Diss. ETH No. 11894

Theory of Evolutionary Algorithms and Application to System Synthesis

A dissertation submitted to the
SWISS FEDERAL INSTITUTE OF TECHNOLOGY
ZURICH

for the degree of
doctor of technical sciences

presented by
TOBIAS BLICKLE
Dipl.-Ing., University of Saarbrücken, Germany
born January, 17th, 1967
citizen of Germany

accepted on the recommendation of
Prof. Dr.-Ing. Lothar Thiele, examiner
Prof. Dr.-Ing. Hans-Paul Schwefel, co-examiner

1996

Ich danke
Prof. Thiele für die gute Zusammenarbeit,
Prof. Schwefel für die Motivation,
Jürgen für's Balisto Essen,
Rob für's Englisch,
Cle für's Mitmachen,
Patricia für's "fahr todi",
Renate für manche 2.-Korrektur,
Peter für die "Subventionen" und
Zürich für den Katzensee.

Contents

Kurzfassung

Die vorliegende Arbeit befasst sich mit dem probabilistischen Optimierungsverfahren der *Evolutionären Algorithmen* und deren Anwendung auf Probleme des automatisierten Entwurfs und der Optimierung komplexer digitaler Systeme, die aus Hardware- und Softwarekomponenten bestehen.

Im Teil I der Arbeit stehen Evolutionäre Algorithmen im Vordergrund. Sie sind ein heuristisches Suchverfahren, das auf eine Menge (Population) von zufällig ausgewählten Punkten (Individuen) des Lösungsraums die "Grundprinzipien" der natürlichen Evolution anwendet: Selektion und zufällige Variation. Die Evolutionären Algorithmen werden zunächst in den Kontext der globalen Optimierung mit Nebenbedingungen eingebettet. Dabei werden die wichtigsten Verfahren zur Behandlung von Nebenbedingungen vorgestellt, sowie ein neues Verfahren (IOS - individual objective switching) beschrieben und experimentell verglichen. Kapitel 4 befasst sich mit einer neuen formalen Beschreibung der Selektionsphase auf Basis von Fitnessverteilungen, die es erlaubt, Selektionsmethoden umfassender zu beschreiben, einheitlich zu betrachten und neue Erkenntnisse über den Einfluss der Parameter einer Selektionsmethode zu erlangen. Ein weiteres Kapitel ist der Variation (Rekombination) bei Evolutionären Algorithmen gewidmet. Da hier je nach Repräsentation des Problems (z.B. Vektor, Baum, Graph) völlig unterschiedliche Betrachtungen erforderlich sind, wird sich auf die Rekombination bei Baum-Repräsentation (sog. *Genetischen Programmieren*) beschränkt. Ein

wesentlicher Teil der Darlegungen behandelt das Problem des "Baumwucherns" (Bloating). Dazu wird eine neue Erklärung basierend auf der Redundanz in Bäumen vorgestellt und mit anderen Methoden zur Erklärung und Vermeidung von Wuchern verglichen.

Teil II der Dissertation ist der Anwendung der Evolutionären Algorithmen zur Optimierung komplexer digitaler Systeme gewidmet. Damit sind insbesondere Systeme gemeint, die aus Hardware- und Softwarekomponenten bestehen und die durch eine hohe Komplexität aufgrund der Heterogenität ihrer Komponenten (Hardware/Software, analog/digital, elektrisch/mechanisch) ausgezeichnet sind. Bei diesen sogenannten *anwendungsspezifischen Systemen* (oder auch *eingebetteten Systemen*) ist eine computerunterstützte Synthese auf der relativ hohen Systemebene deshalb von grossem Interesse, weil sie eine Verkürzung der Entwurfszeiten, eine Erhöhung der Zuverlässigkeit und eine Senkung der Entwurfskosten ermöglicht. Die wesentliche Aufgabe eines solchen Synthesesystems ist die Umsetzung einer verhaltensorientierten Beschreibung der Aufgabe (z.B. in Form eines Algorithmus) in eine strukturelle Darstellung aus Prozessoren, Spezialbausteinen, Speichern und Bussen unter Berücksichtigung von Nebenbedingungen (z.B. maximale Kosten, Reaktionsgeschwindigkeit, Datendurchsatz). Bei einem solchen System stellen sich vielfältige Probleme wie zum Beispiel die Schätzung der Performanz, die Optimierung der Architektur und die Exploration möglicher Entwürfe.

