Print) 1860-9503 (Online), June 2006.

[191] 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*, 2002, volume 1, pages 968–973. See proceedings [246].

[192] Adam Prügel-Bennett and Alex Rogers. Modelling ga dynamics. In Leila Kallel, Bart Naudts, and Alex Rogers, editors, *Theoretical Aspects of Evolutionary Computing*, Natural Computing Series, ISBN: 978-3-540-67396-5, pages 59–86. Springer, Berlin, Germany, 2001. Original:

1999, Online available at http://eprints.ecs.soton.ac.uk/13205/ [accessed 2007-07-28].

[193] 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].

[194] Thomas Bäck and Hans-Paul Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1–23, Spring 1993.

[195] 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, ISBN: 978-0471965886, chapter 29. John Wiley & Sons, Ltd., September 1998. Online available at http://www.wiley.com/legacy/wileychi/ecc/samples/sample10.pdf [accessed 2007-08-17].

[196] 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*, Como, Italy, 2000, volume 1, pages 463–469, New York, NY, USA. ACM Press, ISBN: 1-58113-240-9. Online available at http://doi.acm.org/10.1145/335603.335912 and http://citeseer.ist.psu.edu/335101.html [accessed 2007-08-24].

[197] 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*, 1990, pages 455–469. Published 1991. See proceedings [331] and also [198].

[198] 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 [197].

[199] Hans-Paul Schwefel. *Evolution and Optimum Seeking: The Sixth Generation*. John Wiley & Sons, Inc., Wiley Interscience, New York, NY, USA, ISBN: 0471571482, Har/Dsk: 978-0471571483, 1993. Har/Dsk edition, January 1995.

[200] Alex Rogers and Adam Prügel-Bennett. Modelling the dynamics of a steady-state genetic algorithm. In *Foundations of Genetic Algorithms 5*, 1998, pages 57–68. See proceedings [435]. 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].

[201] 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*, 1999, pages 167–170, Orlando, Florida, USA. Workshop part of GECCO 1999, see proceedings [314]. 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.

[202] 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.

[203] 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*, 2003, pages 813–824. See proceedings [304]. Online available at `http://citeseer.ist.psu.edu/591279.html` and `http://www.cs.uga.edu/~khaled/papers/385.pdf` [accessed 2007-07-28].

[204] 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*, July 1990, pages 94–101. See proceedings [439].

[205] 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*, 1989, pages 116–121. See proceedings [371]. 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].

[206] 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*, 2006, pages 143–150. See proceedings [298]. Online available at `http://doi.acm.org/10.1145/1143997.1144024` [accessed 2007-08-24].

[207] Dipti Srinivasan and Lily Rachmawati. An efficient multi-objective evolutionary algorithm with steady-state replacement model. In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, 2006, pages 715–722. See proceedings [298]. Online available at `http://doi.acm.org/10.1145/1143997.1144122` [accessed 2007-08-24].

[208] 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*, 1990, pages 69–93. See proceedings [439]. Online available at `http://www.cse.unr.edu/~sushil/class/gas/papers/Select.pdf` [accessed 2007-08-24].

[209] 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].

[210] Marco Laumanns, Eckart Zitzler, and Lothar Thiele. A unified model for multi-objective evolutionary algorithms with elitism. In *Proceed-*

ings of the 2000 Congress on Evolutionary Computation CEC00, 2000, pages 46–53. See proceedings [248]. Online available at `http://citeseer.ist.psu.edu/laumanns00unified.html` [accessed 2007-08-27].

[211] John Henry Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, ISBN: 978-026258111, 1975. Reprinted by MIT Press, April 1992.

[212] Karsten Weicker. *Evolutionäre Algorithmen*. Leitfäden der Informatik. B. G. Teubner GmbH, Stuttgart/Leipzig/Wiesbaden, ISBN: 3-519-00362-7, March 2002.

[213] Nicholas J. Radcliffe. Non-linear genetic representations. In *Parallel problem solving from nature 2*, 1992, pages 259–268. See proceedings [330]. Online available at `http://citeseer.ist.psu.edu/radcliffe92nonlinear.html` and `http://users.breathe.com/njr/papers/ppsn92.pdf` [accessed 2007-09-05].

[214] Nicolas J. Radcliffe. The algebra of genetic algorithms. *Annals of Maths and Artificial Intelligence*, 10, 1994. Also technical report TR92-11, 1992 from Edinburgh Parallel Computing Centre, University of Edinburgh, Kings 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].

[215] 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/radcliffe94algebra.html` [accessed 2007-07-28].

[216] Nicholas J. Radcliffe. Equivalence class analysis of genetic algorithms. *Complex Systems*, 5:183–205, 1991. Online available at `http://citeseer.ist.psu.edu/608745.html` [accessed 2007-07-28].

[217] Nicholas J. Radcliffe. Forma analysis and random respectful recombination. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, 1991, pages 222–229. See proceedings [445]. Online available at `http://users.breathe.com/njr/formaPapers.html` [accessed 2007-07-28].

[218] Nicholas J. Radcliffe and Patrick David Surry. Fitness variance of formae and performance prediction. In *Foundations of Genetic Algorithms 3*, 1994, pages 51–72. See proceedings [437]. Online available at `http://citeseer.comp.nus.edu.sg/radcliffe94fitness.html` [accessed 2007-07-28].

[219] Michael D. Vose. Generalizing the notion of schema in genetic algorithms. *Artificial Intelligence*, 50(3):385–396, 1991.

[220] 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*, 1989, pages 20–27. See proceedings [371].

[221] John J. Grefenstette. Conditions for implicit parallelism. In *Foundations of genetic algorithms*, 1990, pages 252–261. See proceedings [439]. Online available at `http://citeseer.ist.psu.edu/41022.html` [accessed 2007-07-29].

[222] 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].

[223] Michael D. Vose and Alden H. Wright. Form invariance and implicit parallelism. *Evolutionary Computation*, 9(3):355–370, 2001. Online available at `http://citeseer.ist.psu.edu/610097.html` [accessed 2007-07-29].

[224] 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*, 1999, pages 115–116. See proceedings [314]. Online available at `http://citeseer.ist.psu.edu/208779.html` [accessed 2007-08-24].

[225] Mihai Oltean. Evolving evolutionary algorithms for function optimization. In *Proceedings of the 5th International Workshop on Frontiers in Evolutionary Algorithms*, 2003, pages 295–298. See proceedings [291]. 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].

[226] Mitchell A. Potter and Kenneth A. 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*, 1994, pages 249–257. See proceedings [329]. Online available at `http://citeseer.ist.psu.edu/potter94cooperative.html` [accessed 2007-08-24].

[227] Peter A. N. Bosman and Dirk Thierens. A thorough documentation of obtained results on real-valued continuous and combinatorial multiobjective 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].

[228] 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*, Atlanta, Georgia, 1997, pages 1–6. Goose Pond

Press, ISBN: 905-6996-150. Online available at http://alexf04.maclisp.org/Fukunaga-Stechert-IEA-1997.pdf and http://de.scientificcommons.org/17874369 [accessed 2007-08-24].

[229] 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*, Seattle, Washington, August 2006. Tax Stats. Internal Revenue Service. United States Department of the Treasury. 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].

[230] 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, September 28, 2007. 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].

[231] 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*, pages 205–224. John Wiley and Sons, Chichester, 1998. 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].

[232] Charles L. Karr and Eric Wilson. A self-tuning evolutionary algorithm applied to an inverse partial differential equation. *Applied Intelligence*, 19:147–155, November 2003. Online available at http://www.springerlink.com/content/x7t26442h41q0221/fulltext.pdf and http://dx.doi.org/10.1023/A:1026097605403 [accessed 2007-08-27].

[233] 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*, 2005, pages 2157–2163. See proceedings [299]. Online available at http://doi.acm.org/10.1145/1068009.1068364 [accessed 2007-08-27].

[234] 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*, 101:7215–7222, May 2004. 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].

[235] 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*, 2000, pages 367–381. See proceedings [287]. Online available at http://citeseer.ist.psu.edu/marchiori00evolutionary.html [accessed 2007-08-27].

[236] C. S. Chang and Chung Min Kwan. Evaluation of evolutionary algorithms for multi-objective train schedule optimization. In *AI 2004: Advances in Artificial Intelligence*, 2004, volume 3339/2004 of *Lecture Notes in Artificial Intelligence, subseries of Lecture Notes in Computer Science (LNCS)*, pages 803–815. Springer-Verlag, ISBN: 978-3-540-24059-4, ISSN: 0302-9743 (Print) 1611-3349 (Online).

[237] 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)*, 2005, volume 2, pages 987–994. See proceedings [243]. Online available at http://www.lania.mx/~ccoello/EMOO/kwan05.pdf.gz [accessed 2007-08-27].

[238] 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)*, 2004, pages 151–168. 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].

[239] Bogdan Filipič and Jurij Šilc, editors. *Proceedings of the Second International Conference on Bioinspired Optimization Methods and their Application, BIOMA 2006*, Jožef Stefan International Postgraduate School, Ljubljana, Slovenia, October 9-10, 2006. Jožef Stefan Institute, ISBN: 978-961-6303-81-1.

[240] Bogdan Filipič and Jurij Šilc, editors. *Proccedings of the International Conference on Bioinspired Optimization Methods and their Applications (BIOMA 2004)*, Jožef Stefan International Postgraduate School, Ljubljana, Slovenia, October 11-12, 2004. Jožef Stefan Institute.

[241] *Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2007*, Swissôtel The Stamford, Singapore, September25-28, 2007, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA. IEEE Press. See http://cec2007.nus.edu.sg/ [accessed 2007-08-06].

[242] 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 2000*, Vancouver, BC, Canada,

July 16-21, 2006, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA. IEEE Press, ISBN: 0-7803-9487-9.

[243] 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*, Edinburgh, Scotland, UK, September 2-5, 2005, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA. IEEE Press, ISBN: 0-7803-9363-5.

[244] *Proceedings of the IEEE Congress on Evolutionary Computation, CEC2004*, Portland, Oregon, June 20-23, 2004, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA. IEEE Press, ISBN: 0-7803-8515-2.

[245] Ruhul Sarker, Robert Reynolds, Hussein Abbass, Kay Chen Tan, Bob McKay, Daryl Essam, and Tom Gedeon, editors. *Proceedings of the IEEE Congress on Evolutionary Computation, CEC2003*, Canberra, Australia, December 8-12, 2003, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA. IEEE Press, ISBN: 0-7803-7804-0. CEC 2003 - A joint meeting of the IEEE, the IEAust, the EPS, and the IEE.

[246] 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*, Honolulu, HI, USA, May 12-17, 2002, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA. IEEE Press, 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).

[247] *Proceedings of the IEEE Congress on Evolutionary Computation, CEC2001*, COEX, World Trade Center, 159 Samseong-dong, Gangnam-gu, Seoul, Korea, May 27-30, 2001, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA. IEEE Press, ISBN: 0-7803-6658-1. CEC-2001 - A joint meeting of the IEEE, Evolutionary Programming Society, Galesia, and the IEE. IEEE Catalog Number: 01TH8546C.

[248] *Proceedings of the IEEE Congress on Evolutionary Computation, CEC00*, La Jolla Marriott Hotel, La Jolla, California, USA, July 6-9, 2000, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA. IEEE Press, ISBN: 0-7803-6375-2. CEC-2000 - A joint meeting of the IEEE, Evolutionary Programming Society, Galesia, and the IEE. IEEE Catalog Number: 00TH8512, Library of Congress Number: 00-018644.

[249] Peter John Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, editors. *Proceedings of the IEEE Congress on Evolutionary Computation, CEC99*, Mayflower Hotel, Washington D.C., USA, July 6-9, 1999, volume 1-3, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331, USA. IEEE Press, ISBN: 0-7803-5536-9, 0-

7803-5537-7. CEC-99 - A joint meeting of the IEEE, Evolutionary Programming Society, Galesia, and the IEE. Library of Congress Number: 99-61143.

[250] *The 1998 IEEE International Conference on Evolutionary Computation, Proceedings of IEEE World Congress on Computational Intelligence, CEC98*, Anchorage, Alaska, USA, May 4-9, 1998. IEEE Press. See http://ieeexplore.ieee.org/servlet/opac?punumber=5621 [accessed 2007-09-06].

[251] Thomas Bäck, Zbigniew Michalewicz, and X. Yao, editors. *IEEE International Conference on Evolutionary Computation, CEC97*, Indianapolis, IN, USA, April 13-16, 1997. IEEE Press, Piscataway, NJ, ISBN: 0-7803-3949-5. See http://ieeexplore.ieee.org/servlet/opac?punumber=4619 [accessed 2007-09-06].

[252] *Proceedings of IEEE International Conference on Evolutionary Computation, CEC96*, Nagoya, Japan, May 20-22, 1996. IEEE Press, Piscataway, NJ. See http://ieeexplore.ieee.org/servlet/opac?punumber=3838 [accessed 2007-09-06].

[253] *IEEE International Conference on Evolutionary Computation, ICEC '95*, Perth, Australia, November 29-December 1, 1995, volume 1-2. IEEE Press. 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.

[254] Zbigniew Michalewicz, J. David Schaffer, Hans-Paul Schwefel, David B. Fogel, and H. Kitano, editors. *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, Orlando, Florida, USA, June 27-29, 1994. IEEE Press, Piscataway, New Jersey, ISBN: 0-7803-1899-4. See http://ieeexplore.ieee.org/servlet/opac?punumber=1125 [accessed 2007-09-06].

[255] Jürgen Branke, Kalyanmoy Deb, Kaisa Miettinen, and Roman Slowinski, editors. *Practical Approaches to Multi-Objective Optimization*, Dagstuhl, Germany, December 10-15, 2006, number 06501 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany IBFI, ISSN: 1862-4405. 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].

[256] Jürgen Branke, Kalyanmoy Deb, Kaisa Miettinen, and Ralph E. Steuer, editors. *Practical Approaches to Multi-Objective Optimization*, Dagstuhl, Germany, November 7-12, 2004, number 04461 in Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany IBFI, ISSN: 1862-4405. 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].

[257] *Proceedings of the 7th International Conference on Artificial Evolution, Evolution Artificielle, EA 2005*, Tours, France, October 29-31, 2007, ISSN: 0302-9743 (Print) 1611-3349 (Online). See http://ea07.hant.li.univ-tours.fr/ [accessed 2007-09-09].

[258] El-Ghazali Talbi, Pierre Liardet, Pierre Collet, Evelyne Lutton, and Marc Schoenauer, editors. *Revised Selected Papers of the 8th International Conference on Artificial Evolution, Evolution Artificielle, EA 2005*, Lille, France, October 26-28, 2005, volume 3871 of *Lecture Notes in Computer Science (LNCS)*. Springer Berlin/Heidelberg, ISBN: 3-540-33589-7, ISSN: 0302-9743 (Print) 1611-3349 (Online). Published in 2006.

[259] 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*, Marseilles, France, October 27-30, 2003, volume 2936 of *Lecture Notes in Computer Science (LNCS)*. Springer Berlin/Heidelberg, ISBN: 3-540-21523-9, ISSN: 0302-9743 (Print) 1611-3349 (Online). Published in 2003.

[260] 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*, Le Creusot, France, October 29-31, 2001, volume 2310 of *Lecture Notes in Computer Science (LNCS)*. Springer Berlin/Heidelberg, ISBN: 3-540-43544-1, ISSN: 0302-9743 (Print) 1611-3349 (Online). Published in 2002.

[261] 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*, Dunkerque, France, November 3-5, 1999, volume 1829 of *Lecture Notes in Computer Science (LNCS)*. Springer Berlin/Heidelberg, ISBN: 3-540-67846-8, ISSN: 0302-9743 (Print) 1611-3349 (Online). Published in 2000.

[262] 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*, Nîmes, France, October 22-24, 1997, volume 1363 of *Lecture Notes in Computer Science (LNCS)*. Springer Berlin/Heidelberg, ISBN: 3-540-64169-6, ISSN: 0302-9743 (Print) 1611-3349 (Online). Published in May 1998.

[263] 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*, Brest, France, September 4-6, 1995, volume 1063 of *Lecture Notes in Computer Science (LNCS)*. Springer Berlin/Heidelberg, ISBN: 3-540-61108-8, ISSN: 0302-9743 (Print) 1611-3349 (Online). Published in 1996.

[264] Jean-Marc Alliot, Evelyne Lutton, and Marc Schoenauer, editors. *European Conference on Artificial Evolution, AE 94*, Toulouse, France, 1994. Cepadues, ISBN: 9782854284119. Published in January 1995.

[265] 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)*, Matsushima/Sendai, Japan, March 5-8, 2007, volume 4403/2007 of *Lecture Notes in Computer Science (LNCS)*. Springer-Verlag, Berlin, ISBN: 978-3-540-70927-5, ISSN: 0302-9743 (Print) 1611-3349 (Online). See `http://www.is.doshisha.ac.jp/emo2007/` [accessed 2007-09-11].

[266] Carlos A Coello Coello, Arturo Hernández Aguirre, and Eckart Zitzler, editors. *Proceedings of the Third International Conference on Evolutionary Multi-Criterion Optimization (EMO'05)*, Guanajuato, Mexico, March 9-11, 2005, volume 3410 of *Lecture Notes in Computer Science (LNCS)*. Springer-Verlag, Berlin, ISBN: 978-3-540-24983-2, ISSN: 0302-9743 (Print) 1611-3349 (Online).

[267] 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)*, Faro, Portugal, April 8-11, 2003, volume 2632/2003 of *Lecture Notes in Computer Science (LNCS)*. Springer-Verlag, Berlin, ISBN: 978-3-540-01869-8, ISSN: 0302-9743 (Print) 1611-3349 (Online).

[268] 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)*, Zurich, Switzerland, March 7-9, 2001, volume 1993/2001 of *Lecture Notes in Computer Science (LNCS)*. Springer-Verlag, Berlin, ISBN: 3-540-41745-1, ISSN: 0302-9743 (Print) 1611-3349 (Online). See `http://www.tik.ee.ethz.ch/emo/` [accessed 2007-09-11].

[269] *Proceedings of the Seventh Conference on Evolutionary and Deterministic Mehtods for Design, Optimization and Control with Applications to Indutsrial and Societal Problems, EUROGEN2007*, University of Jyväskylä, Jyväskylä, Finland, June 11-13, 2007. See `http://www.mit.jyu.fi/scoma/Eurogen2007/` [accessed 2007-09-16].

[270] 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*, TU München, Munich, Germany, September 12-14, 2005. TU München. in association with ECCOMAS and ERCOFTAC. See `http://www.flm.mw.tum.de/EUROGEN05/` [accessed 2007-09-16].

[271] *Proceedings of the EUROGEN2003 Conference: Evolutionary Methods for Design Optimization and Control with Applications to Indus-*

trial Problems, Barcelona, Spain, September 15-17, 2003, ISBN: 84-95999-33-1. Published on CD. See `http://congress.cimne.upc.es/eurogen03/` [accessed 2007-09-16].

[272] 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*, Athens, Greece, September 19-21, 2001. International Center for Numerical Methods in Engineering (Cmine), Gran Capitan s/n, 08034 Barcelona, Spain, ISBN: 84-89925-97-6. Published in 2002. See `http://www.mech.ntua.gr/~eurogen2001` [accessed 2007-09-16].

[273] *European Short Course on Genetic Algorithms and Evolution Strategies, EUROGEN 1999*, Univeristy of Jyväskylä, Jyväskylä, Finland, May 30-June 3, 1999.

[274] D. Quagliarella, J. Périaux, C. Poloni, and G. Winter, editors. *Genetic Algorithms and Evolution Strategy in Engineering and Computer Science: Proceedings of the 2nd European Short Course on Genetic Algorithms and Evolution Strategies, EUROGEN 1997*, Triest, Italy, November 28-December 4, 1997, Recent Advances in Industrial Applications. Wiley & Sons, ISBN: 978-0471977100.

[275] *Genetic Algorithms in Engineering and Computer Science, European Short Course on Genetic Algorithms and Evolution Strategies, EUROGEN 1995*, Las Palmas de Gran Canaria, Spain, 1995. John Wiley & Sons.

[276] Carlos Cotta and Jano I. van Hemert, editors. *Proceedings of the 7th European Conference on Evolutionary Computation in Combinatorial Optimization, EvoCOP 2007*, Valencia, Spain, April 11-13, 2007, volume 4446/2007 of *Lecture Notes in Computer Science (LNCS)*. Springer, ISBN: 978-3-540-71614-3, ISSN: 0302-9743 (Print) 1611-3349 (Online).

[277] Jens Gottlieb and Günther R. Raidl, editors. *Proceedings of the 6th European Conference on Evolutionary Computation in Combinatorial Optimization, EvoCOP 2006*, Budapest, Hungary, April 10-12, 2006, volume 3906/2006 of *Lecture Notes in Computer Science (LNCS)*. Springer, ISBN: 3-540-33178-6, ISSN: 0302-9743 (Print) 1611-3349 (Online).

[278] Günther R. Raidl and Jens Gottlieb, editors. *Proceedings of the 5th European Conference on Evolutionary Computation in Combinatorial Optimization, EvoCOP 2005*, Lausanne, Switzerland, March 30-April 1, 2005, volume 3448/2005 of *Lecture Notes in Computer Science (LNCS)*. Springer, ISBN: 3-540-25337-8, ISSN: 0302-9743 (Print) 1611-3349 (Online).

[279] Jens Gottlieb and Günther R. Raidl, editors. *Proceedings of the 4th European Conference on Evolutionary Computation in Combinatorial Optimization, EvoCOP 2004*, Coimbra, Portugal, April 5-7, 2004, volume

3004/2004 of *Lecture Notes in Computer Science (LNCS)*. Springer, ISBN: 3-540-21367-8, ISSN: 0302-9743 (Print) 1611-3349 (Online).

[280] 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. *Proceedings of Applications of Evolutionary Computing, EvoWorkshop 2003: EvoBIO, EvoCOP, EvoIASP, EvoMUSART, EvoROB, and EvoSTIM*, University of Essex,Essex, UK, April 14-16, 2003, volume 2611/2003 of *Lecture Notes in Computer Science (LNCS)*. Springer, ISBN: 3-540-00976-0, ISSN: 0302-9743 (Print) 1611-3349 (Online).

[281] Stefano Cagnoni, Jens Gottlieb, Emma Hart, Martin Middendorf, and Günther R. Raidl, editors. *Proceedings of Applications of Evolutionary Computing, EvoWorkshops 2002: EvoCOP, EvoIASP, EvoSTIM/EvoPLAN*, Kinsale, Ireland, April 3-4, 2002, volume 2279/2002 of *Lecture Notes in Computer Science (LNCS)*. Springer, ISBN: 3-540-43432-1, ISSN: 0302-9743 (Print) 1611-3349 (Online).

[282] Egbert J. W. Boers, Jens Gottlieb, Pier Luca Lanzi, Robert E. Smith, Stefano Cagnoni, Emma Hart, Günther R. Raidl, and H. Tijink, editors. *Proceedings of Applications of Evolutionary Computing, EvoWorkshops 2001: EvoCOP, EvoFlight, EvoIASP, EvoLearn, and EvoSTIM*, Lake Como, Milan, Italy, April 18-20, 2001, volume 2037/2001 of *Lecture Notes in Computer Science (LNCS)*. Springer, ISBN: 3-540-41920-9, ISSN: 0302-9743 (Print) 1611-3349 (Online).

[283] 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, EvoWorkshops 2007: EvoCoMnet, EvoFIN, EvoIASP, EvoINTERACTION, EvoMUSART, EvoSTOC and EvoTransLog*, Valencia, Spain, April 11-13, 2007, volume 4448/2007 of *Lecture Notes in Computer Science (LNCS)*. Springer, ISBN: 978-3-540-71804-8, ISSN: 0302-9743 (Print) 1611-3349 (Online).

[284] 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. *Proceedings of Applications of Evolutionary Computing, EvoWorkshops 2006: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoINTERACTION, EvoMUSART, and EvoSTOC*, Budapest, Hungary, April 10-12, 2006, volume 3907/2006 of *Lecture Notes in Computer Science (LNCS)*. Springer, ISBN: 3-540-33237-5, ISSN: 0302-9743 (Print) 1611-3349 (Online).

[285] 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. *Pro-*

ceedings of Applications of Evolutionary Computing, EvoWorkshops 2005: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoMUSART, and EvoSTOC, Lausanne, Switzerland, March 30-April 1, 2005, volume 3449/2005 of *Lecture Notes in Computer Science (LNCS)*. Springer, ISBN: 3-540-25396-3, 978-3-540-25396-9, ISSN: 0302-9743 (Print) 1611-3349 (Online).

[286] 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. *Proceedings of Applications of Evolutionary Computing, EvoWorkshops 2004: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoMUSART, and EvoSTOC*, Coimbra, Portugal, April 5-7, 2004, volume 3005/2004 of *Lecture Notes in Computer Science (LNCS)*. Springer, ISBN: 3-540-21378-3, ISSN: 0302-9743 (Print) 1611-3349 (Online).

[287] 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. *Proceedings of Real-World Applications of Evolutionary Computing, EvoWorkshops 2000: EvoIASP, EvoSCONDI, EvoTel, EvoSTIM, EvoROB, and EvoFlight*, Edinburgh, Scotland, UK, April 17, 2000, volume 1803/200 of *Lecture Notes in Computer Science (LNCS)*. Springer, ISBN: 3-540-67353-9, ISSN: 0302-9743 (Print) 1611-3349 (Online).

[288] 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*, Göteborg, Sweden, May 26-27, 1999, volume 1596/1999 of *Lecture Notes in Computer Science (LNCS)*. Springer, ISBN: 3-540-65837-8, ISSN: 0302-9743 (Print) 1611-3349 (Online).

[289] Phil Husbands and Jean-Arcady Meyer, editors. *Proceedings of the First European Workshop on Evolutionary Robotics, EvoRbot98*, Paris, France, April 16-17, 1998, volume 1468/1998 of *Lecture Notes in Computer Science (LNCS)*. Springer, ISBN: 3-540-64957-3, ISSN: 0302-9743 (Print) 1611-3349 (Online).

[290] *Proceedings of the Eighth Joint Conference on Information Science (JCIS 2005), Section: The Sixth International Workshop on Frontiers in Evolutionary Algorithms (FEA 2005)*, Salt Lake City Mariott City Center, Salt Lake City, Utah, USA, July 21-16, 2005. 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].

[291] 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)*, Em-

bassy Suites Hotel and Conference Center, Cary, North Carolina, USA, September 26-30, 2003. Workshop held in conjunction with Seventh Joint Conference on Information Sciences.

[292] 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)*, Research Triangle Park, North Carolina, USA, March 8-13, 2002. JCIS / Association for Intelligent Machinery, Inc., ISBN: 0-9707890-1-7. Workshop held in conjunction with Sixth Joint Conference on Information Sciences.

[293] 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)*, Atlantic City, NJ, USA, February 27-March 3, 2000. The Association for Intelligent Machinery. Workshop held in conjunction with Fifth Joint Conference on Information Sciences.

[294] *Proceedings of the Fourth Joint Conference on Information Science (JCIS 1998), Section: The Second International Workshop on Frontiers in Evolutionary Algorithms (FEA 1998)*, Research Triangle Park, North Carolina, USA, October 24-28, 1998, volume 2. The Association for Intelligent Machinery. Workshop held in conjunction with Fourth Joint Conference on Information Sciences.

[295] *Proceedings of the Third Joint Conference on Information Science (JCIS 1997), Section: The First International Workshop on Frontiers in Evolutionary Algorithms (FEA 1998)*, Sheraton Imperial Hotel & Convention Center, Research Triangle Park, North Carolina, USA, March 1-5, 1997. Workshop held in conjunction with Third Joint Conference on Information Sciences.

[296] Dirk Thierens, Hans-Georg Beyer, Josh Bongard, Jurgen Branke, John Andrew Clark, Dave Cliff, Clare Bates Congdon, Kalyanmoy Deb, Benjamin Doerr, Tim Kovacs, Sanjeev Kumar, Julian F. Miller, Jason Moore, Frank Neumann, Martin Pelikan, Riccardo Poli, Kumara Sastry, Kenneth Owen Stanley, Thomas Stutzle, Richard A Watson, and Ingo Wegener, editors. *Proceedings of Genetic and Evolutionary Computation Conference, GECCO 2007*, University College London, Gower Street, London WC 1E 6BT, London, UK, July 7-12, 2007. ACM Press, New York, NY, USA, ISBN: 978-1-59593-697-4. See also [297].

[297] Dirk Thierens, Hans-Georg Beyer, Josh Bongard, Jurgen Branke, John Andrew Clark, Dave Cliff, Clare Bates Congdon, Kalyanmoy Deb, Benjamin Doerr, Tim Kovacs, Sanjeev Kumar, Julian F. Miller, Jason Moore, Frank Neumann, Martin Pelikan, Riccardo Poli, Kumara Sastry, Kenneth Owen Stanley, Thomas Stutzle, Richard A Watson, and Ingo Wegener, editors. *Genetic and Evolutionary Computation*

Conference - Companion Material, GECCO 2007, University College London, Gower Street, London WC 1E 6BT, London, UK, July 7-12, 2007. ACM Press, New York, NY, USA, ISBN: 978-1-59593-698-1. See also [296], ACM Order Number 910071.

[298] Mike Cattolico, editor. *GECCO '06: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, Renaissance Seattle Hotel, 515 Madison Street, Seattle, Washington 98104, USA, July 8-12, 2006, New York, NY, USA. ACM Press, ISBN: 1-59593-186-4. ACM order number 910060.

[299] 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*, Loews L'Enfant Plaza Hotel, 480 L'enfant Plaza Sw, Washington, D.C. 20024, USA, June 25-29, 2005, Washington DC, USA. ACM Press, ISBN: 1-59593-010-8. 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 [300, 301].

[300] Hans-Georg Beyer and Una-May O'Reilly, editors. *Workshop Proceedings of Genetic and Evolutionary Computation Conference, GECCO 2005*, Loews L'Enfant Plaza Hotel, 480 L'enfant Plaza Sw, Washington, D.C. 20024, USA, June 25-29, 2005. ACM. See also [299, 301].

[301] Franz Rothlauf, editor. *Late Breaking Papers at Genetic and Evolutionary Computation Conference, GECCO 2005*, Loews L'Enfant Plaza Hotel, 480 L'enfant Plaza Sw, Washington, D.C. 20024, USA, June 25-29, 2005. See also [299, 300]. Also distributed on CD-ROM at GECCO-2005.

[302] 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*, Red Lion Hotel, 1415 5th Avenue, Seattle, WA 98101, USA, June 26-30, 2004, volume 3102/2004 of *Lecture Notes in Computer Science (LNCS)*. Springer Berlin/Heidelberg, ISBN: 978-3-540-22344-3, ISSN: 0302-9743 (Print) 1611-3349 (Online). see also [303, 660].

[303] 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*, Red Lion Hotel, 1415 5th Avenue, Seattle, WA 98101, USA, June 26-30, 2004, volume 3103/2004 of *Lecture Notes in Computer Science (LNCS)*. Springer Berlin/Heidelberg, ISBN: 978-3-540-22343-6, ISSN: 0302-9743 (Print) 1611-3349 (Online). see also [302, 660].

[304] 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 F. Miller, editors. *Proceedings of Genetic and Evolutionary Computation - GECCO 2003, Genetic and Evolutionary Computation Conference, Part I*, The Holiday Inn Chicago – Mart Plaza, 350 N. Orleans St., Chicago, IL 60654, USA, July 12-16, 2003, volume 2723/2003 of *Lecture Notes in Computer Science (LNCS)*. Springer Berlin/Heidelberg, ISBN: 978-3-540-40602-0, ISSN: 0302-9743 (Print) 1611-3349 (Online). see also [305, 661].

[305] 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 F. Miller, editors. *Proceedings of Genetic and Evolutionary Computation - GECCO 2003, Genetic and Evolutionary Computation Conference, Part II*, The Holiday Inn Chicago – Mart Plaza, 350 N. Orleans St., Chicago, IL 60654, USA, July 12-16, 2003, volume 2724/2003 of *Lecture Notes in Computer Science (LNCS)*. Springer Berlin/Heidelberg, ISBN: 978-3-540-40603-7, ISSN: 0302-9743 (Print) 1611-3349 (Online). see also [304, 661].

[306] 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*, The Roosevelt Hotel, 45th and Madison Avenue, New York, NY, USA, July 9-13, 2002. Morgan Kaufmann Publishers Inc., ISBN: 1-55860-878-8. see also [307, 309, 308].

[307] Erick Cantú-Paz, editor. *Late Breaking papers at the Genetic and Evolutionary Computation Conference (GECCO-2002)*, The Roosevelt Hotel, 45th and Madison Avenue, New York, NY, USA, July 9-13, 2002. AAAI. see also [306, 309, 308].

[308] Alwyn M. Barry, editor. *GECCO 2002: Proceedings of the Bird of a Feather Workshops, Genetic and Evolutionary Computation Confer-*

ence, The Roosevelt Hotel, 45th and Madison Avenue, New York, NY, USA, July 8, 2002. AAAI, 445 Burgess Drive, Menlo Park, CA 94025.

[309] Michael O'Neill and Conor Ryan, editors. *Grammatical Evolution Workshop (GEWS 2002)*, New York, NY, USA, July 9, 2002. Part of GECCO, see [306] and also `http://www.grammatical-evolution.com/gews2002/` [accessed 2007-09-10].

[310] 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)*, Holiday Inn Golden Gateway Hotel, San Francisco, California, USA, July 7-11, 2001. Morgan Kaufmann, ISBN: 978-1558607743. see also [311].

[311] Erik D. Goodman, editor. *Late Breaking Papers at Genetic and Evolutionary Computation Conference (GECCO '01)*, Holiday Inn Golden Gateway Hotel, San Francisco, California, USA, July 7-11, 2001. see also [310].

[312] 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)*, The Riviera Hotel and Casino, Las Vegas, Nevada, USA, July 8-12, 2000. Morgan Kaufmann, ISBN: 978-1558607088. see also [313].

[313] L. Darrell Whitley, editor. *Late Breaking Papers at Genetic and Evolutionary Computation Conference (GECCO '00)*, The Riviera Hotel and Casino, Las Vegas, Nevada, USA, July 8-12, 2000. see also [312].

[314] Wolfgang Banzhaf, Jason M. Daida, Agoston 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)*, Orlando, Florida, USA, July 13-17, 1999. Morgan Kaufmann, ISBN: 1-55860-611-4. see also [315].

[315] *Late Breaking Papers at Genetic and Evolutionary Computation Conference (GECCO 1999)*, Orlando, Florida, USA, July 13-17, 1999. see also [314].

[316] Bartłomiej Beliczyński, Andrzej Dzieliński, Marcin Iwanowski, and Bernardete Ribeiro, editors. *Proceedings of Adaptive and Natural Computing Algorithms, 8th International Conference, ICANNGA 2007, Part I*, Warsaw University of Technology, Warsaw, Poland, April 11-14, 2007, volume 4431/2007 of *Lecture Notes in Computer Science (LNCS)*. Springer Berlin Heidelberg New York, ISBN: 978-3-540-71589-4, ISSN: 0302-9743. see also [317], `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].

[317] Bartłomiej Beliczyński, Andrzej Dzieliński, Marcin Iwanowski, and Bernardete Ribeiro, editors. *Proceedings of Adaptive and Natural Computing Algorithms, 8th International Conference, ICANNGA*

2007, Part II, Warsaw University of Technology, Warsaw, Poland, April 11-14, 2007, volume 4432/2007 of *Lecture Notes in Computer Science (LNCS)*. Springer Berlin Heidelberg New York, ISBN: 978-3-540-71590-0, ISSN: 0302-9743. see also [316], `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].

[318] 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*, Department of Informatics Engineering, Faculty of Science and Technology, University of Coimbra, Coimbra, Portugal, March 21-23, 2005. Springer, Wien, ISBN: 978-3-211-24934-5 (Print) 978-3-211-27389-0 (Online). see `http://icannga05.dei.uc.pt/` [accessed 2007-08-31].

[319] David W. Pearson, Nigel C. Steele, and Rudolf F. Albrecht, editors. *Proceedings of the 6th International Conference on Artificial Neural Nets and Genetic Algorithms*, Roanne, France, 2003. Springer Verlag, ISBN: 978-3-211-00743-3.

[320] Vera Kurkova, Nigel C. Steele, Roman Neruda, and Miroslav Karny, editors. *Proceedings of the 5th International Conference on Artificial Neural Networks and Genetic Algorithms*, Prague, Czech Republic, 2001, Berlin. Springer Verlag, ISBN: 978-3-211-83651-4.

[321] 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*, Portoroz, Slovenia, 1999. Springer, ISBN: 978-3-211-83364-3.

[322] George D. Smith, Nigel C. Steele, and Rudolf F. Albrecht, editors. *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms*, Norwich, England, U.K., 1997, ISBN: 978-3-211-83087-1.

[323] 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*, University of Iceland, Reykjavik, Iceland, September 9-13, 2006, volume 4193/2006 of *Lecture Notes in Computer Science (LNCS)*. Springer, ISBN: 3-540-38990-3, ISSN: 0302-9743 (Print) 1611-3349 (Online). See `http://ppsn2006.raunvis.hi.is/` [accessed 2007-09-05].

[324] 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*, Birmingham, UK, September 18-22, 2004, volume 3242/2004 of *Lecture Notes in Computer Science (LNCS)*. Springer,

ISBN: 3-540-23092-0, ISSN: 0302-9743 (Print) 1611-3349 (Online). See http://events.cs.bham.ac.uk/ppsn04/ [accessed 2007-09-05].

[325] 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*, Granada, Spain, September 7-11, 2002, volume 2439/2002 of *Lecture Notes in Computer Science (LNCS)*. Springer, ISBN: 3-540-44139-5, ISSN: 0302-9743 (Print) 1611-3349 (Online). See http://ppsn2002.ugr.es/index.html [accessed 2007-09-05].

[326] 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*, Paris, France, September 18-20, 2000, volume 1917/2000 of *Lecture Notes in Computer Science (LNCS)*. Springer, ISBN: 3-540-41056-2, ISSN: 0302-9743 (Print) 1611-3349 (Online).

[327] Agoston 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*, Amsterdam, The Netherlands, September 27-30, 1998, volume 1498/1998 of *Lecture Notes in Computer Science (LNCS)*. Springer, ISBN: 3-540-65078-4, ISSN: 0302-9743 (Print) 1611-3349 (Online).

[328] 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*, Berlin, Germany, September 22-24, 1996, volume 1141/1996 of *Lecture Notes in Computer Science (LNCS)*. Springer, ISBN: 3-540-61723-X, ISSN: 0302-9743 (Print) 1611-3349 (Online). See http://ls11-www.informatik.uni-dortmund.de/PPSN/ppsn4/ppsn4.html [accessed 2007-09-05].

[329] 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*, Jerusalem, Israel, October 9-14, 1994, volume 866/1994 of *Lecture Notes in Computer Science (LNCS)*. Springer, ISBN: 3-540-58484-6, ISSN: 0302-9743 (Print) 1611-3349 (Online). See http://ls11-www.informatik.uni-dortmund.de/PPSN/ppsn3/ppsn3.html [accessed 2007-09-05].

[330] Reinhard Männer and Bernard Manderick, editors. *Proceedings of Parallel Problem Solving from Nature 2, PPSN II*, Brussels, Belgium, September 28-30, 1992. Elsevier. See http://ls11-www.informatik.uni-dortmund.de/PPSN/ppsn2/ppsn2.html [accessed 2007-09-05].

[331] Hans-Paul Schwefel and Reinhard Männer, editors. *Proceedings of the 1st Workshop on Parallel Problem Solving from Nature, PPSN I*, FRG, Dortmund, Germany, October 1-3, 1990, volume 496/1991 of *Lecture Notes in Computer Science (LNCS)*. Springer, ISBN: 3-540-54148-9, ISSN: 0302-9743 (Print) 1611-3349 (Online). Published 1991. See `http://ls11-www.informatik.uni-dortmund.de/PPSN/ppsn1/ppsn1.html` [accessed 2007-09-05].

[332] David B. Fogel, editor. *Evolutionary Computation: The Fossil Record*. Wiley-IEEE Press, ISBN: 978-0780334816, May 1, 1998.

[333] Peter J. Bentley, editor. *Evolutionary Design by Computers*. Morgan Kaufmann, San Francisco, USA, ISBN: 1-55860-605-X, June 15, 1999. See also `http://www.cs.ucl.ac.uk/staff/P.Bentley/evdes.html` [accessed 2007-10-03].

[334] Franz Rothlauf. *Representations for Genetic and Evolutionary Algorithms*. Physica-Verlag, ISBN: 978-3790814965, August 2002. Foreword by David E. Goldberg.

[335] Wolfgang Banzhaf and Frank H. Eeckman, editors. *Evolution and Biocomputation – Computational Models of Evolution*, Monterey, California, USA, July 1992, volume 899 of *Lecture Notes in Computer Science (LNCS)*. Springer-Verlag, London, UK, ISBN: 3-540-59046-3, 0-387-59046-3. Published 1995.

[336] Shu-Heng Chen, editor. *Evolutionary Computation in Economics and Finance*, volume 100 of *Studies in Fuzziness and Soft Computing*. Physica-Verlag Heidelberg, ISBN: 3790814768, August 5, 2002.

[337] Peter A. N. Bosman and Dirk Thierens. Multi-objective optimization with diversity preserving mixture-based iterated density estimation evolutionary algorithms. *International Journal Approx. Reasoning*, 31(3):259–289, 2002.

[338] Peter A. N. Bosman and Dirk Thierens. The naive MidEa: A baseline multi-objective ea. In *Proceedings of the Third International Conference on Evolutionary Multi-Criterion Optimization*, 2005, pages 428–442. See proceedings [266]. Online available at `http://springerlink.metapress.com/content/m01x9t6l2bwd/?p_o=20` [accessed 2007-08-22].

[339] 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].

[340] Jinyun Ke, Mieko Ogura, and William S-Y. Wang. Modeling evolution of sound systems with genetic algorithm. *Computational Linguistics*, 29(1):1–18, 2003. Online available at `http://www.isrl.uiuc.edu/~amag/langev/paper/ke_GAModelSound.html` [accessed 2007-07-29].

[341] 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*, 1996, pages 437–455. See proceedings [436].

[342] David E. Goldberg and Jon Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application*, 1987, pages 41–49. See proceedings [446].

[343] Kalyanmoy Deb. Genetic algorithms in multimodal function optimization. Master's thesis, The Clearinghouse for Genetic algorithms, University of Alabama, Tuscaloosa, 1989. TCGA Report No. 89002.

[344] 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].

[345] 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*, 1994, volume 1, pages 82–87. See proceedings [254]. 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].

[346] N. Srinivas and Kalyanmoy Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1995. Online available at `http://citeseer.ist.psu.edu/srinivas94multiobjective.html` [accessed 2007-07-28].

[347] 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*, 2000, pages 849–858. See proceedings [326]. Online available at `http://citeseer.ist.psu.edu/deb00fast.html` [accessed 2007-07-28].

[348] 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. See collection [177].

[349] 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].

[350] 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].

[351] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In *Evolutionary Methods for Design, Optimisation and Control with Application to Industrial Problems. Proceedings of the EUROGEN2001 Conference*, 2001, pages 95–100. See proceedings [272]. Online available at http://citeseer.ist.psu.edu/514031.html and http://de.scientificcommons.org/526554 [accessed 2007-07-29].

[352] 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].

[353] 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)*, 2001, page 781. See proceedings [310]. 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 [accessed http://citeseer.ist.psu.edu/501944.html]2007-08-25.

[354] A. F. Bissell. Ordered random selection without replacement. *Applied Statistics*, 35(1):73–75, 1986.

[355] David E. Goldberg. A note on Boltzmann tournament selection for genetic algorithms and population-oriented simulated annealing. TCGA Report 90003, Department of Engineering Mechanics, University of Alabama, 1990.

[356] Brad L. Miller and David E. Goldberg. Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, 9:193–212, 1995. Online available at http://citeseer.ist.psu.edu/86198.html [accessed 2007-07-28].

[357] Tobias Blickle and Lothar Thiele. A mathematical analysis of tournament selection. In *Proceedings of the Sixth International Conference on Genetic Algorithms*, 1995, pages 9–16. See proceedings [443]. Online available at http://citeseer.ist.psu.edu/blickle95mathematical.html [accessed 2007-07-29].

[358] David E. Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Proceedings of Foundations of Genetic Algorithms*, 1990, pages 69–93. See proceedings [439]. Online available at http://www.cse.unr.edu/~sushil/class/gas/papers/Select.pdf [accessed 2007-07-29].

[359] 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)*, November 28-30, 2005, pages 1115–1121, Washington, DC, USA. IEEE Computer Society, ISBN: 0-7695-2504-0-02.

[360] L. Darell Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4(2):65–85, June 1994. 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 [1316].

[361] 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*, Czestochowa, Poland, September 2003, volume 3019/2004 of *Lecture Notes in Computer Science (LNCS)*, pages 560–565. Springer Berlin / Heidelberg, ISBN: 978-3-540-21946-0, ISSN: 0302-9743 (Print) 1611-3349 (Online). Published 2004.

[362] 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].

[363] J. David Schaffer. *Multiple Objective Optimization with Vector Evaluated Genetic Algorithms*. PhD thesis, Vanderbilt University, 1984.

[364] J. David Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms and Their Applications*, 1985, pages 93–100. See proceedings [447].

[365] Kalyanmoy Deb and Tushar Goel. Controlled elitist non-dominated sorting genetic algorithms for better convergence. In *EMO '01: Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization*, 2001, pages 67–81. See proceedings [268]. Online available at `http://citeseer.ist.psu.edu/450660.html` and `http://www.lania.mx/~ccoello/EMOO/deb01a.ps.gz` [accessed 2007-08-29].

[366] David W. Corne, Joshua D. Knowles, and Martin J. Oates. The Pareto Envelope-based Selection Algorithm for Multiobjective Optimization. In *Proceedings of the Parallel Problem Solving from Nature VI Conference*, 2000, pages 839–848. See proceedings [326]. Online available at

http://www.lania.mx/~ccoello/EMOO/corne00.ps.gz and http://citeseer.ist.psu.edu/386119.html [accessed 2007-07-29].

[367] David W. Corne, Nick R. Jerram, Joshua D. Knowles, and Martin J. Oates. PESA-II: Region-based selection in evolutionary multiobjective optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, 2001, pages 283–290. See proceedings [310]. Online available at http://citeseer.ist.psu.edu/corne01pesaii.html and http://dbkgroup.org/knowles/GA247.ps.gz [accessed 2007-07-29].

[368] Abdullah Konak, David W. Coit, and Alice E. Smith. Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering & System Safety*, 91(9):992–1007, September 2006. Online available at http://www.rci.rutgers.edu/~coit/RESS_2006_MOGA.pdf and http://www.soe.rutgers.edu/ie/research/working_paper/paper05-008.pdf [accessed 2007-08-29].

[369] 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)*, 2002, pages 458–465. See proceedings [307]. Online available at http://citeseer.ist.psu.edu/595343.html and http://www.lania.mx/~ccoello/EMOO/watanabe02.pdf.gz [accessed 2007-07-29].

[370] 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 3rd International Conference on Genetic Algorithms ICGA*, 1989, pages 191–197. See proceedings [371].

[371] J. David Schaffer, editor. *Proceedings of the 3rd International Conference on Genetic Algorithms*, George Mason University, Fairfax, Virginia, USA, June 1989, San Francisco, CA. Morgan Kaufmann, ISBN: 1-55860-066-3.

[372] Carlos M. Fonseca and Peter J. Fleming. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In *Proceedings of the 5th International Conferenceo on Genetic Algorithms*, 1993, pages 416–423. See proceedings [444]. Online available at http://citeseer.ist.psu.edu/fonseca93genetic.html and http://www.lania.mx/~ccoello/EMOO/fonseca93.ps.gz [accessed 2007-08-29].

[373] Peter A. N. Bosman and Edwin D. de Jong. Exploiting gradient information in numerical multi-objective evolutionary optimization. In *GECCO '05: Proceedings of the 2005 conference on Genetic and evolutionary computation*, 2005, pages 755–762. See proceedings [299]. Online available at http://doi.acm.org/10.1145/1068009.1068138 [accessed 2007-08-22].

[374] Jeffrey Horn and Nicholas Nafpliotis. Multiobjective optimization using the niched pareto genetic algorithm. Technical Report IlliGAl Report 93005, Illinois Genetic Algorithms Laboratory (IlliGAL), Department of Computer Science, Department of General Engineering, University of Illinois at Urbana-Champaign, Urbana, Illinois, USA, June 1993. Online available at http://citeseer.ist.psu.edu/horn93multiobjective.html and http://www.lania.mx/~ccoello/EMOO/93005.ps.gz [accessed 2007-07-28].

[375] Jeffrey Horn. *The Nature of Niching: Genetic Algorithms and the Evolution of Optimal, Cooperative Populations*. PhD thesis, Illinois Genetic Algorithms Laboratory (IlliGAL), Department of Computer Science, Department of General Engineering, University of Illinois at Urbana-Champaign, Champaign, IL, USA, 1997. Advisor: David E. Goldberg. Online available at http://www.lania.mx/~ccoello/EMOO/hornthesis.ps.gz and http://citeseer.ist.psu.edu/horn97nature.html [accessed 2007-07-28].

[376] Mark Erickson, Alex Mayer, and Jeffrey Horn. The niched pareto genetic algorithm 2 applied to the design of groundwater remediation systems. In *Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization*, 2001, pages 681–695. See proceedings [268].

[377] Joshua D. Knowles and David W. Corne. The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation. In *Proceedings of the Congress on Evolutionary Computation*, 1999, volume 1, pages 98–105. See proceedings [249]. Online available at http://citeseer.ist.psu.edu/knowles99pareto.html [accessed 2007-08-27]. Some additional resources can be found at http://dbkgroup.org/knowles/multi/ [accessed 2007-08-27].

[378] António Gaspar Lopes da Cunha and José António Colaço Gomes Covas. RPSGAe - reduced pareto set genetic algorithm: a multiobjectiv algorithm with elistim: application to polymer extrusion. In Xavier Gandibleux, Marc Sevaux, Kenneth Sörensen, and Vincent T'kindt, editors, *MOMH Workshop on Multiple Objective Metaheuristics*, Carré des Sciences, Paris, France, November 4-5, 2002. Poster on the joint PM2O-EU/ME meeting. Online available at http://www2.lifl.fr/PM2O/Reunions/04112002/gaspar.pdf [accessed 2007-09-21], slides available at http://webhost.ua.ac.be/eume/workshops/momh/momh_gaspar_cunha.pdf [accessed 2007-09-21].

[379] António Gaspar Lopes da Cunha, Pedro Oliveira, and José António Colaço Gomes Covas. Use of genetic algorithms in multicriteria optimization to solve industrial problems. In Thomas Bäck, editor, *ICGA, Proceedings of the 7th International Conference on Genetic Algorithms*, 1997, pages 682–688. See proceedings [442].

[380] F. de Toro Negro, E. Ros J. Ortega, B. Paechter S. Mota, and J. M. Martín. Psfga: parallel processing and evolutionary computation for

multiobjective optimisation. *Parallel Computing*, 30(5-6):721–739, 2004. Online available at `http://hera.ugr.es/doi/15057690.pdf` [accessed 2007-07-29].

[381] 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 [118]. 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].

[382] Nils Aaall Barricelli. Esempi numerici di processi di evoluzione. *Methodos*, pages 45–68, 1954.

[383] Nils Aaall Barricelli. Symbiogenetic evolution processes realized by artificial methods. *Methodos*, 9:35–36, 1957.

[384] Nils Aaall Barricelli. Numerical testing of evolution theories. part i. theroetical introduction and basic tests. *Acta Biotheoretica*, 16(1/2):69–98, March 1962. Received: 27 November 1961, see also [385]. Online available at `http://www.springerlink.com/content/y502315688024453/fulltext.pdf` [accessed 2007-10-31].

[385] 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. Received: 27 November 1961, see also [384]. Online available at `http://www.springerlink.com/content/h85817217u25w6q7/fulltext.pdf` [accessed 2007-10-31].

[386] Hans J. Bremermann. Optimization through evolution and recombination. *Self-Organizing systems*, pages 93–10, 1962. Online available at `http://holtz.org/Library/Natural%20Science/Physics/` [accessed 2007-10-31].

[387] Woodrow "Woody" Wilson Bledsoe. Lethally dependent genes using instant selection. Technical Report PRI 1, Panoramic Research Inc., Palo Alto, California, USA, 1961.

[388] 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.

[389] Woodrow "Woody" Wilson Bledsoe. An analysis of genetic populations. Technical report, Panoramic Research Inc., Palo Alto, California, USA, 1962.

[390] Woodrow "Woody" Wilson Bledsoe. The evolutionary method in hill climbing: Convergence rates. Technical report, Panoramic Research Inc., Palo Alto, California, USA, 1962.

[391] Woodrow "Woody" Wilson Bledsoe and I. Browning. Pattern recognition and reading by machine. In *Proceedings of the Eastern Joint Computer Conference (EJCC)*, 1959, pages 225–232.

[392] 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].

[393] 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].

[394] 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].

[395] John Henry Holland. Outline for a logical theory of adaptive systems. *Journal of the ACM*, 9(3):297–314, 1962. Online available at http://portal.acm.org/citation.cfm?id=321128 [accessed 2007-07-28].

[396] Jack L. Crosby. *Computer Simulation in Genetics*. John Wiley and Sons Ltd, ISBN: 0-4711-8880-8, January 1973.

[397] D. Quagliarella, J. Periaux, C. Poloni, and G. Winter. *Genetic Algorithms and Evolution Strategy in Engineering and Computer Science: Recent Advances and Industrial Applications*. John Wiley & Sons Ltd, ISBN: 978-0471977100, January 1998.

[398] 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].

[399] 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*, NASA Ames Research Center, February 2000. Online available at http://ic.arc.nasa.gov/people/jlohn/bio.html and http://citeseer.ist.psu.edu/lohn00parallel.html [accessed 2007-08-07].

[400] 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 Con-*

ference, Big Sky, MT, March 2000. Online available at `http://ic.arc.nasa.gov/people/jlohn/bio.html` and `http://citeseer.ist.psu.edu/336276.html` [accessed 2007-08-07].

[401] Carlos Artemio Coello Coello. An updated survey of ga-based multiobjective optimization techniques. *ACM Computing Surveys*, 32(2):109–143, 2000. Online available at `http://citeseer.ist.psu.edu/coello98updated.html` and `http://portal.acm.org/citation.cfm?id=358923.358929` [accessed 2007-07-28].

[402] Hideyuki Takagi. Active user intervention in an ec search. In *Proceedings of International Conference on Information Sciences (JCIS2000)*, 2000, pages 995–998. See proceedings [293]. Online available at `http://www.kyushu-id.ac.jp/~takagi/TAKAGI/IECpaper/JCIS2K_2.pdf` [accessed 2007-08-29].

[403] Hideyuki Takagi. Interactive evolutionary computation: Fusion of the capacities of ec optimization and human evaluation. *Proceedings of the IEEE*, 89:1275–1296, 2001. Online available at `http://www.design.kyushu-u.ac.jp/~takagi/TAKAGI/IECsurvey.html` [accessed 2007-08-29].

[404] Alex Kosorukoff. Human-based genetic algorithm. In *Proceedings of the 2001 IEEE International Conference on Systems, Man, and Cybernetics*, Tucson, AZ, USA, January 2001, volume 5, pages 3464–3469, 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].

[405] Michelle Okaley Hammond and Terence Claus Fogarty. Cooperative oulipian generative literature using human based evolutionary computing. In *Late breaking paper at Genetic and Evolutionary Computation Conference (GECCO'2005)*, 2005. See proceedings [301]. 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].

[406] David Levine. Application of a hybrid genetic algorithm to airline crew scheduling. *Computers & Operations Research*, 23:547–558, 1996. Online available at `http://www.citeulike.org/user/ilapla/article/1443054` and `http://citeseer.ist.psu.edu/178394.html` [accessed 2007-07-29].

[407] 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*, 1989, pages 160–169. See proceedings [371].

[408] 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*, Cambridge/-Boston, MA, USA, August 1997, volume 471, pages 81–102. Springer, Berlin, ISBN: 3-540-65775-4, ISSN: 0075-8442. Online available

at http://www.citeulike.org/user/ilapla/article/1443013 and http://citeseer.ist.psu.edu/kwan97driver.html [accessed 2007-07-29].

[409] 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].

[410] 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.

[411] 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*, 2005, pages 189–193. See proceedings [318].

[412] 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*, pages 267–277, 1998. Online available at http://imaman.oxfordjournals.org/cgi/reprint/9/3/267.pdf [accessed 2007-07-28].

[413] D. M. Deaven and K.M. Ho. Molecular geometry optimization with a genetic algorithm. *Physical Review Letters*, 75:288–291, July 1995. 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].

[414] A. Cagnoni, A. Dobrzeniecki, R. Poli, and J. 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].

[415] 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].

[416] 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*, Las Vegas, Nevada, USA, June 2003, pages 406–409. CSREA Press, ISBN: 1-932415-04-1.

[417] Staal A. Vinterbo and Lucila Ohno-Machado. A genetic algorithm approach to multi-disorder diagnosis. *Artificial Intelligence in Medicine*, 18:117–132, 2000. Online available at `http://dx.doi.org/10.1016/S0933-3657(99)00036-6` [accessed 2007-09-05].

[418] 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*, 2001, pages 1–6. International Society for Computers and Their Applications (ISCA). Online available at `http://citeseer.ist.psu.edu/min01genetic.html` [accessed 2007-07-29].

[419] Ali Kamrani, Wang Rong, and Ricardo Gonzalez. A genetic algorithm methodology for data mining and intelligent knowledge acquisition. *Computers & Industrial Engineering*, 40:361–377, September 2001.