Diese Arbeit behandelt die formale Beschreibung eines solchen Synthesevorgangs basierend auf einem neuen Graphenmodell, in dem die gesamte Problemstellung in einen sogannten Spezifikationsgraph umgesetzt wird. Auf diesem Graphen werden die Aufgaben der Systemsynthese (Allokation, Bindung und Ablaufplanung) definiert, sowie der Synthesevorgang als globales Optimierungsproblem mit Nebenbedingungen formuliert. Dieses Optimierungsproblem wird dann mittels Evolutionärer Algorithmen gelöst, wobei die im Teil I der Arbeit gewonnen theoretischen Erkenntnisse über Evolutionäre Algorithmen gewinnbringend eingesetzt werden. Schliesslich wird am Beispiel der Synthese eines Videocodec-Chips zur Datenkompression bei Bewegtbildern die Wirksamkeit des Ansatzes demonstriert und gezeigt, dass es mit Evolutionären Algorithmen möglich ist, die Parteo-Punkte des Entwurfsraumes in einem einzigen Optimierungslauf zu bestimmen.

Abstract

The subject of this thesis are *Evolutionary Algorithms* and their application to the automated synthesis and optimization of complex digital systems composed of hardware and software elements.

In Part I the probabilistic optimization method of Evolutionary Algorithms (EAs) is presented. EAs apply the principles of natural evolution (selection and random variation) to a random set of points (population of individuals) in the search space. Evolutionary Algorithms are first embedded in the context of global optimization and the most important and widely used methods for constraint-handling are introduced, including a new method called IOS (individual objective switching). This is followed by a new formal description of selection schemes based on fitness distributions. This description enables an extensive and uniform examination of various selection schemes leading to new insights about the impact of the selection method parameters on the optimization process. Subsequently the variation (recombination) process of Evolutionary Algorithms is examined. As different analysis techniques are necessary depending on the representation of the problem (e.g. bit string, vector, tree, graph) only the recombination process for tree-representation (Genetic Programming) is considered. A major part of the explanation treats the problem of "bloating", i.e., the tree-size increase during optimization. Furthermore, a new redundancy based explanation of bloating is given and several methods to avoid bloating are compared.

Part II is dedicated to the application of Evolutionary Algorithms to the optimization of complex digital systems. These systems are composed of hardware and software components and characterized by a high complexity caused by their heterogeneity (hardware/ software, electrical/mechanical, analog/digital). Computer-aided synthesis at the abstract system level is advantageous for *application specific systems* or *embedded systems* as it enables time-to-market to be reduced with a decrease in design errors and costs. The main task of system-synthesis is the transformation of a behavioral specification (for example given by an algorithm) into a structural specification, such as a composition of processors, general or dedicated hardware modules, memories and busses, while regarding various restrictions, e.g. maximum costs, data throughput rate, reaction time. Problems related to system synthesis are for example performance estimation, architecture optimization and design-space exploration.

This thesis introduces a formal description of system-synthesis based on a new graph model where the specification is translated into a *specification graph*. The main tasks of system-synthesis (allocation, binding and scheduling) are defined for this graph and the process of system synthesis is formulated as a constrained global optimization problem. This optimization problem is solved by Evolutionary Algorithms using the results of Part I of the thesis. Finally, an example of synthesizing implementations of a video codec chip is described demonstrating the effectiveness of the proposed methodology and the capability of the EA to obtain the Pareto points of the design space in a single optimization run.

Chapter 1

Overview of the Thesis

The subject of this thesis is the analysis and application of a class of probabilistic optimization methods called *Evolutionary Algorithms (EAs).* These methods are based on the principle of organic evolution. EAs in this thesis are treated as optimizers, i.e., as a method to solve the global optimization problem. There are many other (and also fascinating) aspects of EAs such as *artificial life* or *simulated evolution* that will not be considered in this thesis.