[420] 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*, Las Vegas, Nevada, USA, June 2006, pages 389–395. CSREA Press, ISBN: 1-60132-004-3. Online available at `http://ww1.ucmss.com/books/LFS/CSREA2006/DMI5509.pdf` [accessed 2007-07-29].

[421] 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].

[422] Jinxiang Chai and Songde Ma. Robust epipolar geometry estimation using genetic algorithm. *Pattern Recognition Letters*, 19(9):829–838, July 1998. See also [423].

[423] 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*, Hong Kong, China, January 1998, volume 1351 of *Lecture Notes in Computer Science (LNCS)*, pages 272–279. Springer, ISBN: 3-540-63930-6. See also [422].

[424] Adel Jedidi, Alexandre Caminada, and Gerd Finke. 2-objective optimization of cells overlap and geometry with evolutionary algorithms. In *Applications of Evolutionary Computing, EvoWorkshops 2004*, 2004, pages 130–139. See proceedings [286].

[425] 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*, Halifax, Canada, August 2003, pages 115–119.

[426] Sourav Kundu, Kazuto Seto, and Shigeru Sugino. Genetic algorithm based design of passive elements for vibration control. In *Proceed-*

ings of 4th MOVIC Conference (Conference On Motion and Vibration Control), August 1998, volume 3, pages 1183–1188. Online available at `http://citeseer.ist.psu.edu/349539.html` and `http://en.scientificcommons.org/362076` [accessed 2007-08-13].

[427] 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)*, Christchurch, New Zealand, January 29-31, 2002, pages 333–337, Los Alamitos, CA, USA. IEEE Computer Society, ISBN: 0-7695-1453-7.

[428] 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].

[429] Khaled El-Fakihy, Hirozumi Yamaguchiz, and Gregor v. 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*, Boston, USA, November 3-6, 1999. 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].

[430] Lidia A. R. Yamamoto and Christian Tschudin. Genetic evolution of protocol implementations and configurations. In *IFIP/IEEE International workshop on Self-Managed Systems and Services (SelfMan 2005)*, Nice, France, 2005. Online available at `http://cn.cs.unibas.ch/pub/doc/2005-selfman.pdf` [accessed 2007-09-17].

[431] 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*, Ciudad Universitaria ("University City"), Universidad Nacional Autonoma de Mexico, Mexico City, Mexico, January 8-11, 2007, volume 4436/2007 of *Lecture Notes in Computer Science (LNCS)*, Berlin Heidelberg. Springer, ISBN: 978-3-540-73479-6, ISSN: 0302-9743 (Print) 1611-3349 (Online). see `http://www.sigevo.org/foga-2007/index.html` [accessed 2007-09-01].

[432] 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*, Aizu-Wakamatsu City, Japan, January 5-9, 2005, volume 3469/2005 of *Lecture Notes in Computer Science (LNCS)*, Berlin Heidelberg. Springer Verlag, ISBN: 978-3-540-27237-3, ISSN: 0302-9743 (Print) 1611-3349 (Online). Published August 22, 2005. see `http://www.cs.umt.edu/foga05/` [accessed 2007-09-01].

[433] Kenneth Alan De Jong, Riccardo Poli, and Jonathan E. Rowe, editors. *Foundations of Genetic Algorithms 7*, Torremolinos, Spain, September 4-6, 2002, San Mateo, CA, USA. Morgan Kaufmann, ISBN: 0-12-208155-2. Published 2003.

[434] William M. Spears and Worthy N. Martin, editors. *Proceedings of the Sixth Workshop on Foundations of Genetic Algorithms*, Charlottesville, VA, USA, July 21-23, 2000, San Mateo, CA, USA. Morgan Kaufmann, ISBN: 1-55860-734-X. see `http://www.cs.uwyo.edu/~wspears/foga00/index.html` [accessed 2007-09-01].

[435] Wolfgang Banzhaf and Colin R. Reeves, editors. *Proceedings of the Fifth Workshop on Foundations of Genetic Algorithms*, Madison, WI, USA, July 22-25, 1998, San Mateo, CA, USA. Morgan Kaufmann, ISBN: 1-55860-559-2. Published April 2, 1991.

[436] Richard K. Belew and Michael D. Vose, editors. *Proceedings of the 4th Workshop on Foundations of Genetic Algorithms*, University of San Diego, San Diego, CA, USA, August 5, 1996, San Francisco, California, USA. Morgan Kaufmann, ISBN: 1-55860-460-X. Published March 1, 1997.

[437] L. Darrell Whitley and Michael D. Vose, editors. *Proceedings of the Third Workshop on Foundations of Genetic Algorithms*, Estes Park, Colorado, USA, July 31-August 2, 1994, San Francisco, CA, USA. Morgan Kaufmann, ISBN: 1-55860-356-5. Published June 1, 1995.

[438] L. Darrell Whitley, editor. *Proceedings of the Second Workshop on Foundations of Genetic Algorithms*, Vail, Colorado, USA, July 26-29, 1992, San Mateo, CA, USA. Morgan Kaufmann, ISBN: 1-55860-263-1. Published February 1, 1993.

[439] Gregory J. E. Rawlins, editor. *Proceedings of the First Workshop on Foundations of Genetic Algorithms*, Indiana University, Bloomington Campus, Indiana, USA, July 15-18, 1990, San Mateo, CA, USA. Morgan Kaufmann, ISBN: 1-55860-170-8. Published July 1, 1991.

[440] *Proceedings of the 2nd IEE GALESIA Conference*, Strathclyde, Glasgow, UK, September 2-4, 1997, ISBN: 0-85296-693-8, ISSN: 0537-9989.

[441] A. M. S. Zalzala, editor. *First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA)*, Scheffield, UK, September 12-14, 1995, volume 414. IEE Conference Publication, Institution of Engineering and Technology, ISBN: 978-0852966501.

[442] Thomas Bäck, editor. *Proceedings of The Seventh International Conference on Genetic Algorithms ICGA'97*, Michigan State University, East Lansing, Michigan, USA, July 19-23, 1997, San Francisco, CA, USA. Morgan Kaufmann Publishers, ISBN: 1-55860-487-1. see `http://garage.cse.msu.edu/icga97/` [accessed 2007-09-01].

[443] Larry J. Eshelman, editor. *Proceedings of the Sixth International Conference on Genetic Algorithms*, Pittsburgh, PA, USA, July 15-19, 1995, San Francisco, CA. Morgan Kaufmann, ISBN: 1-55860-370-0.

[444] Stephanie Forrest, editor. *Proceedings of the 5th International Conference on Genetic Algorithms*, Urbana-Champaign, IL, USA, June 1993, San Francisco, CA. Morgan Kaufmann, ISBN: 1-55860-299-2.

[445] Richard K. Belew and Lashon B. Booker, editors. *Proceedings of the 4th International Conference on Genetic Algorithms*, San Diego, CA, USA, July 1991, San Francisco, CA. Morgan Kaufmann, ISBN: 1-55860-208-9.

[446] John J. Grefenstette, editor. *Proceedings of the 2nd International Conference on Genetic Algorithms*, Cambridge, MA, USA, July 1987, Mahwah, NJ, USA. Lawrence Erlbaum Associates, ISBN: 0-8058-0158-8.

[447] John J. Grefenstette, editor. *Proceedings of the 1st International Conference on Genetic Algorithms and their Applications*, Carnegie-Mellon University, Pittsburgh, PA, USA, July 24-26, 1985, Mahwah, NJ, USA. Lawrence Erlbaum Associates, Inc., ISBN: 0-8058-0426-9.

[448] Mitsuo Gen and Runwei Chen. *Genetic Algorithms (Engineering Design and Automation)*. Wiley Series in Engineering Design and Automation. John Wiley and Sons Ltd, ISBN: 978-0-471-31531-5, February 2001.

[449] Tomasz Dominik Gwiazda. *Crossover for single-objective numerical optimization problems*, volume 1 of *Genetic Algorithms Reference*. Lomianki, e-book: Tomasz Dominik Gwiazda, ISBN: 83-923958-3-2, May 2006.

[450] Hans Winkler. *Verbreitung und Ursache der Parthenogenesis im Pflanzen- und Tierreiche*. Verlag Gustav Fischer, Jena, 1920.

[451] 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].

[452] 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, ISBN: 978-0805346350, 1987.

[453] Sanza Kazadi. Conjugate schema and basis representation of crossover and mutation. *Evolutionary Computation*, 6(2):129–160, September 1998. Online available at `http://www.jisan.org/research_collaborators/former_research/conjugate_schemata/papers/CONJUGATE98.PDF` [accessed 2007-08-29].

[454] 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*, Ann Arbor, Michigan, USA, June 1988, pages 153–161, San Mateo, California. Morgan Kaufmann, ISBN: 0-934613-64-8.

[455] Nicol N. Schraudolph and Richard K. Belew. Dynamic parameter encoding for genetic algorithms. *Machine Learning*, 9:9–21,

June 1992. Online available at http://citeseer.ist.psu.edu/schraudolph92dynamic.html and http://www.springerlink.com/content/u684071468r0x423/fulltext.pdf [accessed 2007-08-29].

[456] Wolfgang Banzhaf. Genotype-phenotype-mapping and neutral variation – A case study in genetic programming. In *Parallel Problem Solving from Nature III*, 1994, pages 322–332. See proceedings [329]. Online available at http://citeseer.ist.psu.edu/banzhaf94genotypephenotypemapping.html [accessed 2007-09-09].

[457] 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*, 1998, pages 280–291. See proceedings [327]. Online available at http://www.cs.ucl.ac.uk/staff/p.bentley/YUBEC2.pdf [accessed 2007-08-17].

[458] 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*, 2006, pages 1721–1728. See proceedings [298]. Online available at http://portal.acm.org/citation.cfm?id=1144278 [accessed 2007-08-17].

[459] Kenneth O. Stanley and Risto Miikkulainen. A taxonomy for artificial embryogeny. *Artificial Life*, 9(2):93–130, 2003. Online available at http://nn.cs.utexas.edu/downloads/papers/stanley.alife03.pdf [accessed 2007-08-17].

[460] 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].

[461] Sanjeev Kumar and Peter J. Bentley. Computational embryology: past, present and future. In Ashish Ghosh and Shigeyoshi Tsutsui, editors, *Theory and Application of Evolutionary Computation: Recent Trends*, ISBN: 9783540433309, pages 461–477. Springer-Verlag New York, Inc., New York, NY, USA, 2003. Online available at http://www.cs.ucl.ac.uk/staff/P.Bentley/KUBECH1.pdf [accessed 2007-08-17].

[462] Chris P. Bowers. Simulating evolution with a computational model of embryogeny: Obtaining robustness from evolved individuals. In Mathieu S. Capcarrère, Alex Alves Freitas, Peter J. Bentley, Colin G. Johnson, and Jon Timmis, editors, *Advances in Artificial Life, Proceedings of 8th European Conference*, Canterbury, UK, September 2005, volume 3630 of *Lecture Notes in Computer Science (LNCS)*, pages 149–158. Springer, ISBN: 354-0288-481. Online available at http://www.cs.bham.ac.uk/~cpb/publications/ecal05_bowers.pdf [accessed 2007-08-17].

[463] 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].

[464] John J. Grefenstette. Deception considered harmful. In *Proceedings of the Second Workshop on Foundations of Genetic Algorithms*, 1992, pages 75–91. See proceedings [438]. Online available at http://citeseer.ist.psu.edu/grefenstette92deception.html [accessed http://www.citeulike.org/user/pduval/article/1433820]2007-08-12.

[465] Melanie Mitchell, Stephanie Forrest, and John Henry Holland. The royal road for genetic algorithms: Fitness landscapes and GA performance. In Francisco J. Varela and Paul Bourgine, editors, *Towards a Practice of Autonomous Systems: Proceedings of the First European Conference on Artificial Life, 1991*, 1992, pages 245–254, Paris. A Bradford book, The MIT Press, ISBN: 0-262-72019-1. Online available at http://citeseer.ist.psu.edu/mitchell91royal.html and http://web.cecs.pdx.edu/~mm/ecal92.pdf [accessed 2007-10-15].

[466] 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].

[467] 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*, 1994, volume 1, pages 379–384. See proceedings [254]. Also: Handbook of Evolutionary Computation, 1997, Institute of Physics Publishing and Oxford University Press, Bristol, Thomas Bäck and David B. Fogel and Zbigniew Michalewicz. New YorkOnline available at http://citeseer.ist.psu.edu/palmer94representing.html and http://www.cs.cinvestav.mx/~constraint/ [accessed 2007-08-12].

[468] Charles Campbell Palmer. *An approach to a problem in network design using genetic algorithms*. PhD thesis, 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].

[469] Simon Ronald. Robust encodings in genetic algorithms: A survey of encoding issues. In *IEEE Forth International Conference on Evolu-*

tionary Computation (IEEE/ICEC'97), 1997, pages 43–48. See proceedings [251].

[470] Ingo Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog Verlag, Stuttgart, ISBN: 978-3772803741, 1973. his dissertation from 1970.

[471] Justinian P. Rosca and Dana H. Ballard. Causality in genetic programming. In *Proceedings of the 6th International Conference on Genetic Algorithms (ICGA95)*, 1995, pages 256–263. See proceedings [443]. Online available at `http://citeseer.ist.psu.edu/rosca95causality.html` [accessed 2007-08-12].

[472] 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].

[473] Reinhard Lohmann. Structure evolution and incomplete induction. *Biological Cybernetics*, 69(4):319–326, August 1993. Online available at `http://www.springerlink.com/content/q2q668316m771073/fulltext.pdf` [accessed 2007-08-12].

[474] Christian Igel. Causality of hierarchical variable length representations. In *Proceedings of the 1998 IEEE World Congress on Computational Intelligence*, 1998, pages 324–329, Anchorage, Alaska, USA. IEEE Press. 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].

[475] 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)*, 1997. See proceedings [442]. Online available at `http://citeseer.ist.psu.edu/sendhoff97condition.html` and `http://arxiv.org/abs/adap-org/9711001` [accessed 2007-08-12].

[476] Peter Stagge and Christian Igel. Structure optimization and isomorphisms. In Leila Kallel, Bart Naudts, and Alex Rogers, editors, *Theoretical Aspects of Evolutionary Computing*, ISBN: 3-540-67396-2, pages 409–422. Springer, Berlin/London, UK, 2000. Online available at `http://citeseer.ist.psu.edu/520775.html` [accessed 2007-08-13].

[477] William Bateson. *Mendel's Principles of Heredity*. Cambridge University Press, Cambridge, ISBN: 9781428648197, 1909. 1930: fourth impression of the 1909 edition.

[478] 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].

[479] Lee Altenberg. Nk fitness landscapes. In *Handbook of Evolutionary Computation*, chapter B2.7.2. Oxford University Press, November 27, 1996. See collection [180]. Online available at `http://citeseer.ist.psu.edu/704814.html` and `http://dynamics.org/Altenberg/FILES/LeeNKFL.pdf` [accessed 2007-11-27].

[480] Yuval Davidor. Epistasis variance: A viewpoint on GA-hardness. In *Proceedings of the First Workshop on Foundations of Genetic Algorithms*, 1990, pages 23–35. See proceedings [439].

[481] Bart Naudts and Alain Verschoren. Epistasis on finite and infinite spaces. In *Proceedings of the 8th International Conference on Systems Research, Informatics and Cybernetics*, 1996, pages 19–23. Online available at `http://en.scientificcommons.org/155291` and `http://citeseer.ist.psu.edu/142750.html` [accessed 2007-08-13].

[482] 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].

[483] 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].

[484] Marc Toussaint and Christian Igel. Neutrality: A necessity for self-adaptation. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2002)*, 2002, pages 1354–1359. See proceedings [246]. Online available at `http://citeseer.ist.psu.edu/toussaint02neutrality.html` [accessed 2007-07-28].

[485] 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*, 1999, pages 1–11. See proceedings [321]. Online available at `http://citeseer.ist.psu.edu/shipman99genetic.html` [accessed 2007-07-29].

[486] 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*, 2000, pages 493–500. See proceedings [248]. Online available at `http://citeseer.ist.psu.edu/409243.html` [accessed 2007-07-29].

[487] 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].

[488] Richard M. Friedberg. A learning machine: Part i. *IBM Journal of Research and Development*, 2:2–13, November 1958. On-

line available at http://www.research.ibm.com/journal/rd/021/ibmrd0201B.pdf [accessed 2007-09-06]. See also [489].

[489] 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 [488].

[490] 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.

[491] 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).

[492] Richard Forsyth and Roy Rada. *Machine Learning applications in Expert Systems and Information Retrieval*. Ellis Horwood series in Artificial Intelligence. Ellis Horwood, Chichester, UK, ISBN: 0-7458-0045-9, 0-470-20309-9, 1986. Contains chapters on BEAGLE, see [491]. Also published by John Wiley & Sons Australia (May 28, 1986) and Halsted Press, New York, NY, USA.

[493] Richard Forsyth. The evolution of intelligence. In Richard Forsyth, editor, *Machine Learning, Priciples and Techniques*, ISBN: 0-412-30570-4, 0-412-30580-1, chapter 4, pages 65–82. Chapman and Hall, 1989. Refers also to PC/BEAGLE, see [491].

[494] 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*, 1985, pages 183–187. See proceedings [447]. Online available at http://www.sover.net/~nichael/nlc-publications/icga85/index.html [accessed 2007-09-06].

[495] 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].

[496] 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].

[497] 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.

[498] 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.

[499] 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)*, Freiburg, Germany, September 3-5, 2001, volume 2168 of *Lecture Notes in Computer Science (LNCS), subseries Lecture Nots in Artificial Intelligence (LNAI)*, pages 314–325. Springer Verlag Berlin/Heidelberg, ISBN: 978-3-540-42534-2, ISSN: 0302-9743 (Print) 1611-3349 (Online). See also [500]. Online available at `http://www.cs.kent.ac.uk/people/staff/aaf/pub_papers.dir/PKDD-2001.ps` [accessed 2007-09-09].

[500] 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. See proceedings [310] and [499]. Online available at `http://citeseer.ist.psu.edu/521000.html` [accessed 2007-09-09].

[501] 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*, 1990, pages 124–128. See proceedings [331]. Online available at `http://citeseer.ist.psu.edu/61578.html` [accessed 2007-09-09].

[502] Tatiana Kalganova and Julian F. Miller. Evolving more efficient digital circuits by allowing circuit layout evolution and multi-objective fitness. In *Evolvable Hardware, Proceedings of 1st NASA/DoD Workshop on Evolvable Hardware (EH '99)*, Pasadena, CA, USA, July 1999, page 54. IEEE Computer Society, ISBN: 0-7695-0256-3. Online available at `http://citeseer.ist.psu.edu/kalganova99evolving.html` [accessed 2007-09-09].

[503] Joel Jones. Abstract syntax tree implementation idioms. In *Proceedings of The 10th Conference on Pattern Languages of Programs PLoP'2003*, Monticello, Illinois, USA, September 8-12, 2003. A workshop presentation online available at `http://hillside.net/plop/plop2003/Papers/Jones-ImplementingASTs.pdf` [accessed 2007-07-03].

[504] 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].

[505] 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*, April 2005, pages 3–18. China University of Geosciences Press. Online available at `http://sc.snu.ac.kr/PAPERS/isica05.pdf` [accessed 2007-08-15].

[506] Douglas A. Augusto and Helio J. C. Barbosa. Symbolic regression via genetic programming. In *Proceedings of Sixth Brazilian Symposium on Neural Networks (SBRN'00)*, 2000, pages 173–178, Los Alamitos, CA, USA. IEEE Computer Society, ISBN: 0-7695-0856-1, ISSN: 1522-4899.

[507] 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)*, November 3-5, 2003, pages 161–167, Los Alamitos, CA, USA. IEEE Computer Society, ISSN: 1082-3409.

[508] Günther R. Raidl. A hybrid gp approach for numerically robust symbolic regression. In *Genetic Programming 1998: Proceedings of the Third Annual Conference*, 22–25 1998, pages 323–328. See proceedings [579]. 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].

[509] 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*, Huntsville, Alabama, 2004, pages 404–405, New York, NY, USA. ACM Press, ISBN: 1-58113-870-9. Online available at `http://doi.acm.org/10.1145/986537.986635` [accessed 2007-09-09].

[510] 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*, Melbourne, Florida, 2003, pages 1063–1067, New York, NY, USA. ACM Press, ISBN: 1-58113-624-2. 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].

[511] 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. Online available at `http://doi.acm.org/10.1145/1064165.1064174` [accessed 2007-09-09].

[512] Emin Erkan Korkmaz and Göktürk Üçoluk. Genetic programming for grammar induction. In *Genetic and Evolutionary Computation Conference 2001 – Late Breaking Papers*, 2001, pages 245–251. See proceedings [311]. Online available at `http://citeseer.ist.psu.edu/451812.html` and `http://de.scientificcommons.org/464341` [accessed 2007-10-14].

[513] 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*, 1997, pages 96–101. See proceedings [581]. Online available at `http://citeseer.ist.psu.edu/43454.html` [accessed 2007-09-09].

[514] 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. See proceedings [306] and also [515].

[515] 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)*, 2003, pages 384–393. See proceedings [572] and also [514]. Online available at `http://www.cs.kent.ac.uk/people/staff/aaf/pub_papers.dir/EuroGP-2003-Fernando.pdf` [accessed 2007-09-09].

[516] 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].

[517] Man Leung Wong and Kwong Sak Leung. *Data Mining Using Grammar Based Genetic Programming and Applications*, volume 3 of *Genetic Programming*. Springer, ISBN: 978-0792377467, January 2000.

[518] 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)*, Pasadena, CA, USA, July 1999, pages 46–53. IEEE Computer Society, ISBN: 0-7695-0256-3. Online available at `http://citeseer.ist.psu.edu/521808.html` [accessed 2007-09-09].

[519] 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: [1317].

[520] 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*, ISBN: 1-55860-605-X, chapter 16, pages 365–385. Morgan Kaufmann, 1999. See collection [333]. Online available at `http://www.genetic-programming.com/jkpdf/edc1999.pdf` [accessed 2007-10-03].

[521] Tatiana Kalganova. An extrinsic function-level evolvable hardware approach. In *Genetic Programming, Proceedings of EuroGP'2000*, 2000, pages 60–75. See proceedings [575]. Online available at `http://citeseer.ist.psu.edu/kalganova00extrinsic.html` [accessed 2007-09-09].

[522] 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*, 1996, pages 132–149. See proceedings [583]. Online available at `http://citeseer.ist.psu.edu/119355.html` [accessed 2007-10-03].

[523] 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*, 1996, pages 94–100. See proceedings [584]. Online available at `http://www.genetic-programming.com/jkpdf/gp1996lbpamplifier.pdf` and `http://citeseer.ist.psu.edu/105348.html` [accessed 2007-10-03].

[524] 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.

[525] 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*, San Diego, CA USA, October 11-14, 1998, volume 3, pages 2184–2189. IEEE.

[526] 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. Online available at `http://dx.doi.org/10.1016/j.artmed.2004.02.007` [accessed 2007-09-09].

[527] Markus 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].

[528] 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), 2003, pages 11–59. See proceedings [572]. Online available at http://www.cs.kent.ac.uk/people/staff/aaf/pub_papers.dir/EuroGP-2003-Celia.pdf [accessed 2007-09-09].

[529] 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].

[530] 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].

[531] 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.

[532] 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*, 2001, pages 343–352. See proceedings [282]. Online available at http://citeseer.ist.psu.edu/485032.html [accessed 2007-09-09].

[533] Jason D. Lohn, Gregory S. Hornby, and Derek Linden. An evolved antenna for deployment on nasas space technology 5 mission. In *Genetic Programming Theory and Practice II*, 2004. See proceedings [588]. Online available at http://ic.arc.nasa.gov/people/hornby/papers/lohn_gptp04.ps.gz [accessed 2007-08-17].

[534] 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)*, 1996, pages 3–11. See proceedings [583] and also [535]. Online available at http://citeseer.ist.psu.edu/33008.html [accessed 2007-08-01].

[535] 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*, Nara, Japan, May 16–18, 1996, volume 1. MIT Press, Cambridge, MA, USA. See also [534]. Online available at http://www.genetic-programming.com/jkpdf/alife1996gkl.pdf and http://citeseer.ist.psu.edu/andre96evolution.html [accessed 2007-10-03].

[536] Hugo de Garis. Artificial embryology: The genetic programming of an artificial embryo. In Branko Souček, editor, *Dynamic, Genetic, and Chaotic Programming*, pages 373–393. John Wiley, New York, 1992.

Online available at http://citeseer.ist.psu.edu/99683.html [accessed 2007-08-01].

[537] 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*, Brussels, Belgium, 1993, pages 274–284. Online available at http://citeseer.ist.psu.edu/degaris93evolving.html [accessed 2007-08-01].

[538] 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*, 1998, pages 412–415. See proceedings [579]. Online available at http://citeseer.ist.psu.edu/kundu98genetic.html [accessed 2007-09-09].

[539] 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, first edition, ISBN: 978-1558605107, November 30, 1997.

[540] Sean Luke, Charles Hohn, Jonathan Farris, Gary Jackson, and James Hendler. Co-evolving 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, Nagoya, Japan. Online available at http://citeseer.ist.psu.edu/3898.html and http://www.cs.umd.edu/~seanl/papers/robocup.pdf [accessed 2007-09-09]. See also [541].

[541] Sean Luke, Charles Hohn, Jonathan Farris, Gary Jackson, and James Hendler. Co-evolving soccer softbot team coordination with genetic programming. In Hiroaki Kitano, editor, *RoboCup-97: Robot Soccer World Cup I*, 1998, number 1395 in Lecture Notes in Computer Science (LNCS), subseries Lecture Notes in Artificial Intelligence (LNAI). Springer Berlin/Heidelberg, ISBN: 978-3-540-64473-6, ISSN: 0302-9743 (Print) 1611-3349 (Online). Online available at hhttp://www.cs.umd.edu/~seanl/papers/robocupc.pdf [accessed 2007-09-09]. See also [540].

[542] 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 [540, 541].

[543] John R. Koza, Martin A. Keane, Matthew J. Streeter, William Mydlowec, Jessen Yu, and Guido Lanza. *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Genetic Programming. Springer, first edition, ISBN: 978-0387250670, May 2005.

[544] John R. Koza and James P. Rice. Automatic programming of robots using genetic programming. In *Proceedings of Tenth National Conference on Artificial Intelligence*, 1992, pages 194–201. See proceedings [1318]. Online available at http://

citeseer.ist.psu.edu/koza92automatic.html and http://www.genetic-programming.com/jkpdf/aaai1992.pdf [accessed 2007-09-07].

[545] David Andre and Astro Teller. Evolving team darwin united. In Minoru Asada and Hiroaki Kitano, editors, *RoboCup-98: Robot Soccer World Cup II*, Paris, France, July 1998, volume 1604 of *Lecture Notes in Artificial Intelligence, subseries of Lecture Notes in Computer Science (LNCS)*, pages 346–351. Springer Verlag, ISBN: 3-540-66320-7, ISSN: 0302-9743 (Print) 1611-3349 (Online). 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].

[546] 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*, Japan, 2000, pages 351–354.

[547] Hitoshi Iba. Evolving multiple agents by genetic programming. In *Advances in genetic programming 3*, pages 447–466. MIT Press, Cambridge, MA, USA, 1999. See collection [595]. Online available at http://www.cs.bham.ac.uk/~wbl/aigp3/ch19.pdf [accessed 2007-10-03].

[548] 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*, 2007, pages 167–173. See proceedings [316]. Online available at http://www.it-weise.de/documents/files/W2007DGPFb.pdf [accessed 2008-1-4] and http://www.springerlink.com/content/978-3-540-71589-4/?p_o=10 [accessed 2007-08-13].

[549] F. Comellas and G. Giménez. Genetic programming to design communication algorithms for parallel architectures. *Parallel Processing Letters*, 8(4):549–560, 1998. 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].

[550] 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 2008-1-4].

[551] Thomas Weise and Kurt Geihs. Genetic programming techniques for sensor networks. In *Proceedings of 5. GI/ITG KuVS Fachgespräch "Drahtlose Sensornetze"*, University of Stuttgart, Stuttgart, Germany, July 17-18, 2006, pages 21–25. 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].

[552] 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, 2006, pages 157–166. See proceedings [239]. Online available at http://www.it-weise.de/documents/files/W2006DGPFc.pdf [accessed 2008-1-4].

[553] Forrest H Bennett III. Emergence of a multi-agent architecture and new tactics for the ant colony foraging problem using genetic programming. In 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*, Cape Code, USA, September 9-13, 1996, pages 430–439. MIT Press, Cambridge, MA, USA, ISBN: 0-262-63178-4.

[554] 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*, 1996, pages 30–38. See proceedings [583].

[555] 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].

[556] Mohammad Adil Qureshi. Evolving agents. In *Genetic Programming 1996: Proceedings of the First Annual Conference*, 1996, pages 369–374. See proceedings [583] and also [557]. Online available at http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/AQ.gp96.ps.gz [accessed 2007-09-17].

[557] Mohammad Adil Qureshi. Evolving agents. Research Note RN/96/4, UCL, Gower Street, London, WC1E 6BT, UK, January 1996. Online available at ftp://ftp.cs.bham.ac.uk/pub/tech-reports/1997/CSRP-97-07.ps.gz [accessed 2007-09-17]. See also [557].

[558] 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 [1319]. 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].

[559] 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)*, 1995, pages 271–278. See proceedings [443]. 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].

[560] 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), 1995, pages 248–255, ISBN: 1-55860-370-0. See proceedings [443].

[561] 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*, Montreal, Quebec, Canada, August 20-25, 1995, volume 1, pages 741–747. Morgan Kaufmann, San Francisco, CA, USA, ISBN: 1-55860-363-8.

[562] 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*, 2002, pages 73–80. See proceedings [308]. Online available at `http://citeseer.ist.psu.edu/614297.html` and `http://hampshire.edu/lspector/pubs/ecomas2002-spector-toappear.pdf` [accessed 2007-12-25].

[563] David Andre. Learning and upgrading rules for an OCR system using genetic programming. In *Proceedings of the 1994 IEEE World Congress on Computational Intelligence*, June 27-29, 1994, Orlando, Florida, USA. IEEE Press. Uses GP both to recognise C in various fonts and to maintain manually produced extremely high level code when a new font is added. 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].

[564] David Andre. Learning and upgrading rules for an optical character recognition system using genetic programming. In *Handbook of Evolutionary Computation*, ISBN: 0-7503-0392-1. Oxford University Press, 1997. See collection [180].

[565] 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*, ISBN: 0-262-01158-1, chapter 8, pages 155–176. MIT Press, 1996. See collection [594]. Online available at `http://www.genetic-programming.com/jkpdf/aigp2aatmjk1996.pdf` and `http://citeseer.ist.psu.edu/75759.html` [accessed 2007-10-03].

[566] 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*, 1995, pages 89–97. See proceedings [1320]. Online available at `http://www.genetic-programming.com/jkpdf/ml1995motif.pdf` and `http://citeseer.ist.psu.edu/83080.html` [accessed 2007-10-03].

[567] Peter John Angeline and Jordan B. Pollack. Coevolving high-level representations. In Christopher G. Langton, editor, *Artificial Life III*, Santa Fe, New Mexico, USA, June 15-19, 1992, volume XVII of *SFI Studies in the Sciences of Complexity*, pages 55–71. Addison-Wesley. Published 1994. Online available at `http://demo.cs.brandeis.edu/papers/alife3.pdf` and `http://citeseer.ist.psu.edu/angeline94coevolving.html` [accessed 2007-10-05].

[568] 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*, Valencia, Spain, April 11-13, 2007, volume 4445/2007 of *Lecture Notes in Computer Science (LNCS)*. Springer Berlin/Heidelberg, ISBN: 978-3-540-71602-0, ISSN: 0302-9743 (Print) 1611-3349 (Online).

[569] Pierre Collet, Marco Tomassini, Marc Ebner, Steven Gustafson, and Anikó Ekárt, editors. *Proceedings of the 9th European Conference Genetic Programming, EuroGP 2006*, Budapest, Hungary, April 10-12, 2006, volume 3905/2006 of *Lecture Notes in Computer Science (LNCS)*. Springer Berlin/Heidelberg, ISBN: 3-540-33143-3, ISSN: 0302-9743 (Print) 1611-3349 (Online).

[570] Maarten Keijzer, Andrea Tettamanzi, Pierre Collet, Jano I. van Hemert, and Marco Tomassini, editors. *Proceedings of the 8th European Conference on Genetic Programming, EuroGP2005*, Lausanne, Switzerland, March 30-April 1, 2005, volume 3447/2005 of *Lecture Notes in Computer Science (LNCS)*. Springer Berlin/Heidelberg, ISBN: 3-540-25436-6, ISSN: 0302-9743 (Print) 1611-3349 (Online). see `http://evonet.lri.fr/eurogp2005/` [accessed 2007-09-01].

[571] Maarten Keijzer, Una-May O'Reilly, Simon M. Lucas, Ernesto Costa, and Terence Soule, editors. *Proccedings of the 7th European Conference on Genetic Programming, EuroGP2004*, Coimbra, Portugal, April 5-7, 2004, volume 3003/2004 of *Lecture Notes in Computer Science (LNCS)*. Springer Berlin/Heidelberg, ISBN: 3-540-21346-5, ISSN: 0302-9743 (Print) 1611-3349 (Online).

[572] 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*, Essex, UK, April 14-16, 2003, volume 2610/2003 of *Lecture Notes in Computer Science (LNCS)*. Springer Berlin/Heidelberg, ISBN: 3-540-00971-X, ISSN: 0302-9743 (Print) 1611-3349 (Online).

[573] James A. Foster, Evelyne Lutton, Julian F. Miller, Conor Ryan, and Andrea Tettamanzi, editors. *Proceedings of the 5th European Conference on Genetic Programming, EuroGP 2002*, Kinsale, Ireland, April 3-5, 2002, volume 2278/2002 of *Lecture Notes in Computer Science (LNCS)*. Springer Berlin/Heidelberg, ISBN: 3-540-43378-3, ISSN: 0302-9743 (Print) 1611-3349 (Online).

[574] Julian F. 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*, Lake Como, Italy, April 18-20, 2001, volume 2038/2001 of *Lecture Notes in Computer Science (LNCS)*. Springer Berlin/Heidelberg, ISBN: 3-540-41899-7, ISSN: 0302-9743 (Print) 1611-3349 (Online).

[575] Riccardo Poli, Wolfgang Banzhaf, William B. Langdon, Julian F. Miller, Peter Nordin, and Terence C. Fogarty, editors. *Proceedings of the European Conference on Genetic Programming*, Edinburgh, Scotland, UK, April 15-16, 2000, volume 1802/2000 of *Lecture Notes in Computer Science (LNCS)*. Springer Berlin/Heidelberg, ISBN: 3-540-67339-3, ISSN: 0302-9743 (Print) 1611-3349 (Online).

[576] Riccardo Poli, Peter Nordin, William B. Langdon, and Terence C. Fogarty, editors. *Proceedings of the Second European Workshop on Genetic Programming*, Göteborg, Sweden, May 26-27, 1999, volume 1598/1999 of *Lecture Notes in Computer Science (LNCS)*. Springer Berlin/Heidelberg, ISBN: 3-540-65899-8, ISSN: 0302-9743 (Print) 1611-3349 (Online).

[577] Wolfgang Banzhaf, Riccardo Poli, Marc Schoenauer, and Terence C. Fogarty, editors. *Proccedings of the First European Workshop on Genetic Programming, EuroGP'98*, Paris, France, April 14-15, 1998, volume 1391/1998 of *Lecture Notes in Computer Science (LNCS)*. Springer Berlin/Heidelberg, ISBN: 3-540-64360-5, ISSN: 0302-9743 (Print) 1611-3349 (Online). See alse [578].

[578] Riccardo Poli, William B. Langdon, Marc Schoenauer, Terry Fogarty, and Wolfgang Banzha, editors. *Late Breaking Papers at EuroGP'98: The First European Workshop on Genetic Programming*, Paris, France, April 14-15, 1998. Distributed at the workshop. See alse [577].

[579] 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)*, University of Wisconsin, Madison, Wisconsin, USA, July 22-25, 1998, Los Altos, CA, USA. Morgan Kaufmann, ISBN: 1-55860-548-7. see `http://www.genetic-programming.org/gp98cfp.html` [accessed 2007-09-01] and [580].

[580] John R. Koza, editor. *Late Breaking Papers at the Genetic Programming 1998 Conference*, University of Wisconsin, Madison, Wisconsin, USA, July 22-25, 1998. see also [579].

[581] 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*, Stanford University, CA, USA, July 13-16, 1997, San Francisco, CA, USA. Morgan Kaufmann. see also [582].

[582] *Late Breaking Papers at the 1997 Genetic Programming Conference*, Stanford University, CA, USA, July 13-16, 1997. Stanford Bookstore,

ISBN: 0-18-206995-8. see also [581].

[583] John R. Koza, David E. Goldberg, David B. Fogel, and Rick L. Riolo, editors. *Proceedings of the First Annual Conference Genetic Programming (GP-96)*, Stanford University, CA, USA, July 28–31, 1996. MIT Press.

[584] John R. Koza, editor. *Late Breaking Papers at the First Annual Conference Genetic Programming (GP-96)*, Stanford University, CA, USA, July 28–31, 1996. Stanford Bookstore, ISBN: 0-18-201031-7.

[585] *Genetic Programming Theory and Practice V, Proceedings of the Genetic Programming Theory Practice 2007 Workshop (GPTP-2007)*, The Center for the Study of Complex Systems (CSCS), University of Michigan, Ann Arbor, Michigan, USA, May 17-19, 2007, Genetic and Evolutionary Computation. Springer. See `http://www.cscs.umich.edu/gptp-workshops/gptp2007/` [accessed 2007-09-28].

[586] 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)*, The Center for the Study of Complex Systems (CSCS), University of Michigan, Ann Arbor, Michigan, USA, May 11-13, 2006, Genetic and Evolutionary Computation. Springer, ISBN: 978-0-387-33375-5. See `http://www.cscs.umich.edu/gptp-workshops/gptp2006/` [accessed 2007-09-28].

[587] 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)*, The Center for the Study of Complex Systems (CSCS), University of Michigan, Ann Arbor, Michigan, USA, May 12-14, 2005, volume 9 of *Genetic Programming Series*. Springer, ISBN: 978-0-387-28110-0. See `http://www.cscs.umich.edu/gptp-workshops/gptp2005/` [accessed 2007-09-28].

[588] Una-May O'Reilly, 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)*, The Center for the Study of Complex Systems (CSCS), University of Michigan, Ann Arbor, Michigan, USA, May 13-15, 2004, volume 8 of *Genetic Programming Series*. Springer, ISBN: 978-0-387-23253-9. See `http://www.cscs.umich.edu/gptp-workshops/gptp2004/` [accessed 2007-09-28].

[589] Rick Riolo and Bill Worzel, editors. *Genetic Programming Theory and Practice, Proceedings of the Genetic Programming Theory Practice 2003 Workshop (GPTP-2003)*, The Center for the Study of Complex Systems (CSCS), University of Michigan, Ann Arbor, UMichigan, SA, May 15-17, 2003, Genetic Programming Series. Kluwer Publishers, Boston, MA, ISBN: 1402075812. See `http://www.cscs.umich.edu/gptp-workshops/gptp2003/` [accessed 2007-09-28].

[590] John R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs: Automatic Discovery of Reusable Programs*. Com-

plex Adaptive Systems. The MIT Press, ISBN: 0262111896, July 4, 1994.

[591] 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, ISBN: 978-1558605435, May 1999.

[592] William B. Langdon. *Genetic Programming and Data Structures: Genetic Programming + Data Structures = Automatic Programming!* Genetic Programming. Springer, ISBN: 0792381351, April 30, 1998.

[593] Jr. Kenneth E. Kinnear, editor. *Advances in Genetic Programming*, volume 1. MIT Press, Cambridge, MA, USA, ISBN: 0-262-11188-8, April 1994.

[594] Peter J. Angeline and Kenneth E. Kinnear, Jr, editors. *Advances in Genetic Programming*, volume 2. MIT Press, Cambridge, MA, USA, ISBN: 0-262-01158-1, October 1996.

[595] Lee Spector, William B. Langdon, Una-May O'Reilly, and Peter J. Angeline, editors. *Advances in Genetic Programming*, volume 3. MIT Press, Cambridge, MA, USA, ISBN: 0-262-19423-6, July 1999.

[596] 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, ISBN: 978-3790809640, January 1997. First edition: 1995.

[597] Peter Nordin and Wolfgang Banzhaf. Complexity compression and evolution. In *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, 1995, pages 310–317. See proceedings [443]. Online available at `http://citeseer.ist.psu.edu/nordin95complexity.html` [accessed 2007-09-07].

[598] 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*, June 27-29, 1994, volume 1, pages 250–255, Orlando, Florida, USA. IEEE Press.

[599] 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*, 1996, pages 116–122. See proceedings [583]. Online available at `http://citeseer.ist.psu.edu/4569.html` [accessed 2007-09-09].

[600] 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*, 1996, pages 116–122. See proceedings [583]. 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].

[601] Robert E. Keller and Wolfgang Banzhaf. The evolution of genetic code in genetic programming. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 1999, volume 2, pages 1077–1082. See proceedings [314]. Online available at `http://citeseer.ist.psu.edu/244531.html` and `http://www.cs.mun.ca/~banzhaf/papers/t.pdf` [accessed 2007-09-09].

[602] Motoo Kimura. Evolutionary rate at the molecular level. *Nature*, 217:624–626, February 1968.

[603] Motoo Kimura. *The Neutral Theory of Molecular Evolution*. Cambridge University Press, ISBN: 978-0521317931, 1983. reprint edition, 1985, February.

[604] 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].

[605] Cândida Ferreira. Gene expression programming: a new adaptive algorithm for solving problems. *Complex Systems*, 13:87, 2001. Online available at `http://arxiv.org/ftp/cs/papers/0102/0102027.pdf` and `http://www.gene-expression-programming.com/GEPBiblio.asp` [accessed 2007-08-23].

[606] 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)*, Research Triangle Park, North Carolina, USA, March 2002, pages 614–617. JCIS / Association for Intelligent Machinery, Inc., ISBN: 0-9707890-1-7. Online available at `http://www.gene-expression-programming.com/GEPBiblio.asp` [accessed 2007-08-23].

[607] Cândida Ferreira. Analyzing the founder effect in simulated evolutionary processes using gene expression programming. In *Soft Computing Systems - Design, Management and Applications, HIS 2002*, 2002, pages 153–162. See proceedings [137]. Online available at `http://www.gene-expression-programming.com/GEPBiblio.asp` [accessed 2007-08-23].

[608] 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].

[609] 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].

[610] Cândida Ferreira. Discovery of the boolean functions to the best density-classification rules using gene expression programming. In James A. Foster, Evelyne Lutton, Julian Miller, Conor Ryan, and Andrea G. B. Tettamanzi, editors, *Proceedings of the 5th European Conference on Genetic Programming, EuroGP 2002*, 2002, pages 50–59, ISBN: 3-540-43378-3. See proceedings [573]. Online available at `http://www.gene-expression-programming.com/webpapers/ferreira-EuroGP02.pdf` [accessed 2007-09-09].

[611] 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].

[612] 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, 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].

[613] 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)*, June 2002, pages 1355–1361, Las Vegas, U.S.A.

[614] 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*, Beijing, China, August 11-13, 2002, volume 2419/2002 of *Lecture Notes in Computer Science (LNCS)*, pages 281–294. Springer-Verlag, ISBN: 3-540-44045-3, ISSN: 0302-9743 (Print) 1611-3349 (Online). Computer Department, Sichuan University China.

[615] 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*, 7:519–531, December 2003.

[616] 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].

[617] 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*, 2003, pages 1819–1825. See proceedings [245].

[618] Candida 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].

[619] 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, 2005.

[620] Liliana Teodorescu. High energy physics data analysis with gene expression programming. In *Nuclear Science Symposium Conference Record*, October 23-29, 2005, volume 1, pages 143–147. IEEE, ISSN: 1082-3654. INSPEC Accession Number:8976991.

[621] 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*, Izmir, Turkey, 2005, pages 280–285.

[622] Abdulkadir Çevik. A new formulation for web crippling strength of cold-formed steel sheeting using genetic programming. *Journal of Constructional Steel Research*, 63:867–883, July 2007.

[623] 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 [624].

[624] 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 [623] and is itself superseded by [625].

[625] 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).

[626] 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. See collection [594]. Online available at http://citeseer.ist.psu.edu/haynes96type.html [accessed 2007-10-04].

[627] Hendrik James Antonisse. A grammar-based genetic algorithm. In *Proceedings of Foundations of Genetic Algorithms FOGA-90*, 1990, pages 193–204. See proceedings [439].

[628] Pawel A. Stefanski. Genetic programming using abstract syntax trees, 1993. Notes from Genetic Programming Workshop at ICGA-93 [444]. 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.).

[629] 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].

[630] 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*, June 27-29, 1994, volume 2, pages 661–664. IEEE Press.

[631] 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*, 1998, pages 111–119. See proceedings [166]. Online available at http://citeseer.ist.psu.edu/360445.html [accessed 2007-11-12].

[632] 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*, 1994, pages 159–164. 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].

[633] Man Leung Wong and Kwong Sak Leung. Combining genetic programming and inductive logic programming using logic grammars. In *IEEE Conference on Evolutionary Computation*, 1995, volume 2, pages 733–736. See proceedings [253]. 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].

[634] 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*, 1995, pages 737–752. See proceedings [805], 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].

[635] 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. 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].

[636] Man Leung Wong and Kwong Sak Leung. Applying logic grammars to induce sub-functions in geneticprogramming. In *Proceedings of IEEE International Conference on Evolutionary Computation*, 1995, volume 2. See proceedings [253]. 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].

[637] 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].

[638] P. A. 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.

[639] P. A. Whigham. Grammatically-based genetic programming. In *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, 1995, pages 33–41. See proceedings [1320]. Online available at http://citeseer.ist.psu.edu/whigham95grammaticallybased.html [accessed 2007-08-15].

[640] P. A. Whigham. Inductive bias and genetic programming. In *First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, GALESIA*, 1995, pages 461–466. See proceedings [441]. Online available at http://citeseer.ist.psu.edu/343730.html [accessed 2008-08-15].

[641] P. A. Whigham. Search bias, language bias, and genetic programming. In *Genetic Programming 1996: Proceedings of the First Annual Conference*, 1996, pages 230–237. See proceedings [583]. 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].

[642] Walter Bohm and Andreas Geyer-Schulz. Exact uniform initialization for genetic programming. In *Foundations of Genetic Algorithms IV*, 1996, pages 379–407. See proceedings [436]. k-bounded context-free languages.

[643] 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].

[644] Norman R. Paterson and Mike Livesey. Distinguishing genotype and phenotype in genetic programming. In *Late Breaking Papers at the Genetic Programming 1996 Conference*, 1996, pages 141–150. See proceedings [584]. 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].

[645] Norman R. Paterson and Mike Livesey. Evolving caching algorithms in c by gp. In *Genetic Programming 1997: Proceedings of the Second Annual Conference*, 1997, pages 262–267. See proceedings [581].

[646] 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.

[647] 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*, 1998, pages 83–95. See proceedings [577]. Online available at http://www.grammatical-evolution.org/papers/eurogp98.ps and http://citeseer.ist.psu.edu/ryan98grammatical.html [accessed 2007-09-09].

[648] Conor Ryan and Michael O'Neill. Grammatical evolution: A steady state approach. In *Late Breaking Papers at the Genetic Programming 1998 Conference*, 1998. See proceedings [580]. Online available at http://citeseer.ist.psu.edu/260828.html and http://www.grammatical-evolution.org/papers/gp98/index.html [accessed 2007-09-09].

[649] Michael O'Neill, Finbar Leahy, and Anthony Brabazon. Grammatical swarm: A variable-length particle swarm algorithm. In Nadia Nedjah and Luiza de Macedo Mourelle, editors, *Swarm Intelligent Systems*, ISBN: 3-540-33868-3, chapter 5. Springer, 2006.

[650] 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. 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].

[651] 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*, Las Vegas, Nevada, USA, June 26-29, 2006, volume 1, pages 231–236. CSREA

Press, ISBN: 1-932415-96-3. Online available at http://ww1.ucmss.com/books/LFS/CSREA2006/ICA4864.pdf [accessed 2007-09-09].

[652] Conor Ryan. Grammatical evolution tutorial. In *Proceedings of Genetic and Evolutionary Computation Conference, GECCO 2006*, 2006. See proceedings [298]. Online available at http://www.grammaticalevolution.org/tutorial.pdf [accessed 2007-09-09].

[653] Michael O'Neill and Conor Ryan. Evolving multi-line compilable c programs. In *Proceedings of the Second European Workshop on Genetic Programming*, 1999, pages 83–92. See proceedings [576]. Online available at http://citeseer.ist.psu.edu/278127.html and http://www.grammatical-evolution.org/papers/eurogp99.ps.gz [accessed 2007-09-09].

[654] Michael O'Neill. *Automatic Programming in an Arbitrary Language: Evolving Programs with Grammatical Evolution*. PhD thesis, University of Limerick, 2001.

[655] Michael O'Neill, J. J. Collins, and Conor Ryan. Automatic programming of robots. In *Proceedings of Artificial Intelligence and Cognitive Science AICS 2000*, 2000.

[656] 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*, 2002, pages 1011–1018. See proceedings [306]. 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].

[657] 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*, London Metropolitan University, London, September 8, 2003.

[658] 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*, June 2003, volume II, pages 492–498. CSREA Press, ISBN: 1-932415-13-0.

[659] 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*, 2004, pages 138–149. See proceedings [571]. Online available at http://ncra.ucd.ie/papers/eurogp2004.pdf [accessed 2007-08-28].

[660] Michael O'Neill and Conor Ryan, editors. *The 3rd Grammatical Evolution Workshop (GEWS 2004)*, Seattle, WA, USA, June 26, 2004. Part of GECCO, see [302, 303] and also http://www.grammatical-evolution.com/gews2004/ [accessed 2007-09-10].

[661] Michael O'Neill and Conor Ryan, editors. *The 2nd Grammatical Evolution Workshop (GEWS 2003)*, Chicago, IL, USA, July

2003. Part of GECCO, see [304, 305] and also http://www.grammatical-evolution.com/gews2003/ [accessed 2007-09-10].

[662] 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].

[663] 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].

[664] 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].

[665] Henning Christiansen. Syntax, semantics, and implementation strategies for programming languages with powerful abstraction mechanisms. In *Proceedings of 18th Hawaii International Conference on System Sciences*, 1985, volume 2, pages 57–66. Also Datalogiske skrifter 1, Department of Computer Science, Roskilde University, 1985.

[666] 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)*, ISBN: 978-3-540-26319-7, ISSN: 0302-9743 (Print) 1611-3349 (Online), pages 182–191. Springer Berlin / Heidelberg, August 2005. Online available at http://arantxa.ii.uam.es/~alfonsec/docs/confint/iwinac05.pdf [accessed 2007-09-09].

[667] 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*, 11:77–90, February 2007. Online available at http://arantxa.ii.uam.es/~alfonsec/artint/ieeetec.pdf [accessed 2007-09-09].

[668] 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)*, Canberra ACT, Australia, July 14, 2001, volume 1 of *ADFA Monographs in Computer Science Series*, pages 93–100, ISBN: 0-7317-0507-6. Online available at http://sc.snu.ac.kr/PAPERS/TAG3P.pdf [accessed 2007-09-09].

[669] Xuan Hoai Nguyen, Robert Ian McKay, D. L. 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*, 2002, pages 1326–1331. See proceedings [246]. Online available at `http://sc.snu.ac.kr/PAPERS/MCEC2002.pdf` [accessed 2007-09-09].

[670] Xuan Hoai Nguyen, Robert Ian McKay, and D. L. Essam. Some experimental results with tree adjunct grammar guided genetic programming. In *EuroGP '02: Proceedings of the 5th European Conference on Genetic Programming*, 2002, pages 229–238. See proceedings [573]. Online available at `http://sc.snu.ac.kr/PAPERS/TAGGGP.pdf` [accessed 2007-09-09].

[671] Xuan Hoai Nguyen, Robert Ian McKay, and D. L. 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*, June 29-July 1, 2002, volume 2, pages 1113–1117. IEEE Press. Online available at `http://sc.snu.ac.kr/PAPERS/hoaietal.pdf` [accessed 2007-09-09].

[672] Xuan Hoai Nguyen, Robert Ian McKay, and H. A. Abbass. Tree adjoining grammars, language bias, and genetic programming. In *Genetic Programming: 6th European Conference*, 2003, pages 157–183. See proceedings [572]. Online available at `http://sc.snu.ac.kr/PAPERS/eurogp03.pdf` [accessed 2007-09-10].

[673] 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.

[674] 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].

[675] 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*, 2001, pages 339–351. See proceedings [138]. See also [676].

[676] Xuan Hoai Nguyen, Robert Ian McKay, and D. L. 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*, ISBN: 978-3-540-20265-3, pages 221–236. Springer-Verlag, January 2004. See also [675]. Online available at `http://sc.snu.ac.kr/PAPERS/trigonometry.pdf` [accessed 2007-09-09].

[677] Peter Nordin. A compiling genetic programming system that directly manipulates the machine code. In *Advances in genetic programming 1*,

chapter 14, pages 311–331. The MIT Press, 1994. See collection [593]. Machine code GP Sun Spark and i868.

[678] Markus 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].

[679] Markus Brameier and Wolfgang Banzhaf. Evolving teams of predictors with linear genetic programming. *Genetic Programming and Evolvable Machines*, 2(4):381–407, December 2001. Online available at `http://dx.doi.org/10.1023/A:1012978805372` [accessed 2007-09-09].

[680] Markus 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*, 2002, pages 37–49. See proceedings [573]. Online available at `http://citeseer.ist.psu.edu/552561.html` [accessed 2007-09-09].

[681] Markus Brameier and Wolfgang Banzhaf. Neutral variations cause bloat in linear gp. In *Proceedings of 6th European Conference on Genetic Programming, EuroGP*, 2003, pages 997–1039. See proceedings [572].

[682] Markus Brameier. *On Linear Genetic Programming*. PhD thesis, Fachbereich Informatik, Universität Dortmund, February 2004. 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].

[683] 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 [686]. Online available at `http://citeseer.ist.psu.edu/86223.html` [accessed 2007-11-04].

[684] Riccardo Poli. Some steps towards a form of parallel distributed genetic programming. In *The 1st Online Workshop on Soft Computing (WSC1)*, August 19-30, 1996. Nagoya University, Japan. 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].

[685] Riccardo Poli. Evolution of graph-like programs with parallel distributed genetic programming. In *Genetic Algorithms: Proceedings of the Seventh International Conference*, 1997, pages 346–353. See proceedings [442]. 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].

[686] Riccardo Poli. Parallel distributed genetic programming. In *New Ideas in Optimization*, pages 403–432. McGraw-Hill Education, 1999. See collection [178] and also [683]. Online available at `http://cswww.essex.ac.uk/staff/rpoli/papers/Poli-NIO-1999-PDGP.pdf` [accessed 2007-11-04].

[687] 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. See proceedings [322]. 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].

[688] 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 [689].

[689] Riccardo Poli. Evolution of recursive transistion networks for natural language recognition with parallel distributed genetic programming. In David Corne and Jonathan L. Shapiro, editors, *Proceedings of the Workshop on Artificial Intelligence and Simulation of Behaviour (AISB) International Workshop on Evolutionary Computing*, Manchester, UK, April 7-8, 1997, volume 1305 of *Lecture Notes in Computer Science (LNCS)*, pages 163–177. Springer-Verlag, ISBN: 3-540-63476-2. 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 [688].

[690] 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)*, ISBN: 978-3-540-64954-0, ISSN: 0302-9743 (Print) 1611-3349 (Online), pages 25–35. Springer Berlin/Heidelberg, 1998. 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].

[691] 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*, 1999, volume 2, pages 1135–1142. See proceedings [314]. 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].

[692] Julian Francis Miller and Peter Thomson. Cartesian genetic programming. In *Genetic Programming, Proceedings of EuroGP'2000*, 2000,

pages 121–132. See proceedings [575]. 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].

[693] 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*, 2007, pages 1580–1587. See proceedings [297]. 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].

[694] 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*, 2001, pages 204–217. See proceedings [574]. 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].

[695] Peter John Angeline and Jordan Pollack. Evolutionary module acquisition. In *Proceedings of the Second Annual Conference on Evolutionary Programming*, 1993, pages 154–163. See proceedings [807]. 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].

[696] 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*, 2004, pages 187–197. See proceedings [571]. 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].

[697] 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)*, Sitges, Spain, September 12-14, 2005, volume 3637 of *Lecture Notes in Computer Science (LNCS)*, pages 131–142. Springer, ISBN: 3-540-28736-1. Online available at http://www.cartesiangp.co.uk/papers/2005/wmices2005.pdf [accessed 2007-11-03].

[698] 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*, 2005, pages 1649–1656. See proceedings [299]. 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].

[699] Julian Francis Miller and Peter Thomson. Aspects of digital evolution: Evolvability and architecture. In *Parallel Problem Solving from Nature – PPSN V*, 1998, pages 927–936. See proceedings [327]. 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].

[700] 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*, 1998, pages 863–868. See proceedings [579]. Online available at http://citeseer.ist.psu.edu/miller98evolving.html [accessed 2007-11-03].

[701] 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*, July 19-21, 1999, pages 54–63, Pasadena, California. IEEE Computer Society, ISBN: 0-7695-0256-3. Online available at http://citeseer.ist.psu.edu/kalganova99evolving.html [accessed 2007-11-03].

[702] Simon Harding L. and Julian Francis Miller. Evolution of robot controller using cartesian genetic programming. In *Genetic Programming, Proceedings of EuroGP 2005*, 2005, pages 62–73, ISBN: 978-3-540-25436-2. See proceedings [570]. Online available at http://www.cartesiangp.co.uk/papers/2005/hmeurogp2005.pdf [accessed]2007-11-03.