Many problems in engineering can be formulated as optimization problems, e.g., in chip design, filter design, control, etc. In this thesis an optimization problem from the domain of system synthesis is addressed; a task that is becoming increasingly important for the design of embedded systems. Unfortunately this problem - as most other optimization problems - is *multimodal*, i.e., there exist more than one local optimum. Hence local optimization techniques (which are well elaborated) are not suitable for such problems. Therefore, methods for global optimization are important from a practical point of view. The problem of global optimization for multimodal problems is stated in Chapter 2. An overview of some optimization techniques is also presented and serves to embed Evolutionary Algorithms in the broad field of optimization techniques.

Part I of the thesis starts with the explanation of the basic principles of EAs in Chapter 3. Here different types of EAs are introduced. Furthermore, EAs are investigated with respect to optimization and, in particular, the handling of constraints is discussed. A comparison of the advantages and disadvantages of EAs completes this chapter.

These early chapters are followed by two chapters devoted to a mathematical analysis of Evolutionary Algorithms. Corresponding with the principle of EAs the analysis is split up in an analysis of the selection mechanism (Chapter 4) and an analysis of the recombination mechanism (Chapter 5). The analysis of selection is based on a new description model and opens new insights into the impact of different selection schemes on the behavior of EAs. An analysis of recombination is restricted to a special subclass of Evolutionary Algorithms, Genetic Programming. In particular, it is concerned with an explanation of the so-called *bloating phenomenon* and a comparison of techniques to improve the performance of GP. The treatment in this part of the thesis is mathematical although accompanied by some examples.

The problem solving technique established in Part I will be applied to an engineering problem in Part II, the system-level synthesis of embedded systems. Due to the enormous progress in VLSI and CAD technology more complex chip-designs can be developed in shorter time. This requires an extra level of abstraction to be applied, the system-level. System-level synthesis can be described as a mapping from a behavioral description, where the functional objects possess the granularity of algorithms, task, procedures, or processes onto a structural specification, with structural objects being general or special purpose processors, application specific integrated circuits (ASICs), buses and memories. In the search of a appropriate design methodology, there are many tradeoffs which have to be examined. The high mutual dependency of different design criteria makes it hard to automatically explore the design space. In Chapter 6 these problems will be discussed in detail as well as the requirements for a adequate model of system-synthesis. Furthermore, it will be shown how the problem of hardware/software codesign can be viewed as a special case of the system-synthesis problem.

Chapter 7 describes the task of system-synthesis formally with a new graph-based model. As a consequence the task of system- synthesis can be formulated as a global optimization problem. Chapter 8 establishes the connection with Part I of this thesis by applying a particular Evolutionary Algorithm as optimization tool to the system-synthesis problem. It also emphasizes the advantages of using an EA to solve this problem. The subsequent chapter demonstrates the suitability of the approach by modeling and synthesizing a H261 video coding and decoding system.

Finally, Chapter 10 summarizes the main results of this thesis and proposes some future directions of research.

Chapter 2

Global Optimization

A large amount of problems in engineering or management science can be formulated as global optimization problem. Unfortunately, global optimization problems are very difficult to solve. In order to understand the difficulties it is important to notice that all local optimization techniques can at most locate a local minimum. Moreover, there is no local criterion to decide whether a local solution is also the global solution. Therefore, conventional optimization methods that make use of derivatives, gradients and the like are, in general, not able to locate or identify the global optimum.

One way to solve optimization problems is to use exact methods. Unfortunately, these methods become soon computational intractable and are only available for a subclass of optimization problems. Another possibility is the application of heuristics, i.e., methods designed for a particular class of problems incorporating knowledge about the problem. The drawback of this approach is the missing guarantee of optimality. But for real-world applications one is often content with a “good” solution, even if it is not the best. Consequently, heuristics are widely used for global optimization problems.

In the remainder of this chapter the global optimization problem will first formally be established. Next, the basic features of optimization problems are discussed and finally some global optimization algorithms are presented. Although these algorithms are heuristics they can be applied to a wide range of optimization problems due to their generality. One of these algorithms will be the outline of an Evolutionary Algorithm, the main topic of this monograph.

2.1 The Global Optimization Problem

Definition 2.1 ((Constrained) Global Optimization Problem) *Let* $\mathbf{J}$ *be the search space and* $h : \tilde{\mathbf{J}} \mapsto \boldsymbol{R}$ *an objective function and* $g_i : \mathbf{J} \mapsto \boldsymbol{R}$, $i \in \{1, \ldots, q\}$ *a set of functions (called* constraints*). The* global optimization problem *is then given as the task*

$$\begin{gathered} \textit{minimize } h(J), \\ \textit{such that } g_i(J) \geq 0 \; \forall i \in \{1, \ldots, q\}, \; J \in \mathbf{J}. \end{gathered}$$

The subset $\tilde{\mathbf{J}} = \{J | g_i(J) \geq 0 \;\; \forall i \in \{1, \ldots, q\}\}$ *is called the* feasible region*. The value* $h^* := h(J^*) > -\infty$ *is called a* global minimum*, iff*

$$\forall J \in \tilde{\mathbf{J}} : h(J^*) \leq h(J) \tag{2.1}$$

and $J^* \in \tilde{\mathbf{J}}$ *is called a* global minimum point*.*

At a point $J \in \mathbf{J}$ *a constraint* g_i *is called*

satisfied *, iff* $g_i(J) \geq 0$*,*
active *, iff* $g_i(J) = 0$*,*
inactive *, iff* $g_i(J) > 0$*,*
violated *, iff* $g_i(J) < 0$*.*

Here, the optimization problem is specified as a minimization task. Most optimization problems in this thesis will be formulated this way.

This does not restrict generality, as every maximation problem can be specified as minimization problem using the relation

$$\max\{h(J)|J \in \tilde{\mathbf{J}}\} = -\min\{-h(J)|J \in \tilde{\mathbf{J}}\}.$$

Note that no specification of $\mathbf{J}$ is given, i.e., the optimization can be performed on arbitrary complex or discrete spaces. The only requirement is the existence of the mapping h for any point $J \in \tilde{\mathbf{J}}$. Often this requirement can be relaxed to the existence of $h(J)$ in the whole search space ($J \in \mathbf{J}$).

Most commonly, the search space is the Euclidean space, i.e., $\mathbf{J} \subseteq \boldsymbol{R}^d$, and the corresponding solution techniques are addressed in several textbooks on global optimization [Törn and Zilinskas, 1987; Horst and Tuy, 1992]. Classifications exist for problems and optimization algorithms. For example, in [Horst and Tuy, 1992] fundamental concepts of an optimization technique are used to separate algorithms, such as outer approximation, concavity cuts, and branch and bound techniques. In [Törn and Zilinskas, 1987] a total of seven different classifications of optimization algorithms are given.

In the following, an example is given to illustrate a problem where the search space is not Euclidean.

Example 2.1 *As an example of an optimization problem having a complex discrete space consider the space of Boolean functions that can be produced using the function set* $\mathcal{F}$=\{`AND`, `OR`, `NOT`\} *and the terminal set* $\mathcal{T}$=\{`D0, D1, D2, D3, D4, D5, D6`\}. *Each function* $J \in \mathbf{J}$ *can be represented as a tree and the set of possible functions* $\mathbf{J}$ *is countable but infinite.*

Consider now the optimization problem of finding a function y*, that performs a 6-multiplexer function, i.e., to find*

$$y(\vec{x}) = y(\mathtt{D0}, \mathtt{D1}, \mathtt{D2}, \mathtt{D3}, \mathtt{D4}, \mathtt{D5}) = \begin{cases} \mathtt{D2} & : \quad \textit{if } (\mathtt{D0}, \mathtt{D1}) = (0,0) \\ \mathtt{D3} & : \quad \textit{if } (\mathtt{D0}, \mathtt{D1}) = (0,1) \\ \mathtt{D4} & : \quad \textit{if } (\mathtt{D0}, \mathtt{D1}) = (1,0) \\ \mathtt{D5} & : \quad \textit{if } (\mathtt{D0}, \mathtt{D1}) = (1,1) \end{cases}$$

Using the following notation

$$\begin{array}{ll} \vec{x}_0 = (0,0,0,0,0,0) & y_0 = y(x_0) \\ \vec{x}_1 = (0,0,0,0,0,1) & y_1 = y(x_1) \\ \vec{x}_{63} = (1,1,1,1,1,1) & y_{63} = y(x_{63}) \end{array}$$

the objective function h can be written as

$$h(J) = \sum_{i=0}^{63} \left\{ \begin{array}{lcl} 0 & : & J(\vec{x}_i) = y_i \\ 1 & : & else \end{array} \right.$$

The objective function "counts" the number of wrongly classified patterns accoring to the 6-multiplexer function: Consequently, $h^*(J) = 0$, i.e. any program that performs the correct boolean function has an (optimal) objective value of zero.