[703] 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*, 2006, pages 911–918. See proceedings [298]. Online available at http://www.cartesiangp.co.uk/papers/2006/wmgecco2006.pdf [accessed 2007-11-03].

[704] James Alfred Walker and Julian Francis Miller. Predicting prime numbers using cartesian genetic programming. In *Proceedings of the 10th European Conference on Genetic Programming*, 2007, pages 205–216. See proceedings [568].

[705] 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 [332]. See also [811].

[706] 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.

[707] Stephen Frederick Smith. *A learning system based on genetic adaptive algorithms*. PhD thesis, University of Pittsburgh, 1980. AAI8112638.

[708] 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*, 6-9 1990, pages 335–341, Herndon, VA. IEEE Computer Society Press, Los Alamitos, CA. IEEE Cat. No. 90CH2915-7.

[709] 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*, 2007, pages 2759–2764. See proceedings [296]. Online available at `http://portal.acm.org/citation.cfm?id=1274000.1274067` [accessed 2007-08-01].

[710] 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)*, 1999, pages 345–352. See proceedings [314]. Online available at `http://webspace.elet.polimi.it/lanzi/papers//lanzi1999GECCOgp.pdf` [accessed 2007-08-01].

[711] Mark A. Bedau. Artificial life: organization, adaptation and complexity from the bottom up. *TRENDS in Cognitive Sciences*, 7:505–512, 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].

[712] Lee Spector. Autoconstructive evolution: Push, pushgp, and pushpop. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2001*, 2001, pages 137–146. See proceedings [310]. Online available at `http://citeseer.ist.psu.edu/445431.html` and `http://hampshire.edu/lspector/pubs/ace.pdf` [accessed 2007-12-24].

[713] 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].

[714] Lee Spector. *Automatic Quantum Computer Programming – A Genetic Programming Approach*, volume 7 of *Genetic Programming*. (Originally published by Kluwer Academic Publishers), Springer SciencenBusiness Media, LLC, 233 Spring Street, New York, NY 10013, USA, paperback edition 2007 edition, ISBN: 0-387-36496-X, 0-387-36791-8, 978-0387-36496-4, 978-0387-36791-0, 2004. Series editor: John Koza. Library of Congress Control Number: 2006931640.

[715] 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].

[716] 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*, 2005, volume 2, pages 1689–1696. See proceedings [300]. Online available at http://doi.acm.org/10.1145/1068009.1068292 and http://hampshire.edu/lspector/pubs/push3-gecco2005.pdf [accessed 2007-12-25].

[717] 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*, 2002, pages 733–739. See proceedings [306]. Online available at http://citeseer.ist.psu.edu/608051.html and http://alum.hampshire.edu/~rpc01/gp234.pdf [accessed 2007-12-25].

[718] 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].

[719] 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*, University of New South Wales, Sydney, NSW, Australia, December 9-12, 2002, pages 142–145. MIT Press, Cambridge, MA, USA, ISBN: 0-262-69281-3. Online available at http://www.alife.org/alife8/proceedings/sub962.pdf and http://citeseer.ist.psu.edu/627720.html [accessed 2007-12-25].

[720] Martin Davis. *Engines of Logic*. W.W. Norton & Company, London, ISBN: 039-3322-297, 2000.

[721] Michael Sipser. *Introduction to the Theory of Computation*. PWS Publishing, second edition, ISBN: 053-4947-28X, 2006.

[722] Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58(2):345–363, April 1936.

[723] Alonzo Church. A note on the Entscheidungsproblem. *Journal of Symbolic Logic*, 1(1):40–41, March 1936.

[724] Alan 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].

[725] Alan 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 [724].

[726] Boris Beizer. *Software Testing Techniques*. International Thomson Computer Press, second edition, ISBN: 185-0328-803, 978-1850328803, June 1990.

[727] Mark Fewster and Dorothy Graham. *Software Test Automation*. Addison-Wesley Professional, ISBN: 020-1331-403, August 25, 1999.

[728] D. Gelperin and B. Hetzel. The growth of software testing. *Communications of the ACM*, 31:687–695, 1988. Online available at `http://doi.acm.org/10.1145/62959.62965` [accessed 2007-09-15].

[729] Cem Kaner, Jack Falk, and Hung Quoc Nguyen. *Testing Computer Software*. John Wiley and Sons, second edition, ISBN: 047-1358-460, 978-0471358466, April 12, 1993.

[730] Sean Luke. Code growth is not caused by introns. In *Late Breaking Papers at the 2000 Genetic and Evolutionary Computation Conference*, 2000, pages 228–235. See proceedings [313]. Online available at `http://citeseer.ist.psu.edu/300709.html` and `http://www.cs.gmu.edu/~sean/papers/intronpaper.pdf` [accessed 2007-09-07].

[731] 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, Saarbrücken)*, 1994, pages 33–38, Im Stadtwald, Building 44, D-66123 Saarbrücken, Germany. Max-Planck-Institut für Informatik (MPI-I-94-241). 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].

[732] William B. Langdon and Riccardo Poli. Fitness causes bloat: Mutation. In *Proceedings of the First European Workshop on Genetic Programming*, 1998, pages 37–48. See proceedings [577]. 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].

[733] Walter Alden Tackett. *Recombination, selection, and the genetic construction of computer programs*. PhD thesis, University of Southern California, Los Angeles, CA, USA, 1994. Online avail-

able at http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/ftp.io.com/papers/watphd.tar.Z [accessed 2007-09-07].

[734] Nicholas Freitag McPhee and Justin Darwin Miller. Accurate replication in genetic programming. In *Genetic Algorithms: Proceedings of the Sixth International Conference (ICGA95)*, 1995, pages 303–309. See proceedings [443]. 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].

[735] Justinian Rosca. Generality versus size in genetic programming. In *Genetic Programming 1996: Proceedings of the First Annual Conference*, 1996, pages 381–387. See proceedings [583]. 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].

[736] Tobias Blickle. *Theory of Evolutionary Algorithms and Application to System Synthesis*. PhD thesis, ETH Zurich, November 1996, ISBN: 978-3728124333. 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].

[737] 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. The MIT Press, 1999. See collection [595]. 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].

[738] 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*, 1995, pages 6–22. See proceedings [1320] and also [739]. Online available at http://citeseer.ist.psu.edu/nordin95explicitly.html and http://en.scientificcommons.org/362050 [accessed 2007-09-07].

[739] Peter Nordin, Frank Francone, and Wolfgang Banzhaf. Explicitly defined introns and destructive crossover in genetic programming. In Peter John Angeline and Jr Kenneth E. Kinnear, editors, *Advances in genetic programming 2*, pages 111–134. The MIT Press, 1996. See collection [594] and also [738]. Online available at ftp://lumpi.informatik.uni-dortmund.de/pub/biocomp/papers/ML95.ps.gz [accessed 2007-08-17].

[740] 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*, 1998, pages 781–186. See proceedings [250]. Online available at http://citeseer.ist.psu.edu/313655.html [accessed 2007-09-07].

[741] Stefan Bleuler, Martin Brack, Lothar Thiele, and Eckart Zitzler. Multi-objective genetic programming: Reducing bloat using SPEA2. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, 2001, pages 536–543. See proceedings [247]. 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].

[742] Sean Luke and Liviu Panait. A comparison of bloat control methods for genetic programming. *Evolutionary Computation*, 14(3):309–344, 2006.

[743] 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*, 2003, pages 204–217. See proceedings [572].

[744] Ingo Rechenberg. *Cybernetic Solution Path of an Experimental Problem*. Royal Aircraft Establishment, August 1965. Library Translation 1122, Farnborough. Reprinted in [332].

[745] Ingo Rechenberg. *Evolutionsstrategie '94*, volume 1 of *Werkstatt Bionik und Evolutionstechnik*. Frommann Holzboog, Stuttgart, ISBN: 9783772816420, September 1994.

[746] 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*, 1991, pages 2–9. See proceedings [445]. Online available at `http://de.scientificcommons.org/50038` and `http://citeseer.ist.psu.edu/19412.html` [accessed 2007-08-27].

[747] Hans-Georg Beyer and Hans-Paul Schwefel. Evolution strategies – a comprehensive introduction. *Natural Computing: an international journal*, 1(1):3–52, March 2002. Online available at `http://dx.doi.org/10.1023/A:1015059928466` and `http://www.springerlink.com/content/2311qapbrwgrcyey/` [accessed 2007-08-27].

[748] Hans-Georg Beyer. *The theory of evolution strategies*. Natural Computing Series. Springer-Verlag New York, Inc., New York, NY, USA, ISBN: 978-3-540-67297-5, 2001.

[749] 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*, 1995, pages 57–64. See proceedings [443]. Online available at `http://citeseer.ist.psu.edu/261932.html` [accessed 2007-08-27].

[750] Silja Meyer-Nieberg and Hans-Georg Beyer. Self-adaptation in evolutionary algorithms. In Fernando G. Lobo, Claudio F. Lima, and Zbigniew Michalewicz, editors, *Parameter Setting in Evolutionary Algorithms*. Springer, Berlin, 2007. Online available at `http://citeseer.ist.psu.edu/741946.html` [accessed 2007-07-28].

[751] 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*, 1996, pages 312–317. See proceedings [252]. 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].

[752] 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*, Aachen, 1997, pages 650–654. Verlag Mainz, Wissenschaftsverlag. Online available at `http://citeseer.ist.psu.edu/356709.html` [accessed 2007-08-27].

[753] 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].

[754] G. A. Jastrebski and D. V. Arnold. Improving evolution strategies through active covariance matrix adaptation. In *Proceedings of IEEE World Congress on Computational Intelligence*, Vancouver, BC, July 2006, pages 9719–9726. IEEE Press, Piscataway, New Jersey.

[755] 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*, 1999, volume 1, page 943.

[756] Peter Roosen and Fred Meyer. Determination of chemical equilibria by means of an evolution strategy. In *PPSN-II, Parallel problem solving from nature 2*, 1992, pages 411–420. See proceedings [330].

[757] 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*, 2005, pages 54–63. See proceedings [285].

[758] 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*, Plymouth, UK, April 2000, pages 55–67, New York. Springer, London. Online available at `http://citeseer.ist.psu.edu/359756.html` and `http://www.liacs.nl/~emmerich/pdf/EGG+00.pdf` [accessed 2007-08-27].

[759] 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 avail-

able at http://www.atmos-chem-phys-discuss.net/3/1333/2003/acpd-3-1333-2003.html [accessed 2007-08-27].

[760] Volker Nissen and Matthias Krause. Constrained combinatorial optimization with an evolution strategy. In *Proceedings of Fuzzy Logik, Theorie und Praxis, 4. Dortmunder Fuzzy-Tage*, Dortmund, Germany, June 1994, pages 33–40, London, UK. Springer-Verlag, ISBN: 3-540-58649-0.

[761] 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. See collection [595]. Online available at http://www.cs.bham.ac.uk/~wbl/aigp3/ch03.pdf [accessed 2007-08-18].

[762] Klaus Weinert and Jörn Mehnen. Discrete nurbs-surface approximation using an evolutionary strategy. Technical Report Sonderforschungsbereich (SFB) 531, Department of Machining Technology, University of Dortmund, Germany, October 2001. Online available at http://citeseer.ist.psu.edu/566038.html [accessed 2007-08-27].

[763] 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*, 1992, pages 361–370. See proceedings [330].

[764] 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. See collection [180].

[765] 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*, 1995, pages 33–51. See proceedings [805]. Online available at http://citeseer.ist.psu.edu/101945.html and http://de.scientificcommons.org/114486 [accessed 2007-08-27].

[766] 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*, 2006, pages 1645–1652. See proceedings [298]. Online available at http://doi.acm.org/10.1145/1143997.1144268 [accessed 2007-08-27].

[767] Frank Hoffmeister and Hans-Paul Schwefel. Korr 2.1 – an implementation of a $(\mu \dot{+} \lambda)$-evolution strategy. Technical report, Universität Dortmund, Fachbereich Informatik, November 1990. Interner Bericht.

[768] 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].

[769] 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*, 2002, volume 2, pages 1848–1853. See proceedings [246].

[770] Hannes Geyer, Peter Ulbig, and Siegfried Schulz. Verschachtelte evolutionsstrategien zur optimierung nichtlinearer verfahrenstechnischer regressionsprobleme. *Chemie Ingenieur Technik*, 72(4):369–373, 2000. Online available at http://www3.interscience.wiley.com/cgi-bin/fulltext/76500452/PDFSTART [accessed 2007-08-27].

[771] Kenneth V. Price, Rainer M. Storn, and Jouni A. Lampinen. *Differential Evolution – A Practical Approach to Global Optimization*. Natural Computing Series. Springer, ISBN: 978-3-540-20950-8, October 2005.

[772] Vitaliy Feoktistov. *Differential Evolution – In Search of Solutions*, volume 5 of *Springer Optimization and Its Applications*. Springer, ISBN: 978-0-387-36895-5, December 2006.

[773] Efrñn 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*, 2006, pages 485–492. See proceedings [298]. Online available at http://portal.acm.org/citation.cfm?id=1144086 and http://delta.cs.cinvestav.mx/~ccoello/2006.html [accessed 2007-08-13].

[774] 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*, 2006, pages 35–44. See proceedings [239]. Online available at http://labraj.uni-mb.si/index.php/Bibliografija [accessed 2007-08-13].

[775] Jouni Lampinen and Ivan Zelinka. On stagnation of the differential evolution algorithm. In *Proceedings of MENDEL 2000, 6th International Mendel Conference on Soft Computing*, 2000, pages 76–83. See proceedings [164]. Online available at http://citeseer.ist.psu.edu/317991.html and http://www.lut.fi/~jlampine/MEND2000.ps [accessed 2007-08-13].

[776] 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)*, 2006, pages 505–509. See proceedings [284].

[777] 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 Sci-

ence 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].

[778] Jouni Lampinen and Ivan Zelinka. Mechanical engineering design optimization by differential evolution. In David Corne, Marco Dorigo, and Fred Glover, editors, *New Ideas in Optimization*, McGraw-Hill's Advanced Topics In Computer Science Series, pages 127–146. McGraw-Hill, London, 1999.

[779] 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].

[780] 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*, 2000, pages 410–417. See proceedings [248].

[781] 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].

[782] 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. Online available at http://scripts.iucr.org/cgi-bin/paper?S0108768102011928 [accessed 2007-08-13].

[783] 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. Online available at http://www.rsc.org/publishing/journals/CC/article.asp?doi=b609138e [accessed 2007-08-13].

[784] 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*, 2000, volume 1, pages 593–600. See proceedings [248].

[785] 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*, 1996, pages 519–523, Piscataway, NJ. IEEE Press.

[786] 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].

[787] Rainer Storn. Designing digital filters with differential evolution. In *New Ideas in Optimization*, pages 109–125. McGraw-Hill Education, 1999. See collection [178].

[788] Lawrence J. Fogel, Alvin J. Owens, and Michael J. Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley & Sons New York, ISBN: 978-0471265160, October 1966.

[789] David B. Fogel. *System Identification through Simulated Evolution: A Machine Learning Approach to Modeling*. Ginn Press, Needham Heights, MA, ISBN: 978-0536579430, June 1991.

[790] David B. Fogel. Evolving a checkers player without relying on human experience. *Intelligence*, 11(2):20–27, 2000. Online available at http://doi.acm.org/10.1145/337897.337996 [accessed 2007-08-27].

[791] 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, ISBN: 978-1558607835, September 2001.

[792] Lawrence J. Fogel and David B. Fogel. A preliminary investigation on extending evolutionary programming to include self-adaptation on finite state. *Informatica (Slovenia)*, 18(4), 1994.

[793] Vincent W. Porto, David B. Fogel, and Lawrence J. Fogel. Alternative neural network training methods. *IEEE Expert: Intelligent Systems and Their Applications*, 10(3):16–22, 1995.

[794] Daniel K. Gehlhaar, Gennady Verkhivker, Paul A. Rejto, David B. Fogel, Lawrence J. 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. See proceedings [805].

[795] 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*, 1996, pages 411–417. See proceedings [804].

[796] 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*, 1996, pages 419–429. See proceedings [804].

[797] 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*, 1996, pages 73–81. See proceedings [804].

[798] 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*, 1995, pages 503–519. See proceedings [805].

[799] Manish Sarkar. Evolutionary programming-based fuzzy clustering. In *Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming*, 1996, pages 247–256. See proceedings [804].

[800] Jong-Hwan Kim and Hyun Myung. A two-phase evolutionary programming for general constrained optimization problem. In *Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming*, 1996, pages 295–304. See proceedings [804].

[801] 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*, 1995, pages 631–644. See proceedings [805].

[802] V. William Porto, N. Saravanan, D. Waagen, and Agoston E. Eiben, editors. *Proceedings of the 7th International Conference on Evolutionary Programming VII, EP98, in cooperation with IEEE Neural Networks Council*, Mission Valley Marriott, San Diego, California, USA, March 25-27, 1998, volume 1447/1998 of *Lecture Notes in Computer Science (LNCS)*, Heidelberg, Germany. Springer Verlag, ISBN: 978-3540648918, ISSN: 0302-9743 (Print) 1611-3349 (Online).

[803] Peter John Angeline, Robert G. Reynolds, John Robert McDonnell, and Russel C. Eberhart, editors. *Proceedings of the 6th International Conference on Evolutionary Programming*, Indianapolis, Indiana, USA, 1997, volume 1213/1997 of *Lecture Notes in Computer Science (LNCS)*. Springer Verlag Telos, ISBN: 978-3540627883, ISSN: 0302-9743 (Print) 1611-3349 (Online).

[804] Lawrence J. Fogel, Peter John Angeline, and Thomas Bäck, editors. *Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming (EP'96)*, San Diego, California, USA, February 1996, Cambridge, Massachusetts. MIT Press, ISBN: 0-262-06190-2.

[805] John Robert McDonnell, Robert G. Reynolds, and David B. Fogel, editors. *Proceedings of the 4th Annual Conference on Evolutionary Programming*, San Diego, CA, USA, March 1995, A Bradford Book, Complex Adaptive Systems, Cambridge, Massachusetts. The MIT Press, ISBN: 0-262-13317-2.

[806] A.V. Sebald and Lawrence J. Fogel, editors. *Proceedings of the 3rd Annual Conference on Evolutionary Programming*, 1994, River Edge, NJ. World Scientific Publishing.

[807] David B. Fogel and W. Atmar, editors. *Proceedings of the 2nd Annual Conference on Evolutionary Programming*, La Jolla, CA, USA, February 25-26, 1993, 9363 Towne Centre Dr., San Diego, CA 92121, USA. Evolutionary Programming Society.

[808] David B. Fogel and W. Atmar, editors. *Proceedings of the 1st Annual Conference on Evolutionary Programming*, 1992, 9363 Towne Centre Dr., San Diego, CA 92121, USA. Evolutionary Programming Society.

[809] R. Davis and J. King. An overview of production systems. Technical Report STAN-CS-75-524, Stanford Computer Science Department, Stanford University, October 1975. Online available at `http://handle.dtic.mil/100.2/ADA019702` [accessed 2007-07-18].

[810] R. Davis and J. King. An overview of production systems. *Machine Intelligence*, 8, 1977.

[811] John Henry Holland and Judith S. Reitman. Cognitive systems based on adaptive algorithms. *ACM SIGART Bulletin*, 63:49, 1977. Online available at `http://doi.acm.org/10.1145/1045343.1045373` [accessed 2007-08-18]. See also [705].

[812] 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].

[813] 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].

[814] 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*, 2005, pages 63–66. See proceedings [300]. 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].

[815] 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].

[816] 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*, 2005, pages 1883–1890. See proceedings [299] and also [1321]. Online available at `http://doi.acm.org/10.1145/1068009.1068326` [accessed 2007-09-12].

[817] 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*, 2003, pages 272–278. See proceedings [136] and also [818].

[818] 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. See collection [826] and also [817].

[819] Walling Cyre. Learning grammars with a modified classifier system. In *CEC '02: Proceedings of the Congress on Evolutionary Computation*, 2002, pages 1366–1371. See proceedings [246].

[820] John H. Holmes. Discovering risk of disease with a learning classifier system. In *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA97)*, 1997. See proceedings [442]. 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].

[821] Jianhua Lin. Adaptive image quantization based on learning classifier systems. In *DCC '95: Proceedings of the Conference on Data Compression*, March 28-30, 1995, page 477, Washington, DC, USA. IEEE Computer Society.

[822] A. D. McAulay and Jae C. Oh. Image learning classifier system using genetic algorithms. In *Proceedings of IEEE National Aerospace Electronics Conference NAECON '89*, May 22-26, 1989, volume 2, pages 705–710. an IEEE conference.

[823] *The Tenth International Workshop on Learning Classifier Systems (IWLCS 2007)*, University College London, in London, England, July 8, 2007, Lecture Notes in Computer Science (LNCS) subseries Lecture Notes in Artificial Intelligence series (LNAI). Springer. Held in association with GECCO-2007 (see [296, 297]).

[824] *Ninth International Workshop on Learning Classifier Systems (IWLCS 2006)*, Seattle, WA, USA, July 8-9, 2006. Held during GECCO-2006 (see [298]).

[825] *Eighth International Workshop on Learning Classifier Systems (IWLCS-2005)*, Washington DC, USA, June 25-29, 2005. Held during GECCO-2005 (see [299, 300]). See also [826].

[826] 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, April 19, 2007, volume 4399/2007 of *Lecture Notes in Computer Science (LNCS), subseries Lecture Notes in Artificial Intelligence (LNAI)*. Springer Berlin/Heidelberg, ISBN: 978-3-540-71230-5, ISSN: 0302-9743 (Print) 1611-3349 (Online).

[827] *Seventh International Workshop on Learning Classifier Systems (IWLCS-2004)*, Seattle, Washington, USA, June 26, 2004. Held during GECCO-2004 (see [302, 303]). See also [826].

[828] *Sixth International Workshop on Learning Classifier Systems (IWLCS-2003)*, The Holiday Inn Chicago, Chicago, IL 60654, USA, July 12-16, 2003. Held during GECCO-2003 (see [304, 305]). See also [826].

[829] Pier Luca Lanzi, Wolfgang Stolzmann, and Stewart W. Wilson, editors. *Revised Papers of the 5th International Workshop on Learning Classifier Systems, IWLCS 2002*, Granada, Spain, September 7-8, 2002, volume 2661/2003 of *Lecture Notes in Computer Science (LNCS)*. Springer, ISBN: 3-540-20544-6. Published in 2003.

[830] *Advances in Learning Classifier Systems, Revices Papers of the Fourth International Workshop on Learning Classifier Systems (IWLCS-2001)*, San Francisco, CA, USA, July 7-8, 2001, volume 2321/2002 of *Lecture Notes in Computer Science (LNCS)*. Springer, ISBN: 3-540-43793-2. Held during GECCO-2001 (see [310]), published in 2002.

[831] 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)*, Paris, France, September 15-16, 2000, volume 1996/2001 of *Lecture Notes in Computer Science (LNCS)*. Springer, ISBN: 3-540-42437-7. A joint workshop of the sixth International Conference on Simulation of Adaptive Behaviour (SAB2000) and the sixth International Conference on Parallel Problem Solving from Nature (PPSN VI, see [326]), published in 2001.

[832] *The Second International Workshop on Learning Classifier Systems (IWLCS-99)*, Orlando, Florida, USA, July 13, 1999. Co-located with GECCO-99 (see [314]). Proceedings are published in [1322].

[833] *Collected Abstracts for the First International Workshop on Learning Classifier Systems (IWLCS-92)*, NASA Johnson Space Center in Houston, Texas, USA, October 6-9, 1992.

[834] 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].

[835] Andreas Geyer-Schulz. Holland classifier systems. In *APL '95: Proceedings of the international conference on Applied programming languages*, San Antonio, Texas, United States, 1995, pages 43–55, New York, NY,

USA. ACM Press, ISBN: 0-89791-722-7. Online available at http://doi.acm.org/10.1145/206913.206955 [accessed 2007-09-12].

[836] Marvin Lee Minsky. *Berechnung: Endliche und Unendliche Maschinen*. Verlag Berliner Union GmbH, Stuttgart, ISBN: 978-3408530294, 1971.

[837] Marvin Lee Minsky. *Computation: Finite and Infinite Machines*. Prentica Hall Series on Automatic Computation. Prentice Hall, ISBN: 978-0131655638, June 1967.

[838] Tommaso Toffoli. Reversible computing. In *Proceedings of the 7th Colloquium on Automata, Languages and Programming*, 1980, Lecture Notes In Computer Science (LNCS), pages 632–644, London, UK. Springer-Verlag, ISBN: 3-540-10003-2.

[839] 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.

[840] 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 [1323].

[841] Marvin Lee Minsky. Steps toward artificial intelligence. In *Proceedings of the IRE*, 1961, volume 49, pages 8–30. Reprint:Feigenbaum & Feldman, Computers and Thought, 1963, luger.

[842] Marvin Lee Minsky. Steps toward artificial intelligence. In Edward A. Feigenbaum and Julian Feldman, editors, *Computers & thought*, ISBN: 0-262-56092-5, pages 406–450. MIT Press, Cambridge, MA, USA, 1995. Online available at http://web.media.mit.edu/~minsky/papers/steps.html [accessed 2007-08-05].

[843] John Henry Holland. Properties of the bucket brigade algorithm. In *Proceedings of the International Conference on Genetic Algorithms and Their Applications*, 1985, pages 1–7. See proceedings [447].

[844] N. 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.

[845] Tim Kovacs. Two views of classifier systems. In *Fourth International Workshop on Learning Classifier Systems - IWLCS-2001*, 7 2001, pages 367–371, San Francisco, California, USA. See proceedings [830]. 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].

[846] John Henry Holland, Lashon B. 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. See collection [1322].

[847] 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 [833], 1992.

[848] 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. See collection [1322].

[849] Stewart W. Wilson and David E. Goldberg. A critical review of classifier systems. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, 1989, pages 244–255. See proceedings [371]. Online available at http://www.eskimo.com/~wilson/ps/wg.ps.gz [accessed 2007-09-12].

[850] H. Brown Cribbs, III and Robert Elliott Smith. What can i do with a learning classifier system? In Charles L. Karr and L. Michael Freeman, editors, *Industrial Applications of Genetic Algorithms*, International Series on Computational Intelligence, ISBN: 978-0849398018, pages 299–319. CRC Press, Boca Raton, FL, 1999.

[851] 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*, Sydney, Australia, August 24-30, 1991, volume 2. Online available at http://citeseer.ist.psu.edu/dejong91learning.html and http://www.cs.uwyo.edu/~wspears/papers/ijcai91.pdf [accessed 2007-09-12].

[852] 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].

[853] 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*, 2006, pages 1441–1448. See proceedings [298]. 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].

[854] 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*, 2005, pages 1843–1850. See proceedings [299]. 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].

[855] David E. Goldberg. *Computer-aided gas pipeline operation using genetic algorithms and rule learning*. PhD thesis, University of Michigan.

Ann Arbor, MI, 1983.

[856] 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*, 2006, pages 1513–1520. See proceedings [298] and also [1324]. Online available at http://doi.acm.org/10.1145/1143997.1144244 [accessed 2007-09-12].

[857] 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].

[858] Dave Cliff and Susi Ross. Adding temporary memory to zcs. *Adaptive Behavio*, 3(2):101–150, Fall 1994. Online available at ftp://ftp.informatics.sussex.ac.uk/pub/reports/csrp/csrp347.ps.Z [accessed 2007-09-12].

[859] Larry Bull and Jacob Hurst. Zcs redux. *Evolutionary Computation*, 10(2):185–205, 2002. Online available at http://www.cems.uwe.ac.uk/lcsg/papers/zcsredux.ps [accessed 2007-09-12].

[860] Larry Bull. On using zcs in a simulated continuous double-auction market. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 1999)*, 1999, pages 83–90. See proceedings [314]. Online available at http://www.cs.bham.ac.uk/~wbl/biblio/gecco1999/GA-806.pdf [accessed 2007-09-12].

[861] 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].

[862] Tim Kovacs. Xcs classifier system reliably evolves accurate, complete, and minimal representations for boolean functions. In Pravir K. Chawdry, Rajkumar Roy, and Raj K. Pant, editors, *Soft Computing in Engineering Design and Manufacturing*, ISBN: 978-3540762140, pages 59–68. Springer-Verlag, August 1998. Online available at http://www.cs.bris.ac.uk/Publications/Papers/1000239.pdf [accessed 2007-08-18].

[863] Stewart W. Wilson. Generalization in the XCS classifier system. In *Proceedings of the Third Annual Conference on Genetic Programming 1998*, 1998, pages 665–674. See proceedings [579]. Online available at http://citeseer.ist.psu.edu/wilson98generalization.html [accessed 2007-09-12].