Furthermore, assume the following constraint:

> $g_1(J) = MAX\_FKT$ *- (number of functions* `AND, OR, NOT` *used in J).*

This means that the maximum number of allowed functions in any program is restricted to MAX_FKT.

In Section 3.2.2 the Genetic Programming optimization method *will be introduced that capable of solving such optimization problems and this particular problem will also serve as an example in Chapter 5.*

2.2 Global Optimization Methods

The most important properties of optimization methods can be itemized as:

- *Accuracy:* The accuracy describes the difference between the optimal solution and the solution obtained by the optimization method. One distinguishes between *exact* and *heuristic* methods. Exact methods guarantee to find the optimum. This guarantee is paid with the complexity of the optimization method that has to be at least as high as the complexity of the problem to be solved. For example, the branch-and-bound algorithm is an exact method for solving linear optimization problems with integer restrictions. On the other hand, heuristics do obtain only near-optimal solutions. Furthermore, the accuracy of the solution often can not be predicted for these algorithms.

- *Run time* and *complexity* : The complexity of an optimization method (or an algorithm in general) is measured by the order of the number of elementary operations in dependence of the input size. The input size is the amount of data (e.g. bytes) necessary to specify the problem. As there are many different problem instances having the same problem size there are different possibilities to define the complexity. Most commonly, the complexity is measured in a worst-case asymptotic complexity. "Worst-case" means that the complexity of the algorithm is determined by the "hardest" problem of a fixed size. Because of that, NP-complete tasks are said to have an exponential running time. On the other hand, heuristics have polynomial execution time allowing problems with a several order of magnitudes higher dimensionality to be considered. Usually, the absolute complexity (run time or speed) is not of interest, as it additionally depends on the underlying machine model or implementation. Hence, the "asymptotic" complexity measures the relative increase in time with the length of the problem instance and not the absolute time. Details on algorithm complexity can be found, for example, in [Lengauer, 1990].

- *Memory demand*: The memory demand of an optimization method is an important property that may limit the applicability of the algorithm. Similar to the complexity measure a worst-case memory demand is most commonly used.

In its general form, the optimization problem described by Definition 2.1 is not specific enough to be addressed by a particular optimization method. Nevertheless some general (constructional) characteristics of optimization methods can be identified [Törn and Zilinskas, 1987; Zanakis and Evans, 1981]:

- *Utilization of all a priori information.* It is obvious, that a method that considers a priori information about the problem will outperform a method using less information. The least information that must be known (or must be computable) is the value of the objective function $h(J)$ for any $J \in \tilde{\mathbf{J}}$. Additional information could be used to restrict the search space, to use symmetries in the objective function, etc. [Wolpert and Macready, 1995] state:

  > "the particulars of the cost function are crucial, and blind faith in an algorithm to search effectively across a broad class of problems is rarely justified." [Wolpert and Macready, 1995].

- *Balancing of global reliability and local refinement.* Two competing goals have to be achieved by an optimization strategy. First, as the global minimum can be located anywhere in $\tilde{\mathbf{J}}$ no parts of the region can be neglected. Global reliability therefore corresponds to a strategy where the search points are uniformly distributed over the whole search space. Second, the assumption that the chance of finding a good point in the neighborhood of a good point is higher than in the neighborhood of a bad point. This assumption will surely be fulfilled if $\mathbf{J} \subseteq \boldsymbol{R}^d$ and h is everywhere continuous. However, in general this assumption can not be made. Nevertheless for pragmatic reasons, most optimization methods make this assumption. This leads to a strategy that focus on particular regions or in other words, that performs a local refinement of the search at "promising" points. Most global optimization methods will incorporate a mixture of these two basic strategies.

Recently, interesting facts of optimization methods have been proven, the so-called "No Free Lunch Theorems For Search" [Wolpert and Macready, 1995]. The main result is that

> "... all algorithms that search for an extremum of a cost function perform exactly the same, according to any performance measure, when averaged over all possible cost functions. (...) In short, there are no "free lunches" for effective optimization; any algorithm performs only as well as the knowledge concerning the cost functions put into the cost algorithm." [Wolpert and Macready, 1995]

In face of these facts a particular optimization method can only be "better" than an other for a specific domain of problems, i.e., for a subset of cost functions (objective functions). Unfortunately, the identification of this domain may be very complicated or impossible.

2.3 Selected Optimization Methods

In this section a (more or less subjective) selection of global optimization methods is given. The focus lies on methods that do not make great demands on the search space and the objective function. This means that the search space may be arbitrary complex and no further information of the objective function landscape is known besides the objective values $h(J), J \in \tilde{\mathbf{J}}$, i.e., no derivatives or gradients and the like. Furthermore, the algorithms introduced in this section are specified for an unconstrained optimization problem ($\tilde{\mathbf{J}} = \mathbf{J}$). The difficulties of considering constraints will be discussed in connection with constraint-handling in Evolutionary Algorithms in Section 3.3.

In this thesis, only some stochastic optimization methods will be considered. The key idea is the introduction of randomness. Most of these methods are path-oriented search methods, i.e., they start at an arbitrary point $J_0 \in \mathbf{J}$, apply some random decisions to obtain a new point J_1, and use again a random decision whether to accept this point as new starting point or not. By this, the points represent a path in the search space. The contrary approach is given by *volume-oriented* methods. Volume-oriented methods are designed accoring to global reliability strategy, i.e., they are based on the fact, that the whole search space must be scanned.

2.3.1 Monte Carlo

A simple stochastic optimization method is given by the Monte Carlo-algorithm. It randomly generates points in the search space according to some probability distribution $\mathcal{P}$ and memorizes the best point found so far (see Algorithm 1). This algorithm is a typical representative of a global reliability strategy of search: it performs a blind search through the search space. No local refinement is used.

Algorithm 1: (Monte Carlo)

Input: objective function $h : \mathbf{J} \mapsto \boldsymbol{R}$,
probability distribution $\mathcal{P}$
Output: $J \in \mathbf{J}$ with minimal objective value found so far.

```
i ← 0
J ← random_sample(J, P)
while (i < MAX_ITERATIONS) do
    J' ← random_sample(J, P)
    if (h(J') < h(J))
        J ← J'
    endif
    i ← i + 1
od
return J
```

The main drawback of this algorithm is the fixed probability distribution $\mathcal{P}$ used to sample the search space. Furthermore, no “steps back" are allowed, i.e. only improvements of the objective function are possible and it is difficult for the algorithm to escape a local optimum.

2.3.2 Hill Climbing

The Hill Climbing (HC) heuristic is very similar to the Monte Carlo algorithm. However, a new point J' is obtained by modifying the current point J using a neighborhood function $\mathcal{N}$ or mutation function (see Algorithm 2). This way the Hill Climbing algorithm explores only the region that can be reached by the neighborhood function with a probability depending on the construction of the neighborhood function. Again, this algorithm allows only improvements in the objective value.

Algorithm 2: (Hill Climbing)

Input: objective function $h : \mathbf{J} \mapsto \boldsymbol{R}$,
starting point J_0
neighborhood function $\mathcal{N}$
Output: $J \in \mathbf{J}$ with minimal objective value found so far.

```
i ← 0
J ← J_0
while (i < MAX_ITERATIONS) do
    chose J' ∈ N(J)
    if (h(J') < h(J))
        J ← J'
    endif
    i ← i + 1
od
return J
```

2.3.3 Simulated Annealing

Simulated Annealing (SA) is inspired by the physical behaviour of material during the annealing process. The annealing is performed by many small decreases in temperature. Only when having reached a thermal equilibrium (Boltzmann-distribution) the temperature is decreased further. By this, a regular crystal lattice is obtained and the system ends up in the state of minimum energy.