[864] Stewart W. Wilson. State of XCS classifier system research. In Pier Luca Lanzi, Wolfgang Stolzmann, and Steward W. Wilson, editors, *Learning Classifier Systems, From Foundations to Applications*, volume 1813/2000 of *Lecture Notes in Computer Science (LNCS)*, ISBN: 3-540-67729-1, ISSN: 0302-9743 (Print) 1611-3349 (Online), pages 63–82. Springer-Verlag Berlin/Heidelberg, London, UK, 2000. Online available at http://citeseer.ist.psu.edu/72750.html and http://www.eskimo.com/~wilson/ps/state.ps.gz [accessed 2007-08-23].

[865] 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 [1325].

[866] Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, second edition, ISBN: 0137903952, December 2002.

[867] 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*, New York, NY, USA, 2004, pages 287–296, New York, NY, USA. ACM Press, ISBN: 1-58113-844-X. Online available at `http://citeseer.ist.psu.edu/xi04smart.html` [accessed 2007-09-11].

[868] 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*, 2005, volume III of *Multi-Robot Systems: From Swarms to Intelligent Automata*, pages 65–77. Springer, ISBN: 978-1-4020-3388-9. 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].

[869] 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*, August 23-26, 2004, volume 2, pages 287–290, Los Alamitos, CA, USA. IEEE Computer Society, ISSN: 1051-4651.

[870] Alexandre Temporel and Tim Kovacs. A heuristic hill climbing algorithm for mastermind. In *Proceedings of the 2003 UK Workshop on Computational Intelligence (UKCI-03)*, Department of Engineering Mathematics and Department of Computer Science The University of Bristol, UK, September 1-3, 2003, pages 189–196. University of Bristol, ISBN: 0862925371. 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].

[871] 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*, Ottawa, Canada, June 7-9, 2001, volume 2056/2001 of *Lecture Notes in Computer Science (LNCS)*, page 57. Springer Berlin / Heidelberg, ISBN: 3-540-42144-0, ISSN: 0302-9743 (Print) 1611-3349 (Online). Online available at `http://citeseer.ist.psu.edu/holte01combinatorial.html` [accessed 2007-09-11].

[872] 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, Washington, D.C., United States, 2001, volume 12, pages 903–909. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, ISBN: 0-89871-490-7. Online available at `http://portal.acm.org/citation.cfm?id=365411.365805` and `http://citeseer.ist.psu.edu/carson01hillclimbing.html` [accessed 2007-09-11].

[873] 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*, Boğaziçi University, Istanbul, Turky, June 24-25, 1993, pages 208–212. Online available at `http://citeseer.ist.psu.edu/yuret93dynamic.html` and `http://www.denizyuret.com/pub/tainn93.html` [accessed 2007-09-11].

[874] 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, ISBN: 978-0-471-33052-3, June 2003.

[875] 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, ISBN: 0-471-05669-3, November 2000.

[876] J. Matyas. Random optimization. *Automation and Remote Control*, 26:244–251, 1965.

[877] N. Baba. Convergence of a random optimization method for constrained optimization problems. *Journal of Optimization Theory and Applications*, 33(4):451–461, April 1981. Communicated by N. Avriel, Online available at `http://www.springerlink.com/content/q1g0k627688684k3/` [accessed 2007-08-26].

[878] Junyi Li and R. Russell Rhinehart. Heuristic random optimization. *Computers and Chemical Engineering*, 22:427–444, February 1998. Received 10 August 1995; revised 28 March 1996.

[879] Sandeep Chandran and R. Russell Rhinehart. Heuristic random optimizer-version ii. In *Proceedings of the American Control Conference*, 2002, volume 4, pages 2589–2594. IEEE, Piscataway NJ, US, ISSN: 0743-1619.

[880] Nimit Worakul, Wibul Wongpoowarak, and Prapaporn Boonme. Optimization in development of acetaminophen syrup formulation. *Drug Development and Industrial Pharmacy*, 28:345–351, 2002. see erratum [881].

[881] Nimit Worakul, Wibul Wongpoowarak, and Prapaporn Boonme. Optimization in development of acetaminophen syrup formulation – erratum. *Drug Development and Industrial Pharmacy*, 28:1043–10045, 2002. see paper [880].

[882] 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*, 67:197–205, January 2000. Online available at http://www3.interscience.wiley.com/cgi-bin/abstract/71003651/ [accessed 2007-08-26].

[883] 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*, 15:459–464, September 2002.

[884] Scott Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, May 13, 1983. Online available at http://fezzik.ucd.ie/msc/cscs/ga/kirkpatrick83optimization.pdf [accessed 2007-09-15].

[885] 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*, 21:1087–1092, June 1953. Online available at http://scitation.aip.org/getabs/servlet/GetabsServlet?prog=normal&id=JCPSA6000021000006001087000001&idtype=cvips&gifs=yes [accessed 2007-09-15].

[886] 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 [1326].

[887] 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. Online available at http://www.springerlink.com/content/gt0743622913gg33/fulltext.pdf [accessed 2007-09-15].

[888] 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].

[889] 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].

[890] Daniel A. Liotard. Algorithmic tools in the study of semiempirical potential surfaces. *International Journal of Quantum Chemistry*, 44(5):723–741, November 5, 1992. Online avail-

able at http://www3.interscience.wiley.com/cgi-bin/fulltext/109573913/PDFSTART [accessed 2007-09-15].

[891] 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].

[892] 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*, May 1995, volume 2434, pages 378–386.

[893] 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*, 49:707–713, June 2002.

[894] 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. Online available at stacks.iop.org/PMB/51/4217 [accessed 2007-08-25].

[895] Lester Ingber. A simple options training model. *Mathematical Computer Modelling*, 30:167–182, 1999. Online available at http://www.ingber.com/markets99_spread.pdf and http://citeseer.ist.psu.edu/ingber99simple.html [accessed 2007-08-25].

[896] 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].

[897] R. A. Rutenbar. Simulated annealing algorithms: an overview. *IEEE Circuits and Devices Magazine*, 5:19–26, January 1989.

[898] 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*, October 1994, volume 1, pages 270–275. IEEE.

[899] 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.

[900] H. Martinez-Alfaro, H. Valdez, and J. Ortega. Linkage synthesis of a four bar mechanism for n precision points using simulated anneal-

ing. In *Proceedings of the 1998 ASME Design Engineering Technical Conference*, Atlanta, USA, September 13-16, 1998, page 7.

[901] 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*, Irvine/CA, USA, August 1996, page 8.

[902] Horacio Martínez-Alfaro and Manuel Valenzuela-Rendón. Using simulated annealing for paper cutting optimization. In *MICAI 2004: Advances in Artificial Intelligence*, 2004, volume 2972/2004 of *Lecture Notes in Computer Science (LNCS)*, pages 11–20. Springer Berlin / Heidelberg, ISBN: 978-3-540-21459-5, ISSN: 0302-9743 (Print) 1611-3349 (Online).

[903] Mrinal K. Sen and Paul L. Stoffa. Nonlinear one-dimensional seismic waveform inversion using simulated annealing. *Geophysics*, 56(10):1624–1638, October 1991.

[904] P. J. M. Laarhoven and E. H. L. Aarts, editors. *Simulated annealing: theory and applications*. Mathematics and Its Applications. Springer, Kluwer Academic Publishers, Norwell, MA, USA, ISBN: 978-902772513, June 30, 1987. Reviewed in [905].

[905] Chii-Ruey Hwang. Simulated annealing: Theory and applications. *Acta Applicandae Mathematicae: An International Survey Journal on Applying Mathematics and Mathematical Applications*, 12:108–111, May 1988. Academia Sinica. This is a review of [904]. Online available at `http://www.springerlink.com/content/t1n71m85j33556m4/` [accessed 2007-08-25].

[906] Mitsunori Miki, Tomoyuki Hiroyasu, and Junya Wako. Adaptive temperature schedule determined by genetic algorithm for parallel simulated annealing. In *The 2003 Congress on Evolutionary Computation, CEC '03*, 2003. See proceedings [245]. Online available at `http://mikilab.doshisha.ac.jp/dia/research/person/wako/society/cec2003/cec03_wako.pdf` [accessed 2007-09-15].

[907] 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, ISBN: 0521750342, 978-0521750349, February 7, 2002.

[908] Fred Glover and M. Laguna. Tabu search. In Collin R. Reeves, editor, *Modern Heuristic Techniques for Combinatorial Problems*, ISBN: 978-0470220795. Blackwell Scientific Publishing, Halsted Press, Oxford, England, 1993. Online available at `http://citeseer.ist.psu.edu/glover97tabu.html` [accessed 2007-09-16].

[909] Fred Glover and M. Laguna. Tabu search. In Panos M. Pardalos and Ding-Zhu Du, editors, *Handbook of Combinatorial Optimization*, volume 3 of *Combinatorial Optimization*, pages 621–757. Kluwer Academic Publishers, Springer Netherlands, 1998.

[910] A. Hertz, E. Taillard, and D. de Werra. A tutorial on tabu search. In *Proceedings of Giornate di Lavoro AIRO'95 (Enterprise Systems: Management of Technological and Organizational Changes)*, Italy, 1995, pages 13–24. Online available at `http://citeseer.ist.psu.edu/73006.html` [accessed 2007-09-15].

[911] 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].

[912] Djurdje Cvijović and Jacek Klinowski. Taboo search: an approach to the multiple minima problem. *Science*, 267(5198):664–666, February 3, 1995.

[913] 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)*, Piscataway, October 1998, volume 3, pages 2385–2390. IEEE.

[914] Buyang Cao and Fred Glover. Tabu search and ejection chains-application to a node weighted version of the cardinality-constrained tsp. *Management Science*, 43:908–921, July 1997.

[915] 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, ISBN: 978-3-540-34559-6 (Print) 978-3-540-34560-2 (Online), ISSN: 1434-8322, 2006.

[916] 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*, 2005, pages 975–982. See proceedings [299]. Online available at `http://doi.acm.org/10.1145/1068009.1068175` [accessed 2007-08-25].

[917] 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*, 106:570–584, April 1998. Online available at `http://dx.doi.org/10.1016/S0377-2217(97)00292-0` [accessed 2007-08-25].

[918] 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*, San Diego, USA, November 2005, pages 172–178. IEEE. Online available at `http://www.man.poznan.pl/~lpopenda/HomePage/pub/CIBCB2005172.pdf` [accessed 2007-08-25].

[919] 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*, 49:88–97, February 2006.

[920] 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.

[921] 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].

[922] Luca Maria Gambardella and Marco Dorigo. Solving symmetric and asymmetric TSPs by ant colonies. In *International Conference on Evolutionary Computation*, 1996, pages 622–627. See proceedings [252]. Online available at `http://citeseer.ist.psu.edu/gambardella96solving.html` and `http://www.idsia.ch/~luca/icec96-acs.pdf` [accessed 2007-08-05].

[923] Marco Dorigo and Christian Blum. Ant colony optimization theory: a survey. *Theoretical Computer Science*, 344(2-3):243–278, 2005. Online available at `http://code.ulb.ac.be/dbfiles/DorBlu2005tcs.pdf` [accessed 2007-08-05].

[924] Marco Dorigo, Gianni Di Caro, and L. 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 [1327].

[925] 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*, Stanford, California, 1988.

[926] Vittorio Maniezzo, Luca Maria Gambardella, and Fabio de Luigi. Ant colony optimization. In Godfrey C. Onwubolu and B. V. Babu, editors, *New Optimization Techniques in Engineering*, Studies in Fuzziness and Soft Computing, ISBN: 978-3540201670, chapter 5, pages 101–117. Springer-Verlag Berlin Heidelberg, March 5, 2004. Online available at `http://www.idsia.ch/~luca/aco2004.pdf` and `http://citeseer.ist.psu.edu/maniezzo04ant.html` [accessed 2007-09-13].

[927] Pierre-Paul Grassé. La reconstruction du nid et les coordinations interindividuelles chez bellicositermes natalensis et cubitermes sp. la théorie de la stigmergie: Essai d'interprétation des termites constructeurs. *Insectes Sociaux*, 6:41–48, 1959.

[928] 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*, October 2006, pages 73–84. Online available at `http://csd.ijs.si/silc/articles/BIOMA06DASA.pdf` [accessed 2007-08-05].

[929] 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].

[930] 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*, Bergeggi, Italy, October 2003, pages 3–9. Online available at `http://www.idsia.ch/~luca/MAS2003_18.pdf` [accessed 2007-08-19].

[931] 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)*, 2000, pages 893–900. See proceedings [312]. 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].

[932] Ruud Schoonderwoerd, Owen E. Holland, Janet L. Bruten, and Leon J. M. Rothkrantz. Ant-based load balancing in telecommunications networks. *Adaptive Behavior*, 5:169–207, 1996. Online available at `http://citeseer.ist.psu.edu/schoonderwoerd96antbased.html` and `http://www.ifi.unizh.ch/ailab/teaching/FG06/` [accessed 2007-08-19].

[933] K.M. Sim and W.H. Sun. Multiple ant-colony optimization for network routing. In *First International Symposium on Cyber Worlds (CW'02)*, 2002, page 0277, Los Alamitos, CA, USA. IEEE Computer Society, ISBN: 0-7695-1862-1.

[934] Luca Maria Gambardella, É. 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. 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].

[935] 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*, October 1995, pages 39–43. IEEE Press, ISBN: 0-7803-2676-8.

[936] James Kennedy and Russel C. Eberhart. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks, 1995*, Perth, WA, Australia, November 27-December 1, 1995, volume 4, pages 1942–1948, ISBN: 0-7803-2768-3. Online available at `http://www.engr.iupui.edu/~shi/Coference/psopap4.html` [accessed 2007-08-21].

[937] 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].

[938] 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*, June 15-19, 2004, volume 3, pages 2245–2247, ISBN: 0-7803-8273-0.

[939] Russel C. Eberhart and Yuhui Shi. Comparison between genetic algorithms and particle swarm optimization. In *Proceedings of the 7th International Conference on Evolutionary Programming*, 1998, pages 611–616. See proceedings [802].

[940] 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*, 1998, pages 601–610. See proceedings [802].

[941] Michael Meissner, Michael Schmuker, and Gisbert Schneider. Optimized particle swarm optimization (opso) and its application to artificial neural network training. *BMC Bioinformatics*, 7, 2006. Online available at `http://www.biomedcentral.com/1471-2105/7/125` [accessed 2007-08-21].

[942] Thomas Kiel Rasmussen and Thiemo Krink. Improved hidden markov model training for multiple sequence alignment by a particle swarm optimization-evolutionary algorithm hybrid. *BioSystems*, 72:5–17, November 2003. Online available at `http://dx.doi.org/10.1016/S0303-2647(03)00131-X` [accessed 2007-08-21].

[943] K. E. Parsopoulos and M. N. Vrahatis. Recent approaches to global optimization problems through particle swarm optimization. *Natural Computing*, 1:235–306, June 2002. Online available at `http://citeseer.ist.psu.edu/parsopoulos02recent.html` [accessed 2007-08-21].

[944] 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].

[945] 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*, Fort Collins, Colorado, March 20-22, 2006, pages 1–12. Online available at `http://www.lania.mx/~ccoello/EMOO/baltar06.pdf.gz` and `http://hydrologydays.colostate.edu/Proceedings_2006.htm` [accessed 2007-08-21].

[946] 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*, 17:255–263, February 2003. Online available at `http://www.springerlink.com/content/j523757110202636/` and `http://www.dimitris-agrafiotis.com/` [accessed 2007-08-21].

[947] 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.*, 44:2027–2031, November 2004. Online available at `http://dx.doi.org/10.1021/ci034292+` [accessed 2007-08-21].

[948] Yuhui Shi and Marco Dorigo, editors. *Proceedings of the 2007 IEEE Swarm Intelligence Symposium. SIS'07*, Hilton Hawaiian Village Beach Resort & Spa, Hawaii, USA, April 1-5, 2007, IEEE International Symposia on Swarm Intelligence. Institute of Electrical & Electronics Enginee (IEEE). `http://www.computelligence.org/sis/2007/` [accessed 2007-08-26].

[949] *Proceedings of the 2006 IEEE Swarm Intelligence Symposium. SIS'06*, Hilton Indianapolis Hotel, Indianapolis, Indiana, USA, May 12-14, 2006, IEEE International Symposia on Swarm Intelligence. Institute of Electrical & Electronics Enginee (IEEE). `http://www.computelligence.org/sis/2006/` [accessed 2007-08-26].

[950] *Proceedings of the 2005 IEEE Swarm Intelligence Symposium. SIS2005*, The Westin Pasadena, Pasadena, California, USA, June 8-10, 2005, IEEE International Symposia on Swarm Intelligence. Institute of Electrical & Electronics Enginee (IEEE), ISBN: 978-0780389168.

[951] *Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03*, University Place Conference Center, Indiana University Purdue University Indianapolis, Indianapolis, Indiana, USA, April 24-26, 2003, IEEE International Symposia on Swarm Intelligence. Institute of Electrical & Electronics Enginee (IEEE), ISBN: 978-0780379145. Cat. No.03EX706, `http://www.computelligence.org/sis/2003/` [accessed 2007-08-26].

[952] 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].

[953] 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].

[954] Jason Digalakis and Konstantinos Margaritis. A parallel memetic algorithm for solving optimization problems. In *Proceedings of the 4th Metaheuristics International Conference*, 2001, pages 121–125. See proceedings [173]. Online available at http://citeseer.ist.psu.edu/digalakis01parallel.html and http://www.it.uom.gr/people/digalakis/digiasfull.pdf [accessed 2007-09-12].

[955] 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*, 2001, pages 85–90. See proceedings [173]. Online available at http://citeseer.ist.psu.edu/458772.html and http://citeseer.ist.psu.edu/649471.html [accessed 2007-09-12].

[956] Luciana Buriol, Paulo M. França, and Pablo Moscato. A new memetic algorithm for the asymmetric traveling salesman problem. *Journal of Heuristics*, 10:483–506, September 2004. Online available at http://www.springerlink.com/content/w617486q60mphg88/fulltext.pdf and http://citeseer.ist.psu.edu/544761.html [accessed 2007-09-12].

[957] 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, ISBN: 0-2625-3196-8, first edition: 978-0262031417, August 2001. First edition June 1990.

[958] 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].

[959] Barry D. Hughes. *Random Walks and Random Environments: Volume 1: Random Walks*. Oxford University Press, USA, ISBN: 0198537883, 978-0198537885, May 16, 1995.

[960] William Feller. *An Introduction to Probability Theory and Its Applications*, volume 1. Wiley, 3 edition, ISBN: 0471257087,978-0471257080, 1968.

[961] 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].

[962] John H. Gillespie. Molecular evolution over the mutational landscape. *Evolution*, 38(5):1116–1129, September 1984.

[963] John Maynard Smith. Natural selection and the concept of a protein space. *Nature*, 225:563–564, February 7, 1970. Received November 7, 1969.

[964] Herb Sutter and James Larus. Software and the concurrency revolution. *Queue*, 3(7):54–62, 2005. Online available at `http://research.microsoft.com/~larus/Papers/queue01.pdf` and `http://portal.acm.org/citation.cfm?id=1095421` [accessed 2007-08-13].

[965] 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*, Seattle, Washington, United States, 1989, pages 309–319, New York, NY, USA. ACM Press, ISBN: 0-89791-307-8. Online available at `http://portal.acm.org/citation.cfm?id=73036` and `http://libra.msra.cn/paperDetail.aspx?id=907688` [accessed 2007-08-13].

[966] Valmir C. Barbosa. *An Introduction to Distributed Algorithms*. The MIT Press, ISBN: 978-0262024129, September 1996.

[967] Gerard Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, second edition, ISBN: 978-0521794831, January 2004.

[968] 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*, Santa Margherita Ligure, Italy, June 1995, pages 392–403. ACM Press. 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].

[969] 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*, Cambridge, Massachusetts, United States, 1996, pages 2–11, New York, NY, USA. ACM Press, ISBN: 0-89791-767-7. 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].

[970] Geoff Koch. Discovering multi-core: Extending the benefits of moores 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].

[971] 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].

[972] Erick Cant'u-Paz. A summary of research on parallel genetic algorithms. Technical report, 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].

[973] 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)*, Beijing, September 1998, pages 343–348. Online available at http://citeseer.ist.psu.edu/296101.html [accessed 2007-08-13].

[974] 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].

[975] 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*, Madrid, Spain, 2002, pages 595–602, New York, NY, USA. ACM Press, ISBN: 1-58113-445-2. Online available at http://doi.acm.org/10.1145/508791.508906 [accessed 2007-08-14].

[976] 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*, 2005, pages 811–816. See proceedings [299]. Online available at http://portal.acm.org/citation.cfm?id=1068145 [accessed 2007-08-14].

[977] Christian Gagné and Marc Parizeau. Open beagle: a c++ framework for your favorite evolutionary algorithm. *SIGEVOlution*, 1(1):12–15, 2006. Online available at http://www.sigevolution.org/2006/01/issue.pdf [accessed 2007-08-14].

[978] L. Darrell Whitley and Timothy Starkweather. Genitor ii.: A distributed genetic algorithm. *Journal of Experimental & Theoretical Artificial Intelligence,*, 2(3):189–214, July 1990.

[979] W.N. Martin, Jens Lienig, and James P. Cohoon. Island (migration) models: Evolutionary algorithms based on punctuated equilibria. In Thomas Bäck, Lawrence J. Fogel, and Zbigniew Michalewicz, editors, *Handbook of Evolutionary Computation*, chapter 6.3. IOP Publishing and Oxford University Press, may 97 release edition, 1997. Online available at http://www.cs.virginia.edu/papers/Island_Migration.pdf [accessed 2007-08-13].

[980] Zbigniew Skolicki and Kenneth A. 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*, 2005, pages 1295–1302. See proceedings [299]. Online available at http://portal.acm.org/citation.cfm?id=1068009.1068219 [accessed 2007-08-14].

[981] Zbigniew Skolicki. An analysis of island models in evolutionary computation. In *Workshop Proceedings of Genetic and Evolutionary Compu-*

tation Conference, GECCO 2005, 2005, pages 386–389. See proceedings [300]. 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].

[982] Fuey Sian Chong and William B. Langdon. Java based distributed genetic programming on the internet. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 1999, volume 2, page 1229. See proceedings [314]. Online available at http://libra.msra.cn/paperDetail.aspx?id=589775 and http://citeseer.ist.psu.edu/chong99java.html [accessed 2007-08-14].

[983] 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*, 1991, pages 150–159, ISBN: 3-540-54148-9. See proceedings [331].

[984] Reiko Tanese. Distributed genetic algorithms. In *Proceedings of the 3rd International Conference on Genetic Algorithms*, 1989, pages 434–439. See proceedings [371].

[985] Hisao Ishibuchi, Tadashi Yoshida, and Tadahiko Murata. Balance between genetic search and local search in hybrid evolutionary multicriterion optimization algorithms. In *GECCO '02: Proceedings of the Genetic and Evolutionary Computation Conference*, 2002, pages 1301–1308. See proceedings [306]. Online available at http://citeseer.ist.psu.edu/632937.html [accessed 2007-09-10].

[986] 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)*, Dallas, TX, USA, November 7-9, 2001, page 136 ff. IEEE Computer Society, ISBN: 0-7695-1417-0.

[987] 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*, 2000, pages 628–635, London, UK. Springer-Verlag, ISBN: 3-540-67442-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].

[988] Xin Yao. Optimization by genetic annealing. In M. Jabri, editor, *Proceedings of Second Australian Conference on Neural Networks*, Sydney, Australia, 1991, pages 94–97. 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].

[989] 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)*, 2004, volume 1, pages 146–151. See proceedings [244]. Online avail-

able at http://www-ra.informatik.uni-tuebingen.de/software/JCell/publications.html [accessed 2007-08-24].

[990] Robert J. Collins and David R. Jefferson. Selection in massively parallel genetic algorithms. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, 1991, pages 249–256. See proceedings [445]. Online available at http://citeseer.ist.psu.edu/349839.html [accessed 2007-11-30].

[991] Heinz Mühlenbein. Evolution in time and space – the parallel genetic algorithm. In *Foundations of Genetic Algorithms 1*, 1990, pages 316–337. See proceedings [439]. Online available at http://muehlenbein.org/parallel.PDF and http://citeseer.ist.psu.edu/11201.html [accessed 2007-11-30].

[992] Sean Luke and Liviu Panait. A survey and comparison of tree generation algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2001)*, 2001, pages 81–88. See proceedings [310]. Online available at http://citeseer.ist.psu.edu/luke01survey.html and http://www.cs.gmu.edu/~sean/papers/treegenalgs.pdf [accessed 2007-07-28].

[993] Edmund Burke, Steven 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*, 2002, page 341 ff. See proceedings [325]. Online available at http://citeseer.ist.psu.edu/529057.html [accessed 2007-09-07].

[994] 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 [1328, 243].

[995] Stuart Alan Kauffman and Simon Levin. Towards a general theory of adaptive walks on rugged landscapes. *Journal of Theoretical Biology*, 128(1):11–45, September 7, 1987. Online available at http://dx.doi.org/10.1016/S0022-5193(87)80029-2 [accessed 2007-11-27].

[996] 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*, Santa Fe, New Mexico, USA, June-July 1988, volume Lecture I of *Santa Fe Institute Studies in the Sciences of Complexity*, pages 527–618. Addison Wesley Publishing Company, Redwood City, ISBN: 978-0201510157, 0201510154. Published in September 1989.

[997] 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*, 141:211–245,

November 21, 1989. Online available at http://dx.doi.org/10.1016/S0022-5193(89)80019-0 [accessed 2007-10-14].

[998] Edward D. Weinberger. Local properties of kauffman's nk model, a tuneably rugged energy landscape. *Physical Review A*, 44:6399–6413, 1991.

[999] 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. Online available at http://dx.doi.org/10.1103/PhysRevE.47.2083 and http://citeseer.ist.psu.edu/94823.html [accessed 2007-11-27].

[1000] 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*, 2003, pages 3–14. See proceedings [259]. Online available at http://arxiv.org/abs/0707.0548 and http://www.i3s.unice.fr/~verel/publi/ea03-fromRRtoER.pdf [accessed 2007-12-01].

[1001] Edward D. Weinberger. 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].

[1002] 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].

[1003] Terry Jones. A description of holland's royal road function. *Evolutionary Computation*, 2:411–417, 1994. See also [1329].

[1004] Stephanie Forrest and Melanie Mitchell. Relative building-block fitness and the building-block hypothesis. In *Foundations of Genetic Algorithms 2*, 1992, pages 109–126. See proceedings [438]. 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].

[1005] Michael Defoin Platel, Manuel Clergue, and Philippe Collard. Maximum homologous crossover for linear genetic programming. In *Genetic Programming: 6th European Conference*, 2003, pages 29–48. See proceedings [572]. Online available at http://www.i3s.unice.fr/~clergue/siteCV/publi/eurogp_03.pdf [accessed 2007-12-01].

[1006] 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. See collection [594]. Online available at http://citeseer.ist.psu.edu/147908.html [accessed 2007-10-14].

[1007] Robert J. Collins and David R. Jefferson. Antfarm: Towards simulated evolution. In Christopher G. Langton, Charles Taylor, J. Doyne Farmer, and Steen Rasmussen, editors, *Artificial Life II*, pages 579–601. Addison-Wesley, Redwood City, CA, 1992. Online available at http://citeseer.ist.psu.edu/collins91antfarm.html and http://en.scientificcommons.org/108082 [accessed 2007-08-05].

[1008] Robert J. Collins and David R. Jefferson. Representations for artificial organisms. In *Proceedings of the first international conference on simulation of adaptive behavior on From animals to animats*, Paris, France, 1990, pages 382–390, Cambridge, MA, USA. MIT Press, ISBN: 0-262-63138-5.

[1009] 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 Christopher G. Langton, C.E. Taylor, D.J. Farmer, and S. Rasmussen, editors, *Artificial Life II*, pages 549–578. Addison-Wesley, 1991.

[1010] 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, ISSN: 0944-4122, 2002.

[1011] 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 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].

[1012] Ibrahim Kuscu. Evolving a generalised behavior: Artificial ant problem revisited. In *7th Annual Conference on Evolutionary Programming*, 1998, page 799 ff., ISBN: 3-540-64891-7. See proceedings [802].

[1013] 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*, 6-9 1990, pages 819–827. IEEE Computer Society Press, Los Alamitos, CA, USA. Online available at http://citeseer.ist.psu.edu/koza90genetically.html and http://www.lania.mx/~ccoello/koza90.ps.gz [accessed 2007-09-09].

[1014] Euclid of Alexandria, editor. *Stoicheia (Elements)*. Euclid of Alexandria, Alexandria, 300 BC. A series consisting of 13 books, to which two books were added lateron. 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 [1015] and http://en.wikipedia.org/wiki/Euclid%27s_Elements [accessed 2007-10-05].

[1015] Thomas L. Heath. *The Thirteen Books of Euclid's Elements*. Dover Publications, New York, second edition, ISBN: 0-486-60088-2 (vol. 1), 0-486-60089-0 (vol. 2), 0-486-60090-4 (vol. 3), 1956. Three volumes. Original publication: Cambridge University Press, 1925. Euclid's original: [1014].

[1016] 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)*, 2006, International Proceedings Series. SAICSIT, ACM, ISBN: 1-59593-567-3.

[1017] 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)*, Tucson, AZ U.S.A, March 26-29, 2007. IEEE, ISBN: 0-7695-2809-0. Online available at http://www.it-weise.de/documents/index.html#BRZWG2007AR [accessed 2007-09-15].

[1018] 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*, 2007. Springer. Online available at http://neuro.informatik.uni-ulm.de/basilic/Publications/2007/KRHB07/AdaptColorSeg-388.pdf [accessed 2007-09-15].

[1019] William J. Frawley, Gregory Piatetsky-Shapiro, and Christopher J. Matheus. Knowledge discovery in databases: An overview. *AI Magazine*, pages 213–228, Fall 1992. 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].

[1020] David J. Hand, Heikki Mannila, and Padhraic Smyth. *Principles of Data Mining*. MIT Press, Cambridge, MA, ISBN: 0-262-08290-X., August 2001.

[1021] Joseph P. Bigus. *Data Mining With Neural Networks: Solving Business Problems from Application Development to Decision Support*. Mcgraw-Hill (Tx), ISBN: 978-0070057791, May 20, 1996.

[1022] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, USA, ISBN: 978-0198538646, November 3, 1995.

[1023] Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Information Science and Statistics. Springer-Verlag, second edition, ISBN: 978-0387987804, November 19, 1999 (first edition: 1995).

[1024] Lipo Wang, editor. *Support Vector Machines: Theory and Applications*, volume 177 of *Studies in Fuzziness and Soft Computing*. Springer,

ISBN: 978-3540243885, ISSN: 1434-9922 (Print) 1860-0808 (Online), August 2005.

[1025] 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].

[1026] 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*, November 23-25, 2005, pages 160–165. Online available at `http://www.meraka.org.za/pubs/CvdWalt.pdf` [accessed 2007-08-08].

[1027] Alan Agresti. *An Introduction to Categorical Data Analysis*. Wiley-Interscience, first edition, ISBN: 978-0471113386, January 1996.

[1028] Michael J. A. Berry and Gordon S. Linoff. *Mastering Data Mining: The Art and Science of Customer Relationship Management*. Wiley, first edition, ISBN: 978-0471331230, December 1999.

[1029] 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, ISBN: 978-1558605527, October 1999.

[1030] 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. 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].

[1031] 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*, 2001, pages 41–46. American Association for Artificial Intelligence. Online available at `http://www.cc.gatech.edu/~isbell/classes/reading/papers/Rish.pdf` [accessed 2007-08-11].

[1032] 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].

[1033] Robert E. Schapire. The strength of weak learnability. *Machine Learning*, 5:197–227, June 1990. Online available at `http://www.springerlink.com/content/x02406w7q5038735/fulltext.pdf` and `http://citeseer.ist.psu.edu/schapire90strength.html` [accessed 2007-09-15].

[1034] 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*, 1995, volume 904 of

Lecture Notes In Computer Science (LNCS), pages 23–37. Springer, ISBN: 3-540-59119-2. 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 [1330].

[1035] Yoav Freund. Boosting a weak learning algorithm by majority. In *COLT: Proceedings of the third annual Workshop on Computational Learning Theory*, University of Rochester, Rochester, New York, USA, August 8-9, 1990. Morgan Kaufmann Publishers, ISBN: 1-55860-146-5. Online available at `http://citeseer.ist.psu.edu/freund95boosting.html` [accessed 2007-09-15].

[1036] 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*, Nashville, Tennessee, USA, July 1997, pages 322–330. Morgan Kaufmann, ISBN: 1-55860-486-3. Online available at `http://citeseer.ist.psu.edu/schapire97boosting.html` [accessed 2007-09-15]. See also [1331].

[1037] Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Pearson Education, Prentice Hall, second edition, ISBN: 9780133056990, September 19, 2002.

[1038] Jabir and J. W. Moore. A search for fundamental principles of software engineering. *Computer Standards & Interfaces*, 19:155–160, March 1998. 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].

[1039] Ingrid Wetzel. Information systems development with anticipation of change: Focussing on professional bureaucracies. In *Proceedings of Hawaii International Conference on Systems Sciences, HICSS 34*, Maui, Hawaii, USA, January 2001. IEEE Computer Society. 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].

[1040] 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, ISBN: 978-0-470-03614-3, April 2006.

[1041] Thomas Erl. *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*. The Prentice Hall Service-Oriented Computing Series from Thomas Erl. Prentice Hall PTR, ISBN: 978-0131858589, August 2, 2005.

[1042] 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].

[1043] 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].

[1044] 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 [1045].

[1045] 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*, 1:77–106, 2005. See also http://www.wsmo.org/ [accessed 2007-09-02] and [1044].

[1046] 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*, Vrije Universiteit Amsterdam, The Netherlands, December 12, 2005, volume RC23821 of *IBM Research Report*. Yorktown Heights: IBM Research Devision. See also [1047]. Online available at http://www.it-weise.de/documents/files/BW2005QASD.pdf [accessed 2008-1-4].

[1047] Steffen Bleul, Thomas Weise, and Kurt Geihs. An ontology for quality-aware service discovery. *Computer Systems Science Engineering*, 21(4), July 2006. See also [1046]. Special issue on "Engineering Design and Composition of Service-Oriented Applications". Online available at http://www.it-weise.de/documents/files/BWG2006QASD.pdf [accessed 2008-1-4].

[1048] Steffen Bleul and Kurt Geihs. Addo: Automatic service brokering in service oriented architectures, project homepage. Online available at http://www.vs.uni-kassel.de/ADDO/ [accessed 2007-09-02].

[1049] LSDIS Lab (Large Scale Distributed Information Systems), Department of Computer Science, University of Georgia. *METEOR-S: Semantic Web Services and Processes*, 2004. Online available at http://lsdis.cs.uga.edu/projects/meteor-s/ [accessed 2007-09-02].

[1050] Asunción Gómez-Pérez, Rafael González-Cabero, and Manuel Lama. Ode sws: A framework for designing and composing semantic web services. *IEEE Intelligent Systems*, 19:24–31, July-August 2004. Online

available at http://iswc2004.semanticweb.org/demos/14/paper.pdf [accessed 2007-09-02]. See also [1051].

[1051] 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*, History Corner, main quad (Building 200), Stanford University, CA, USA, March 22-24, 2004. Online available at http://www.daml.ecs.soton.ac.uk/SSS-SWS04/44.pdf [accessed 2007-09-02]. See also [1050].

[1052] 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*, March 29-April 1, 2005, pages 780–783. Online available at http://ws-challenge.georgetown.edu/ws-challenge/The%20EEE.htm [accessed 2007-09-02].

[1053] 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)*, 2006, pages 505–508. See proceedings [1332].

[1054] 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)*, 2006, pages 422–423. See proceedings [1333].

[1055] 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)*, 2006, pages 427–429. See proceedings [1332]. 1st place in 2006 WSC. Online available at http://www.it-weise.de/documents/files/BWG2006WSC.pdf [accessed 2008-1-4]. See 2007 WSC [1056] and [1057].

[1056] 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)*, 2007, pages 517–520. See proceedings [1333]. 2nd place in 2007 WSC. Online available at http://www.it-weise.de/documents/files/BWG2007WSC.pdf [accessed 2008-1-4]. See 2006 WSC [1055] and [1057].

[1057] Thomas Weise, Steffen Bleul, and Kurt Geihs. Web service composition systems for the web service challenge - a detailed review. Kasseler Informatikschriften (KIS) 2007, 7, Univer-

sity of Kassel, FB16, Distributed Systems Group, Wilhelmshöher Allee 73, 34121 Kassel, Germany, 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 [1055] and [1056].

[1058] 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].

[1059] 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].

[1060] IBM, BEA Systems, Microsoft, SAP AG, Siebel Systems. *BPEL4WS, Business Process Execution Language for Web Services Version 1.1*, May 2003. Online available at http://www.ibm.com/developerworks/library/specification/ws-bpel/ [accessed 2007-09-03].

[1061] 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.

[1062] Thomas Weise. *Global Optimization Algorithms – Theory and Application*. Thomas Weise, 2008-1-4 edition, 2008. Online available at http://www.it-weise.de/ [accessed 2008-1-4].

[1063] D. G. Kleinbaum, L. L. Kupper, and K. E. Muller, editors. *Applied regression analysis and other multivariable methods*. PWS Publishing Co., Boston, MA, USA, ISBN: 0-871-50123-6, 1988.

[1064] John Fox. *Applied Regression Analysis, Linear Models, and Related Methods*. Sage Publications, Inc, New York, subsequent edition, ISBN: 978-0803945401, February 1997.

[1065] D. M. Ellis. Book reviews: "applied regression analysis" by N. P. Draper and H. Smith. *Applied Statistics*, 17(1):83–84, 1968. Reviews [1066].

[1066] Norman Richard Draper and H. Smith. *Applied regression analysis*. Wiley, New York, 1966. Reviewed in [1065].

[1067] 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 [1068]. Online available at http://www.dsse.ecs.soton.ac.uk/techreports/95-03/97-3.html [accessed 2007-09-09].

[1068] Abdellah Salhi, Hugh Glaser, and David De Roure. Parallel implementation of a genetic-programming based tool for symbolic regression. *Information Processing Letters*, 66(6):299–307, June 30, 1998. Online available at http://dx.doi.org/10.1016/S0020-0190(98)00056-8 [accessed 2007-09-09]. See also [1067].

[1069] 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*, June 24-26, 1999, page 150, 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: [1070].

[1070] 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. See collection [336]. Presented at CEF'99 (see [1069]). Online available at http://www.pitt.edu/~jduffy/docs/Usr.ps [accessed 2007-09-09].

[1071] Maarten Keijzer. Scaled symbolic regression. *Genetic Programming and Evolvable Machines*, 5(3):259–269, September 2004. 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].

[1072] 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].

[1073] 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].

[1074] 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. Online available at http://doi.acm.org/10.1145/322326.322332 [accessed 2007-07-28].

[1075] 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. Online available at `http://citeseer.ist.psu.edu/634034.html` [accessed http://www.cs.arizona.edu/ bkmoon/papers/tkde-stagg.pdf]2007-09-12.

[1076] 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. Online available at `http://make.cs.nthu.edu.tw/alp/alp_paper/Evaluating%20aggregate%20operations%20over%20imprecise%20data.pdf` [accessed 2007-09-12].

[1077] 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*, August 2003, pages 847–852, Acapulco, Mexico. Morgan Kaufmann Publishers, ISBN: 0-127-05661-0. Online available at `http://citeseer.ist.psu.edu/643941.html` and `http://www.icons.rodan.pl/publications/%5BDellArmi2003%5D.pdf` [accessed 2007-09-12].

[1078] John Miles Smith and Diane C. P. Smith. Database abstractions: Aggregation and generalization. *ACM Transaction on Database Systems*, 2(2):105–133, June 1977. 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].

[1079] 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)*, ISSN: 0302-9743 (Print) 1611-3349 (Online), pages 87–92. Springer Berlin/Heidelberg, 2003.

[1080] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. Gossip-based aggregation in large dynamic networks. *ACM Trans. Comput. Syst.*, 23(3):219–252, 2005. 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].

[1081] 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)*, Cambridge, MA, USA, October 11-14, 2003, pages 482–491, Los Alamitos, CA, USA. IEEE Computer Society, ISBN: 0-7695-2040-5, ISSN: 0272-5428. 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].

[1082] 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), March 2004, pages 102–109, Tokyo, Japan. IEEE Computer Society. Online available at `http://citeseer.ist.psu.edu/jelasity04epidemicstyle.html` and `http://en.scientificcommons.org/9048443` [accessed 2007-08-13].

[1083] 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*, Seattle, Washington, United States, 1999, pages 174–185, New York, NY, USA. ACM Press, ISBN: 1-58113-142-9. 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].

[1084] 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].

[1085] 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*, 2001, pages 829–888. See proceedings [260]. 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].

[1086] 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)*, 2001, pages 219–230. See proceedings [260]. Online available at `http://fractales.inria.fr/evo-lab/EASEAComparisonFinal.ps.gz` [accessed 2007-08-24].

[1087] 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].

[1088] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *Java(TM) Language Specification*. The Java Series. Prentice Hall PTR, 3rd edition, ISBN: 978-0321246783, June 2005. Online available at `http://java.sun.com/docs/books/jls/` [accessed 2007-09-14].

[1089] 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].

[1090] Guido Krüger. *Handbuch der Java-Programmierung*. Addison-Wesley, 4. aktualisierte edition, ISBN: 3-8273-2361-4 and 3-8273-2447-5, 2006. Online available at `http://www.javabuch.de/` [accessed 2007-07-03].

[1091] Bruce Eckel. *Thinking in Java*. Prentice Hall PTR, 4th edition, ISBN: 978-0131872486, February 2006. 3rd edition online available at `http://www.mindview.net/Books/TIJ/` [accessed 2007-07-03].

[1092] David Flanagan. *Java In A Nutshell*. O'Reilly Media, Inc., 5th edition, ISBN: 978-0596007737, March 2005.

[1093] David Flanagan. *Java Examples in a Nutshell*. O'Reilly Media, Inc., 3rd, illustrated edition, ISBN: 978-0596006204, January 2004.

[1094] Pat Niemeyer and Jonathan Knudsen. *Learning Java*. O'Reilly Media, Inc, third edition, ISBN: 978-0-596-00873-4, June 2005.

[1095] Christian Ullenboom. *Java ist auch eine Insel – Programmieren mit der Java Standard Edition Version 6*. Galileo Computing, Galileo Press, 6. aktualisierte und erweiterte edition, ISBN: 3-89842-838-9, 2007. Online available at `http://www.galileocomputing.de/openbook/javainsel6/` [accessed 2007-07-03].

[1096] Yiping Ding, Ethan D. Bolker, and Arjun Kumar. Performance implications of hyper-threading. In *Proceedings of 29th International Computer Measurement Group Conference*, Dallas, Texas, USA, December 7-12, 2003, pages 21–29. Computer Measurement Group.

[1097] Dirk Büche, Sibylle D. Müller, and Petros Koumoutsakos. Self-adaptation for multi-objective evolutionary algorithms. In *Second International Conference on Evolutionary Multi-Criterion Optimization*, 2003, pages 267–281. See proceedings [267]. Online available at `http://citeseer.ist.psu.edu/570406.html` [accessed 2007-07-28].

[1098] Robert G. Reynolds and Chan-Jin Chung. Regulating the amount of information used for self-adaptation in cultural algorithm. In *Proceedings of the 7th International Conference on Genetic Algorithms*, 1997, pages 401–408. See proceedings [442].

[1099] Efren Mezura-Montes, Carlos Artemio Ceollo Coello, and Ricardo Landa-Becerra. Engineering optimization using a simple evolutionary algorithm. *ictai*, 00:149, 2003. Online available at `http://citeseer.ist.psu.edu/mezura-montes03engineering.html` [accessed 2007-07-28].

[1100] Hussein A. Abbass. The self-adaptive pareto differential evolution algorithm. In *Proceedings of Congress on Evolutionary Computation (CEC'2002)*, 2002, volume 1, pages 831–836. See proceedings [246]. Online available at `http://citeseer.ist.psu.edu/abbass02selfadaptive.html` [accessed 2007-07-28].

[1101] Marcio Nunes de Miranda, Ricardo N. B. Lima, Aloysio C. P. Pedroza, and Antônio C. de Mesquita. Hw/sw codesign of protocols based on performance optimization using genetic algorithms. In Jorge Moreira De Souza, Nelson L. S. da Fonseca, and Edmundo A. Souza e Silva, editors, *Teletraffic Engineering in the Internet Era: Proceedings of the International Teletraffic Congres*, Salvador Da Bahia, Brazil,

September 2001, volume 1 of *Teletraffic Science and Engineering, volume 4*, pages 259–269. Amsterdam: Elsevier, North-Holland Publishing Co, ISBN: 978-0444509116. Online available at `http://citeseer.ist.psu.edu/553319.html` and `http://www.gta.ufrj.br/ftp/gta/TechReports/MBAM01b.ps.gz` [accessed 2009-09-13].

[1102] B. von Haller, A.J. Ijspeert, and D. Floreano. Co-evolution of structures and controllers for neubot underwater modular robots. In Mathieu S. Capcarrere, Alex A. Freitas, Peter J. Bentley, Colin G. Johnson, and Jon Timmis, editors, *Proceedings of the VIIIth European Conference on Artificial Life ECAL 2005*, University of Kent, Canterbury, Kent (UK), September 5-9, 2005, volume 3630 of *Lecture Notes in Computer Science (LNCS) subseries Lecture Notes in Artificial Intelligence (LNAI)*, pages 189–199. Springer Verlag. Online available at `http://birg2.epfl.ch/publications/fulltext/vonhaller05.pdf` and `http://infoscience.epfl.ch/getfile.py?recid=63993` [accessed 2009-10-13].

[1103] Ernesto Benini and Andrea Toffolo. Optimal design of horizontal-axis wind turbines using blade-element theory and evolutionary computation. *Journal of Solar Energy Engineering*, 124:357–363, November 2002.

[1104] Mateen M. Rizki, Michael A. Zumda, and Louis A. Tamburino. Evolving pattern recognition systems. *IEEE transactions on evolutionary computation*, 6:594–609, December 2002.

[1105] Sidney S. Fels and Jonatas Manzolli. Interactive, evolutionary textured sound composition. In J. A. Jorge, N. M. Correia, H. Jones, and M. B. Kamegai, editors, *Proceedings of the sixth Eurographics workshop on Multimedia*, Manchester, UK, September 8-9, 2001, pages 153–164. Springer Wien, ISBN: 3-211-83769-8, ISSN: 0946-2767. Online available at `http://citeseer.comp.nus.edu.sg/fels01interactive.html` and `http://hct.ece.ubc.ca/publications/pdf/EG01.pdf` [accessed 2007-09-13].

[1106] Jeanine Graf and Wolfgang Banzhaf. Interactive evolution of images. In *Evolutionary Programming*, 1995, pages 53–65. See proceedings [805]. Online available at `http://citeseer.ist.psu.edu/110968.html` and `http://www.cs.mun.ca/~banzhaf/papers/ep95gb.ps.gz` [accessed 2007-09-13].

[1107] Frank Buschmann, Regine Meunier, Hand Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. John Wiley & Sons, ISBN: 047-1958-697, 978-0471958697, August 8, 1996.

[1108] Jan Philipps and Bernhard Rumpe. Refinement of pipe-and-filter architectures. In J. M. Wing, J. Woodcock, and J. Davies, editors, *FM'99 – Formal Methods, Proceedings of the World Congress on Formal Methods in the Development of Computing System, volume 1*, 1999, volume 1708 of *Lecture Notes in Computer Science (LNCS)*, pages

96–115. Springer, ISBN: 3-540-66587-0. Online available at http://citeseer.ist.psu.edu/269622.html and http://www4.in.tum.de/~philipps/pub/fm99.ps.gz [accessed 2007-09-15].

[1109] Robert T. Monroe, Andrew Kompanek, Ralph Melton, and David Garlan. Architectural styles, design patterns, and objects. *IEEE Software*, 14(1):43–52, January 1997. Online available at http://citeseer.ist.psu.edu/monroe97architectural.html and http://www.cs.cmu.edu/~able/publications/ObjPatternsArch-ieee97/ [accessed 2007-09-15].

[1110] A. Corana, M. Marchesi, C. Martini, and S. Ridella. Minimizing multimodal functions of continuous variables with the simulated annealing algorithm. *ACM Transactions on Mathematical Software (TOMS)*, 13(3):262–280, September 1987. Online available at http://doi.acm.org/10.1145/29380.29864 [accessed 2007-09-15].

[1111] Patrick Siarry, Gérard Berthiau, François Durdin, and Jacques Haussy. Enhanced simulated annealing for globally minimizing functions of many-continuous variables. *ACM Transactions on Mathematical Software (TOMS)*, 23(2):209–228, 1997. Online available at http://doi.acm.org/10.1145/264029.264043 [accessed 2007-09-15].

[1112] Jian Ping Li, Marton E. Balazs, Geoffrey T. Parks, and P. John Clarkson. Erratum: A species conserving genetic algorithm for multimodal function optimization. *Evolutionary Computation*, 11(1):107–109, Spring 2003.

[1113] Janez Brest, Sašo Greiner, Borko Boškovic, and Viljem Žumer. A heuristic algorithm for function optimization. In *Proceedings of MIPRO, 28th international conference on information and communication technology, electronics and microelectronics*, Opatija, Croatia, 2005, pages 91–94. Online available at http://labraj.uni-mb.si/images/9/97/AHeuristicAlgorithmForFunctionOptimization.pdf [accessed 2007-09-12].

[1114] Marcel Dekker. *Introduction to Set Theory*. CRC, revised, and expanded, third edition, ISBN: 0-8247-7915-0, June 22, 1999.

[1115] Paul R. Halmos. *Naive Set Theory*. Undergraduate Texts in Mathematics. Springer-Press New York Inc., first edition, ISBN: 0-3879-0092-6, June 2001. New Edition January 1998, ISBN: 978-0387900926.

[1116] Robert R. Stoll. *Set Theory and Logic*. Dover Publications, reprint edition, ISBN: 0-4866-3829-4, 978-0486638294, October 1, 1979.

[1117] Donald Ervin Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming (TAOCP)*. Addison-Wesley, Reading, Massachusetts, second edition, ISBN: 0-201-89685-0, 1998.

[1118] Robert Sedgewick. *Algorithms in Java, Parts 1-4*. Addison Wesley, third edition, ISBN: 0-2013-6120-5, September 2002.

[1119] 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 [1334].

[1120] Bas C. van Fraassen. *Laws and Symmetry*. Clarendon Paperbacks. Oxford University Press, USA, ISBN: 978-0198248606, January 1990.

[1121] David A. R. Wallace. *Groups, Rings and Fields*. Springer Undergraduate Mathematics Series. Springer, ISBN: 978-3540761778, February 2004. Online available at http://books.google.de/books?id=EmO9ejuMHNUC [accessed 2007-07-28].

[1122] Alfréd Rényi. *Probability Theory*. Dover Publications, ISBN: 978-0486458670, May 2007.

[1123] Gregory F. Lawler. *Introduction to Stochastic Processes*. Chapman & Hall/CRC, second edition, ISBN: 978-1584886518, May 2006.

[1124] Edwin Thompson Jaynes. *Probability Theory: The Logic of Science*. Washington University, preprint edition, 1996. See also printed version [1125]. G. Larry Bretthorst ed. Online available at http://bayes.wustl.edu/etj/prob/book.pdf [accessed 2007-08-08].

[1125] Edwin Thompson Jaynes. *Probability Theory: The Logic of Science*. Cambridge University Press, ISBN: 978-0521592710, June 2003. See also [1124]. Larry Bretthorst ed. Online available at http://bayes.wustl.edu/etj/prob/book.pdf [accessed 2007-08-08].

[1126] Olav Kallenberg. *Foundations of modern probability*. Probability and Its Applications. Springer, New York, second edition, ISBN: 978-0387953137,0-3872-5115-4, January 8, 2002. OCLC: 60740995.

[1127] Olav Kallenberg. *Probabilistic symmetries and invariance principles*. Probability and its Applications. Springer, New York, ISBN: 978-0-387-25115-8, 0-3879-4957-7, 0-3879-5313-2, July 27, 2005. OCLC: 46937587.

[1128] 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].

[1129] Richard von Mises. The unlimited extension of the validity of the exact sciences was a characteristic feature of the exaggerated rationalism of the eighteenth century. *Probability, Statistics, and Truth*, page 9, 1957.

[1130] Andrei Nikolajevich Kolmogorov. *Foundations of the Theory of Probability*. Chelsa Publishing Company New York, second edition, ISBN: 978-0828400237, 1956, June 1960. monograph "Grundbegriffe der Wahrscheinlichkeitsrechnung" 1933, book 1950, Online available at http://www.mathematik.com/Kolmogorov/ [accessed 2007-09-15].

[1131] 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].

[1132] 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].

[1133] 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.

[1134] 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. Online available at http://www.springerlink.com/content/712tn58716485674/fulltext.pdf [accessed 2007-09-15].

[1135] Donald L. Snyder and Michael I. Miller. *Random Point Processes in Time and Space*. Springer, second edition, ISBN: 978-0387975771, June 19, 1991.

[1136] 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*, ISBN: 978-0486612720, chapter 26. Dover Publications / National Bureau of Standards, first. (new ed june 1, 1965) edition, 1964.

[1137] 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. Online available at http://dx.doi.org/10.1016/j.jmva.2004.02.014 [accessed 2007-09-15].

[1138] N. J. H. Small. Miscellanea: Marginal skewness and kurtosis in testing multivariate normality. *Applied Statistics*, 29(1):85–87, 1980.

[1139] J. P. Royston. Some techniques for assessing multivariate normality based on the Shapiro-Wilk W. *Applied Statistics*, 32(2):121–133, 1983.

[1140] Luc Devroye. *Non-Uniform Random Variate Generation*. Springer-Verlag, New York, ISBN: 0-387-96305-7, 3-540-96305-7, 1986. Online available at http://cg.scs.carleton.ca/~luc/rnbookindex.html [accessed 2007-07-05].

[1141] William Sealy Gosset. The probable error of a mean. *Biometrika*, 6:1–25, 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 [1142].

[1142] William Sealy Gosset. The probable error of a mean. In E. S. Pearson and John Wishart, editors, *"Students" Collected Papers*, pages 11–34. Cambridge University Press for the Biometrika Trustees, 1942. With a Foreword by Launce McMullen. Reprint of [1141].

[1143] 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].

[1144] John A. Rice. *Mathematical Statistics and Data Analysis*. Statistics. Duxbury Press, third edition, ISBN: 978-0534399429, 978-0534209346 (2nd ed.), April 2006.

[1145] Steven M. Kay. *Fundamentals of Statistical Signal Processing, Volume I: Estimation Theory*, volume 1. Prentice Hall PTR, us es edition, ISBN: 978-0133457117, March 26, 1993.

[1146] H. Vincent Poor. *An Introduction to Signal Detection and Estimation*. Springer, second edition, ISBN: 978-0387941738, June 2005.

[1147] Harry L. Van Trees. *Detection, Estimation, and Modulation Theory, Part I*. Wiley-Interscience, reprint edition, ISBN: 978-0471095170, February 2007.

[1148] Dan Simon. *Optimal State Estimation: Kalman, H Infinity, and Nonlinear Approaches*. Wiley-Interscience, ISBN: 978-0471708582, June 2006.

[1149] John Aldrich. R.a. fisher and the making of maximum likelihood 1912-1922. *Statistical Science*, 3:162–176, January 1995. Online available at `http://projecteuclid.org/euclid.ss/1030037906` [accessed 2007-09-15].

[1150] R. L. Plackett. Some theorems in least squares. *Biometrika*, 37:149–157, 1950. Online available at `http://biomet.oxfordjournals.org/cgi/reprint/37/1-2/149` [accessed 2007-09-15].

[1151] Sir Ronald Aylmer Fisher. *Statistical methods and scientific inference*. New York, Hafner Press/Macmillan Pub Co, revised (june 1973) edition, ISBN: 0028447409, 1956.

[1152] George Casella and Roger L. Berger. *Statistical Inference*. Duxbury Thomson Learning, Pacific Grove, CA, USA, second edition, ISBN: 0-534-24312-6, 2002.

[1153] 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.

[1154] 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*, September 2005, pages 6–7, Leuven, Belgium. Online available at `http://www.best.tuke.sk/simka/pub.html` [accessed 2007-09-15].

[1155] 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 Applications – FPL 2004*, August 2004, volume 3203 of *Lecture Notes in Computer Science (LNCS)*, pages 555–564, Lueven, Belgium. Springer-Verlag. Online available at `http://www.best.tuke.sk/simka/pub.html` [accessed 2007-09-15].

[1156] 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. Online available at `http://www.cacr.math.uwaterloo.ca/~dstinson/papers/rng-IEEE.pdf` [accessed 2007-09-15].

[1157] Martin Šimka. Testing true random number generators used in cryptography. In *Proceedings of the IV. PhD students conference*, May 2004, pages 95–96. Technical University of Košice, Slovakia. Online available at `http://www.best.tuke.sk/simka/pub.html` [accessed 2007-09-15].

[1158] Ueli Maurer. Fast generation of prime numbers and secure public-key cryptographic parameters. *Journal of Cryptology*, 8(3):123–155, September 1995. Online available at `http://www.springerlink.com/content/u3710818146qq153/fulltext.pdf` and `http://citeseer.ist.psu.edu/186041.html` [accessed 2007-09-15].

[1159] Ilpo Vattulainen, T. Ala-Nissila, and K. Kankaala. Physical tests for random numbers in simulations. *Physical Review Letters*, 73:2513–2516, 1994. Online available at `http://citeseer.ist.psu.edu/vattulainen94physical.html` [accessed 2007-07-28].

[1160] Walter Selke, L.N. Shchur, and A.L. Talapov. Cluster–flipping monte carlo algorithm and correlations in "good" random number generators. *JETP Letters*, 58:684–686, 1993.

[1161] Alan M. Ferrenberg, D. P. Landau, and Y. Joanna Wong. Monte carlo simulations: Hidden errors from "good" random number generators. *Physical Review Letters*, 69:3382–3384, December 1992. Online available at `http://prola.aps.org/pdf/PRL/v69/i23/p3382_1` [accessed 2007-08-18].

[1162] 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. Online available at `http://dx.doi.org/10.1016/S0377-0427(01)00424-1` [accessed 2007-09-16].

[1163] 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].

[1164] Lenore Blum, Manuel Blum, and Michael Shub. A simple unpredictable pseudo-random number generator. *SIAM Journal on Computing*, 15:364–383, May 1986.

[1165] E. F. Carter. The generation and application of random numbers. *Forth Dimensions*, XVI(1 and 2), 1994.

[1166] Michael Hennecke. Ranexp: experimental random number generator package. *Computer Physics Communications*, 79:261–267,

April 1994. Online available at http://dx.doi.org/10.1016/0010-4655(94)90072-8 [accessed 2007-08-19].

[1167] Claude Overstreet Jr. and Richard E. Nance. A random number generator for small word-length computers. In *ACM'73: Proceedings of the annual conference*, Atlanta, Georgia, United States, 1973, pages 219–223, New York, NY, USA. ACM Press/CSC-ER. Online available at http://doi.acm.org/10.1145/800192.805707 [accessed 2007-09-15].

[1168] 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*, Santa Fe, New Mexico, April 26, 2004, page 156a. IEEE Computer Society, Los Alamitos, CA, USA, ISBN: 076-9521-320.

[1169] Derrick Henry Lehmer. Mathematical methods in large-scale computing units. In *Proceedings of the 2nd Symposium on Large-Scale Digital Calculating Machinery, 1949*, Cambridge, MA, USA, 1951, pages 141–146. Harvard University Press.

[1170] Derrick Henry Lehmer. Mathematical methods in large-scale computing units. *Math. Rev.*, 13(1):495, 1952.

[1171] 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. Online available at http://doi.acm.org/10.1145/272991.273009 [accessed 2007-09-15].

[1172] Donald Ervin Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Addison-Wesley, third edition, ISBN: 020-1896-842,978-0201038026, 1997.

[1173] George Edward Pelham Box and Mervin Edgar Muller. A note on the generation of random normal deviates. *Annals Math. Stat*, 29:610–611, 1958. Online available at http://projecteuclid.org/euclid.aoms/1177706645 [accessed 2007-09-15].

[1174] 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.

[1175] David W. Scott. *Multivariate Density Estimation: Theory, Practice, and Visualization*. Wiley Series in Probability and Statistics. Wiley-Interscience, John Wiley & Sons, ISBN: 0-4715-4770-0, August 1992.

[1176] Bernard. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman & Hall/CRC, Bristol, England, ISBN: 0-4122-4620-1, April 1986.

[1177] 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].

[1178] *Classification, Clustering, and Data Mining Applications: Proceedings of the Meeting of the International Federation of Classification Societies (IFCS), ... Data Analysis, and Knowledge Organization)*, 2004, Secaucus, NJ, USA. Springer-Verlag New York, Inc., ISBN: 3540220143.

[1179] Margaret H. Dunham. *Data Mining: Introductory and Advanced Topics*. Prentice Hall PTR, Upper Saddle River, NJ, USA, ISBN: 0130888923, August 2002.

[1180] Boris Mirkin. *Clustering for Data Mining: A Data Recovery Approach*. Computer Science and Data Analysis. Chapman & Hall/CRC, ISBN: 978-1584885344, April 2005.

[1181] 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].

[1182] Michael W. Berry, editor. *Survey of Text Mining: Clustering, Classification, and Retrieval*. Springer, ISBN: 978-0387955636, September 2003.

[1183] Mieczyslaw A. Kłopotek, Slawomir T. Wierzchoń, and Krzysztof Trojanowski, editors. *Intelligent Information Processing and Web Mining: Proceedings of the International IIS: IIPWM05*, Gdansk, Poland, June 2005, Advances in Soft Computing, Berlin, Heidelberg, New York. Springer, ISBN: 978-3-540-25055-2, ISSN: 1615-3871.

[1184] 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].

[1185] 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].

[1186] 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, ISBN: 978-0387245157, March 2005.

[1187] 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).

[1188] Sanjeev Jagannatha Koppal and Srinivasa G Narasimhan. Clustering appearance for scene analysis. In *IEEE Conference on Computer Vision and Pattern Recognition*, June 2006, volume 2, pages 1323–1330. Online available at http://www.ri.cmu.edu/pubs/pub_5376.html [accessed 2007-08-11].

[1189] 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.

[1190] 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].

[1191] 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].

[1192] 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].

[1193] 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].

[1194] 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].

[1195] 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].

[1196] 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].

[1197] Hervé Abdi. Centroïd, center of gravity, center of mass, barycenter. In Neil J. Salkind, editor, *Encyclopedia of Measurement and Statistics*, volume 1, ISBN: 978-1412916110, page 128 ff. Thousand Oaks (Califor-

nia): Sage Publications, Inc, October 2006. Online available at http://www.utdallas.edu/~herve/Abdi-Centroid2007-pretty.pdf [accessed 2007-08-11].

[1198] 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].

[1199] Anil K. Jain and Richard C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall Advanced Reference Series. Prentice-Hall, Upper Saddle River, NJ, USA, ISBN: 0-13-022278-X, 1988.

[1200] J. B. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability*, 1967, volume 1, pages 281–297. Berkeley, University of California Press.

[1201] Sergei Vassilvitskii. How slow is the k-means method? *Discrete and Computational Geometry*, June 2006. 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].

[1202] 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].

[1203] 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.

[1204] Alfred V. Aho, John E. Hopcroft, and Jeffrey D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison Wesley, international ed edition, ISBN: 978-0201000290, January 1974.

[1205] Mikhail J. Atallah and Susan Fox, editors. *Algorithms and Theory of Computation Handbook*. CRC Press, Inc., Boca Raton, FL, USA, first edition, ISBN: 978-0849326493, November 1998. Produced By Suzanne Lassandro.

[1206] Sara Baase and Allen Van Gelder. *Computer Algorithms: Introduction to Design and Analysis*. Addison Wesley, third edition, ISBN: 978-0201612448, November 1999.

[1207] Dexter C. Kozen. *The Design and Analysis of Algorithms (Texts & Monographs in Computer Science)*. Springer New York, ISBN: 978-0387976877, January 1992.

[1208] 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*, ISBN: 978-0124916500, pages 311–344. Academic Press, New York, November

1980. Papers from the Los Alamos International Research Conference on the History of Computing, 1976.

[1209] 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, ISBN: 978-1558606043, August 2004.

[1210] David Salomon. *Assemblers and loaders*. Ellis Horwood Series in Computers and Their Applications. Ellis Horwood, Upper Saddle River, NJ, USA, ISBN: 978-0130525642, 0-13-052564-2, February 1993.

[1211] Masud Ahmad Malik. Evolution of the high level programming languages: a critical perspective. *ACM SIGPLAN Notices*, 33:72–80, December 1998. Online available at `http://doi.acm.org/10.1145/307824.307882` [accessed 2007-09-14].

[1212] Donald Ervin Knuth. *Fundamental Algorithms*, volume 1 of *The Art of Computer Programming (TAOCP)*. Addison-Wesley, Reading, Massachusetts, third edition, ISBN: 0-201-89683-4, 1997.

[1213] Ingo Wegener and Randall Pruim (Translator). *Complexity Theory: Exploring the Limits of Efficient Algorithms*. Springer-Verlag, Berlin Heidelberg, ISBN: 978-3540210450, 2005. Translated from the German "Komplexitätstheorie – Grenzen der Effizienz von Algorithmen", (Springer-Verlag 2003, ISBN: 3-540-00161-1).

[1214] Herbert S. Wilf. *Algorithms and Complexity*. AK Peters, Ltd., second edition, ISBN: 978-1568811789, December 2002.

[1215] Donald E. Knuth. Big omicron and big omega and big theta. *SIGACT News*, 8(2):18–24, 1976. Online available at `http://portal.acm.org/citation.cfm?id=1008328.1008329` [accessed 2007-08-11].

[1216] Sir Charles Antony Richard (Tony) Hoare. Quicksort. *Computer Journal*, 5(1):10–15, 1962.

[1217] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge International Series on Parallel Computation. Cambridge University Press, ISBN: 978-0521474658, August 1995.

[1218] Juraj Hromkovic. *Algorithmics for Hard Computing Problems*. Springer, first edition, ISBN: 978-3540668602, June 2001.

[1219] Michael Mitzenmacher and Eli Upfal. *Probability and Computing : Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, ISBN: 0521835402, January 2005.

[1220] 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, ISBN: 978-3540239499, July 2005.

[1221] Rajeev Motwani and Prabhakar Raghavan. Randomized algorithms. *ACM Comput. Surv.*, 28(1):33–37, 1996.

[1222] 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, ISBN: 0201506041, 978-0201165036, 1995. Edited by Julia Berrisford.

[1223] P. K. Mackeown. *Stochastic Simulation in Physics*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, ISBN: 9813083263, 2001.

[1224] Christian P. Robert and George Casella. *Monte Carlo Statistical Methods (Springer Texts in Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, ISBN: 0387212396, 2005.

[1225] Jun S. Liu. *Monte Carlo Strategies in Scientific Computing*. Springer, ISBN: 0387952306, October 2002.

[1226] Friedemann Mattern. *Verteilte Basisalgorithmen*. Springer-Verlag GmbH, ISBN: 978-3540518358, 1989.

[1227] 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].

[1228] George F. Coulouris, Jean Dollimore, and Tim Kindberg. *Distributed Systems: Concepts and Design*. Addison Wesley, fourth rev. edition, ISBN: 978-0321263544, June 2005.

[1229] Andrew S. Tanenbaum and Maarten van Steen. *Distributed Systems. Principles and Paradigms*. Prentice Hall International, international ed edition, ISBN: 978-0131217867, March 2003. Some information available at `http://www.cs.vu.nl/~ast/books/ds1/` [accessed 2007-08-13].

[1230] 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].

[1231] 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)*, 2003, pages 344–350. IEEE Computer Society, ISBN: 0-7695-1919-9. Online available at `http://dks.sics.se/pub/ccgrid-dks.pdf` [accessed 2007-08-13].

[1232] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Comput. Surv.*, 36(4):335–371, 2004. 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].

[1233] 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)*, San Jose, CA, USA, January 2002. 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].

[1234] Mohammad Ilyas and Imad Mahgoub, editors. *Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems*. CRC, ISBN: 978-0849319686, July 16, 2004.

[1235] Ivan Stojmenović, editor. *Handbook of Sensor Networks: Algorithms and Architectures*. Wiley-Interscience, ISBN: 978-0-471-68472-5, October 5, 2005.

[1236] David Culler, Deborah Estrin, and Mani Srivastava. Guest editors' introduction: Overview of sensor networks. *Computer*, 37(8):41–49, August 2004. Online available at `http://www.archrock.com/downloads/resources/IEEE-overview-2004.pdf` [accessed 2007-09-15].

[1237] S. Sitharama Iyengar and Richard R. Brooks, editors. *Distributed Sensor Networks*. Chapman & Hall/CRC, ISBN: 978-1584883838, December 29, 2004.

[1238] Cauligi S. Raghavendra, Krishna M. Sivalingam, and Taieb Znati, editors. *Wireless Sensor Networks (ERCOFTAC)*. Springer Netherlands, second edition, ISBN: 978-1402078835, June 1, 2004.

[1239] Edgar H. Callaway Jr. *Wireless Sensor Networks: Architectures and Protocols*. AUERBACH, ISBN: 978-0849318238, August 26, 2003.

[1240] Feng Zhao and Leonidas Guibas. *Wireless Sensor Networks: An Information Processing Approach*. Morgan Kaufmann, ISBN: 978-1558609143, July 6, 2004.

[1241] Holger Karl and Andreas Willig. *Protocols and Architectures for Wireless Sensor Networks*. Wiley & Sons, first edition, ISBN: 978-0470095102, June 24, 2005.

[1242] Rajeev Shorey, A. Ananda, Mun Choon Chan, and Wei Tsang Ooi. *Mobile, Wireless, and Sensor Networks: Technology, Applications, and Future Directions*. Wiley-IEEE Press, ISBN: 978-0471718161, March 23, 2006.

[1243] 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*, New England Complex Systems Institute, Marriott Boston Quincy, Boston, MA, USA, June 25-30, 2006. Online available at `http://necsi.org/events/iccs6/viewpaper.php?id=188` [accessed 2007-08-01].

[1244] 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*, Berlin, Germany, 2004, volume 2920/2004 of *Lecture Notes in Computer Science (LNCS)*, pages 1–17. Springer Berlin / Heidelberg, ISBN: 978-3-540-20825-9, ISSN: 0302-9743 (Print) 1611-3349 (Online). Online available at `http://www.eureka.gme.usherb.ca/memslab/docs/PowerReview-2.pdf` [accessed 2007-08-01].

[1245] Joseph A. Paradiso and Thad Starner. Energy scavenging for mobile and wireless electronics. *IEEE Pervasive Computing*, 04(1):18–27, 2005. 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].

[1246] 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*, May 2003, pages 19–24, Taipai, Taiwan. IEEE. Online available at http://www.isi.edu/~johnh/PAPERS/Rahimi03a.pdf and http://www.citeulike.org/user/cri06/article/935917 [accessed 2007-08-01].

[1247] 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*, Tegernsee, Bavaria, Germany, October 4-6, 2006, pages 197–202, New York, NY, USA. ACM Press, ISBN: 1-59593-462-6. Online available at http://portal.acm.org/citation.cfm?id=1098918.1098980 [accessed 2007-08-01].

[1248] 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].

[1249] 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.

[1250] 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*, August 25-27, 2004, pages 446–452, Los Alamitos, CA, USA. IEEE Computer Society, ISBN: 0-7695-2189-4.

[1251] 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].

[1252] Joseph A. Paradiso. Systems for human-powered mobile computing. In *DAC '06: Proceedings of the 43rd annual conference on Design automation*, San Francisco, CA, USA, 2006, pages 645–650, New York, NY, USA. ACM Press, ISBN: 1-59593-381-6. 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].

[1253] 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*, 1998, page 132, Washington, DC, USA. IEEE Computer Society, 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].

[1254] 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)*, Berlin, Germany, January 19-21, 2004, volume 2920 of *Lecture Notes in Computer Science (LNCS)*, pages 94–105. Springer, ISBN: 3-540-20825-9. Online available at http://citeseer.ist.psu.edu/634903.html [accessed 2007-08-01].

[1255] 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)*, Bangalore, India, January 7-12, 2007. IEEE, ISBN: 1-4244-0614-5. Online available at http://jlguillaume.free.fr/www/publis/Guillaume07analysis.pdf [accessed 2007-08-01].

[1256] 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 [1335].

[1257] 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. 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].

[1258] Chee-Yee Chong and S.P. Kumar. Sensor networks: evolution, opportunities, and challenges. *Proceedings of the IEEE*, 91(8):1247–1256, August 2003.

[1259] H. Durrant-Whyte. Data fusion in sensor networks. In *Proceedings of IEEE International Conference on Video and Signal Based Surveillance AVSS '06*, Sydney, Australia, November 2006, pages 39–39, 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].

[1260] 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].

[1261] 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].

[1262] 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].

[1263] 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].

[1264] David Culler and Dr. David Tennenhouse. Largest tiny network yet – large-scale demonstration of self-organizing wireless sensor networks, August 2001. See http://webs.cs.berkeley.edu/800demo/ [accessed 2007-07-03].

[1265] 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*, Baltimore, MD, November 3-5, 2004, pages 214–226. 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].

[1266] 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*, Nashua, NH, June 9-14,

2002. Online available at http://necsi.org/events/iccs2002/NAp07_subramanian_iccs4-1.pdf [accessed 2007-08-01].

[1267] 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.

[1268] 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*, April 4-6, 2005, volume 2, pages 586–591, Washington, DC, USA. IEEE Computer Society, ISBN: 0-7695-2315-3.

[1269] M. Brian Blake. Coordinating multiple agents for workflow-oriented process orchestration. *Information Systems and E-Business Management*, 1(4):387–404, November 2003. Online available at http://www.springerlink.com/content/tk7jdgbraq0pywcr/fulltext.pdf and http://wotan.liu.edu/docis/dbl/isebbm/2003_1_4_387_CMAFWP.htm [accessed 2007-08-13].

[1270] Jesper M. Johansson, Salvatore T. March, and J. David Naumann. Modeling network latency and parallel processing in distributed database design. *Decision Sciences*, 34:677–706, November 2003. Online available at http://findarticles.com/p/articles/mi_qa3713/is_200310/ai_n9253628 [accessed 2007-08-13].

[1271] Robert Beverly, Karen Sollins, and Arthur Berger. SVM learning of IP address structure for latency prediction. In *MineNet '06: Proceedings of the 2006 SIGCOMM workshop on Mining network data*, Pisa, Italy, September 2006, pages 299–304, New York, NY, USA. ACM Press. Online available at http://www.sigcomm.org/sigcomm2006/papers/minenet-04.pdf and http://portal.acm.org/citation.cfm?id=1162678.1162682 [accessed 2007-08-13].

[1272] James Z. Wang and Matti Vanninen. Self-configuration protocols for small-scale p2p networks. In *Proceedings of 10th Network Operations and Management Symposium, NOMS 2006*, Vancouver, Cannada, April 2006, pages 1–4. IEEE/IFIP, ISBN: 1-4244-0142-9, ISSN: 1542-1201. Online available at http://www.cs.clemson.edu/~jzwang/pub/nomshort.pdf [accessed 2007-08-19].

[1273] Marshall Pease, Robert Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, 1980. Online available at http://research.microsoft.com/users/lamport/pubs/reaching.pdf and http://research.microsoft.com/users/lamport/pubs/reaching.pdf [accessed 2007-08-13].

[1274] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982. Online available at http://research.microsoft.

com/users/lamport/pubs/byz.pdf and http://portal.acm.org/citation.cfm?id=357176 [accessed 2007-08-13].

[1275] Leslie Lamport. The weak byzantine generals problem. *Journal of the ACM*, 30(3):668–676, 1983. Online available at http://doi.acm.org/10.1145/2402.322398 and http://research.microsoft.com/users/lamport/pubs/weak-byz.pdf [accessed 2007-08-13].

[1276] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002. Online available at http://research.microsoft.com/users/mcastro/publications.htm and http://portal.acm.org/citation.cfm?id=571640 [accessed 2007-08-13].

[1277] Jean-Philippe Martin, Lorenzo Alvisi, and Michael Dahlin. Minimal Byzantine storage. In *Distributed Computing, 16th international Conference, DISC 2002*, October 2002, pages 311–325, ISBN: 3-540-00073-9. Online available at http://www.cs.utexas.edu/users/jpmartin/papers/MinByz-TR.ps and http://www.springerlink.com/content/5ylhktru1bh994fv/fulltext.pdf [accessed 2007-08-13].

[1278] Jean-Philippe Martin, Lorenzo Alvisi, and Michael Dahlin. Small Byzantine quorum systems. In *Proceedings of the International Conference on Dependable Systems and Networks*, June 2002, pages 374–383. Online available at http://www.cs.utexas.edu/users/lorenzo/papers/smallByz_DSN.pdf and http://citeseer.ist.psu.edu/593329.html [accessed 2007-08-13].

[1279] Jian Yin, Jean-Philippe Martin, Arun Venkataramani, Lorenzo Alvisi, and Mike Dahlin. Separating agreement from execution for byzantine fault tolerant services. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, Bolton Landing, NY, USA, 2003, pages 253–267. ACM Press, ISBN: 1-58113-757-5. Online available at http://portal.acm.org/citation.cfm?id=1165389.945470 and http://www.cs.utexas.edu/users/lasr/papers/Yin03Separating.pdf [accessed 2007-08-13].

[1280] J-P. Martin and L. Alvisi. A framework for dynamic byzantine storage. In *Proceedings of the International Conference on Dependable Systems and Networks*, June 2004. Online available at http://www.cs.utexas.edu/users/lasr/papers/Martin05Fast.pdf [accessed 2007-08-13].

[1281] J-P. Martin and L. Alvisi. Fast Byzantine consensus. In *Proceedings of the International Conference on Dependable Systems and Networks*, June 2005, pages 402–411.

[1282] Harry C. Li, Allen Clement, Edmund L. Wong, Jeff Napper, Indrajit Roy, Lorenzo Alvisi, and Michael Dahlin. Bar gossip. In *Proceedings of the 2006 USENIX Operating Systems Design and Implementation (OSDI)*, November 2006. Online available at http://www.cs.utexas.edu/users/dahlin/papers.html [accessed 2007-08-13].

[1283] Amitanand S. Aiyer, Lorenzo Alvisi, Allen Clement, Michael Dahlin, Jean-Philippe Martin, and Carl Porth. Bar fault tolerance for co-

operative services. In *20th ACM Symposium on Operating Systems Principles*, October 2005. Online available at http://www.cs.utexas.edu/users/lorenzo/papers/sosp05.pdf and http://doi.acm.org/10.1145/1095809.1095816 [accessed 2007-08-13].

[1284] 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].

[1285] 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].

[1286] 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].

[1287] Dick Grune and Ceriel J. H. Jacobs. *Parsing techniques a practical guide*. Ellis Horwood Limited, Chichester, England, ISBN: 978-0136514312, August 1991. Online available at http://citeseer.ist.psu.edu/grune90parsing.html [accessed 2007-09-14].

[1288] 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].

[1289] 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.

[1290] András Kornai. Natural languages and the chomsky hierarchy. In *Proceedings of the second conference on European chapter of the Association for Computational Linguistics*, Geneva, Switzerland, 1985, pages 1–7, Morristown, NJ, USA. Association for Computational Linguistics. 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].

[1291] 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*, 3:299–314, May 1960. See also http://www.masswerk.at/algol60/ [accessed 2007-09-15] and [1336].

[1292] Donald E. Knuth. Backus normal form vs. backus naur form. *Communications of the ACM*, 7(12):735–736, 1964. Online available at http://doi.acm.org/10.1145/355588.365140 [accessed 2007-09-15].

[1293] International Organization for Standardization (ISO). *ISO/IEC 14977:1996: Information technology – Syntactic metalanguage – Ex-*

tended 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].

[1294] 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].

[1295] Donald E. Knuth. Semantics of context-free languages. *Theory of Computing Systems/Mathematical Systems Theory*, 2(2):127–145, 1968. See [1296]. Online available at `http://www.springerlink.com/content/m2501m07m4666813/fulltext.pdf` [accessed 2007-09-15].

[1296] Donald E. Knuth. Correction: Semantics of context-free languages. *Theory of Computing Systems/Mathematical Systems Theory*, 5(1):95–96, 1971. See [1295]. Online available at `http://www.springerlink.com/content/rj10u682v25g6506/fulltext.pdf` [accessed 2007-09-15].

[1297] Donald E. Knuth. The genesis of attribute grammars. In *WAGA: Proceedings of the international conference on Attribute grammars and their applications*, Paris, France, 1990, pages 1–12, New York, NY, USA. Springer-Verlag New York, Inc., ISBN: 0-387-53101-7. Online available at `http://www.dcs.warwick.ac.uk/~sk/cs325/gag.pdf` [accessed 2007-09-15].

[1298] Jukka Paakki. Attribute grammar paradigms – a high-level methodology in language implementation. *ACM Computing Surveys (CSUR)*, 27(2):196–255, 1995. Online available at `http://doi.acm.org/10.1145/210376.197409` [accessed 2007-09-15].

[1299] 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].

[1300] 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*, Berlin, Germany, 1991, pages 299–302, Morristown, NJ, USA. Association for Computational Linguistics. 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].

[1301] Ole Lehrmann Madsen. On defining semantics by means of extended attribute grammars. In *Semantics-Directed Compiler Generation, Proceedings of a Workshop*, Aarhus, Denmark, January 14-18, 1980, volume 94 of *Lecture Notes In Computer Science (LNCS)*, pages 259–299, London, UK. Springer-Verlag, ISBN: 3-540-10250-7, ISSN: 0302-9743 (Print) 1611-3349 (Online).

[1302] David A. Watt and Ole Lehrmann Madsen. Extended attribute grammars. *The Computer Journal*, 26:142–153, 1983. Online available at `http://comjnl.oxfordjournals.org/cgi/reprint/26/2/142` [accessed 2007-09-15]. See also: report 10, Computer Science Department, Uni-

versity of Glasgow (July 1977) and report DAIMI PB-105, Computer Science Department, Aarhus University (November 1979).

[1303] David A. Watt. An extended attribute grammar for pascal. *ACM SIGPLAN Notices*, 14(2):60–74, 1979. Online available at `http://doi.acm.org/10.1145/954063.954071` [accessed 2007-09-15].

[1304] Henning Christiansen. Programming as language development. Technical Report 15, Department of Computer Science, Roskilde University, 1988. Ph.D. thesis (summary).

[1305] Henning Christiansen. The syntax and semantics of extensible languages. Technical Report 14, Department of Computer Science, Roskilde University, 1988.

[1306] Henning Christiansen. Parsing and compilation of generative languages. Technical Report 3, Department of Computer Science, Roskilde University, 1986. abridged version: [1307].

[1307] Henning Christiansen. Recognition of generative languages. In *Proceedings of Programs as Data Objects Workshop*, Copenhagen, Denmark, October 1985, volume 217 of *Lecture Notes in Computer Science (LNCS)*, pages 63–81. Springer-Verlag, ISBN: 0-387-16446-4. Abridged version of [1306].

[1308] 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.

[1309] 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].

[1310] 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*, Paris, France, June 16-18, 2003. Online available at `http://www1.cs.columbia.edu/~rambow/papers/joshi-rambow-2003.pdf` [accessed 2007-09-15].

[1311] 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. Online available at `http://doi.acm.org/10.1145/367177.367199` and `http://citeseer.ist.psu.edu/mccarthy60recursive.html` [accessed 2007-09-15].

[1312] 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, ISBN: 978-0262130110, 0-262-13011-4, August 1962. 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 [1312].

[1313] John L. McCarthy. History of lisp. In Richard L. Wexelblat, editor, *History of Programming Languages: Proceedings of the ACM SIGPLAN Conference*, June 1978, pages 173–197. Academic Press. 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].

[1314] R. Kent Dybvig. *The SCHEME programming language*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, ISBN: 0-13-791864-X, 978-0-262-54148-0, 1987. With Illustrations by Jean-Pierre Hébert. Third edition (October 2003) Online available at http://www.scheme.com/tspl3/ [accessed 2007-09-15].

[1315] 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.

[1316] 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 [360].

[1317] 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 [519]. Online available at http://cum.unex.es/profes/profes/fcofdez/escritorio/investigacion/pgp/thesis/phd.html [accessed 2007-09-09].

[1318] William R. Swartout, editor. *Proceedings of the 10th National Conference on Artificial Intelligence, AAAI*, San Jose, California, USA, July 12-16, 1992. The AAAI Press/The MIT Press, ISBN: 0-262-51063-4. See http://www.aaai.org/Conferences/AAAI/aaai92.php [accessed 2007-09-06].

[1319] 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*, San Francisco, USA, June 12-14, 1995, page 450. AAAI Press/MIT Press, ISBN: 0-262-62102-9. 1 page poster, see also [558].

[1320] 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, Rochseter, New York, USA, July 9, 1995. Held in conjunction with the twelfth International Conference on Machine Learning.

[1321] 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 [816].

[1322] 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, ISBN: 3-540-67729-1, ISSN: 0302-9743 (Print) 1611-3349 (Online), 2000.

[1323] 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. See also [840].

[1324] 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 [856].

[1325] 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 [865].

[1326] 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 [886].

[1327] Marco Dorigo, Gianni Di Caro, and L. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5:137–172, 1999. Online available at http://www.idsia.ch/~luca/ij_23-alife99.pdf [accessed 2007-09-15]. See [924].

[1328] 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 [994, 243].

[1329] 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 [1003].

[1330] 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 [1034].

[1331] 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 [1036].

[1332] *Proceedings of 2006 IEEE Joint Conference on E-Commerce Technology (CEC'06) and Enterprise Computing, E-Commerce and E-Services (EEE'06)*, The Westin San Francisco Airport, 1 Old Bayshore Highway, Millbrae, United States, June 26-29, 2006. IEEE Computer Society, Los Alamitos, California, Washington, Tokyo, ISBN: 978-0-7695-2511-2.

[1333] IEEE Computer Society. *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)*, National Center of Sciences, Tokyo, Japan, July 23-26, 2007. IEEE Computer Society, ISBN: 978-0-7695-2913-4.

[1334] Steven R. Finch. *Mathematical Constants (Encyclopedia of Mathematics and its Applications)*. Cambridge University Press, ISBN: 978-0521818056, August 18, 2003. See `http://algo.inria.fr/bsolve/` [accessed 2007-09-15].

[1335] 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)*, 2005, pages 88–92, Washington, DC, USA. IEEE Computer Society, ISBN: 0-7695-2281-5. See also [1256].

[1336] 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. 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 [1291].

Index

List of Figures

List of Tables

List of Algorithms

List of Listings