This physical process is translated into the following optimization algorithm (see Algorithm 3): The energy function corresponds to the objective function of the optimization problem. A neighborhood function $\mathcal{N}(J)$ has to be specified for each problem and creates new points $J' \in \mathbf{J}$ from a given point $J \in \mathbf{J}$. The algorithm starts with a single initial point J_0 (also termed *initial configuration*) and a given temperature T_0. The temperature is decreased in an outer loop accoring to a *cooling scheme* until a stopping criterion is fulfilled. In each step of this loop a quasi-Boltzmann-distribution is obtained by randomly creating configurations J' and accepting or rejecting the new points with probability

$$P(\Delta h) = \begin{cases} 1 & : \quad \Delta h \leq 0 \\ e^{\frac{-\Delta h}{T}} & : \quad \text{else} \end{cases}$$

where $\Delta h = h(J') - h(J)$.

The main parameters of the algorithm are the cooling scheme, the stopping criterion and the criterion to decide when the Boltzmann-equilibrium has been obtained.

Algorithm 3: (Simulated Annealing)

Input: initial solution $J_0 \in \mathbf{J}$, initial temperature $T_0 \in \boldsymbol{R}^+$
neighborhood function $\mathcal{N} : \mathbf{J} \mapsto 2^{\mathbf{J}}$,
objective function $h : \mathbf{J} \mapsto \boldsymbol{R}$
a stopping criterion $\iota : \mathbf{J} \mapsto \{\mathbf{true},\mathbf{false}\}$
Output: Current solution $J \in \mathbf{J}$

```
T ← T_0
J ← J_0
while not (ι(J) ≠ true) do
    while not (equilibrium) do
        chose J' ∈ N(J)
        if (h(J') − h(J) ≤ 0) or (e^((h(J)−h(J'))/T) > random[0,1])
            J ← J'
        endif
    od
    T ← cooling_scheme(T)
od
return J
```

This algorithm allows "steps back". In the early stages when the temperature is high a worse objective value is more likely to be accepted and becomes more and more unlikely as the temperature decreases.

2.3.4 Evolutionary Algorithms

Evolutionary Algorithms (EAs) are inspired by the natural process of evolution and make use of the same terminology. The peculiarity of EAs is the maintenance of a set of points (called *population*) that are searched in parallel. Each point (*individual*) is evaluated according to the objective function (*fitness function*). Furthermore, a set of genetic operators is given that work on populations. They contribute to the the two basic principles in evolution *selection* and *variation*. Selection focuses the search to a “better” region of the search space by giving individuals with “better” fitness values a higher probability to be member of the next *generation* (loop iteration). On the other hand, variation creates new points in the search space. Here not only random changes of a particular point are possible (*mutations* that correspond to the neighborhood function of Simulated Annealing) but also the random mixing of the information of two or more individuals (*crossover* or *recombination*). A basic Evolutionary Algorithm is depicted by Algorithm 4.

The population is initialized with a random sample of the search space. Then the generational loop is entered. First, the fitness values are calculated using the objective function h and stored in the vector $\vec{\phi} = (\phi_1, \ldots, \phi_N)$. Next selection is performed using the current population $\vec{J}(k)$ and the current fitness vector $\vec{\phi}(k)$. Finally new points are created from this population using a variation-incorporating process and form the population of the next generation.

EAs are often characterized as combining features from *path-oriented* methods and *volume-oriented* methods [Bäck, 1994a]. EAs combine these contrary features in so far that in the beginning of the search the population is usually spread out in the search space, corresponding to a volume-oriented search. In later stages of the search the algorithm has focussed to few (or single) region due to selection and this single region is examined further. In this respect the algorithm behaves like a path-oriented search.

An other possible identification of these two stages of the search could be the correspondence of the first stage to a global reliability strategy and the second stage to a local refinement strategy.

A slightly modified version of HC and SA have been compared to an Evolutionary Algorithm (in particular Genetic Programming) on several program induction problems (similar to Example 2.1) in [O'Reilly, 1995]. The results showed a similar performance of the HC and EA approach. However, Simulated Annealing performed clearly better than HC and EA. Many more comparisons have been made between Simulated Annealing and Evolutionary Algorithms. A single bibliography lists 36 references of comparisons [Alander, 1995b].

Algorithm 4: (Basic Evolutionary Algorithm)

Input: Fitness function $f : \mathbf{J} \mapsto \boldsymbol{R}$, population size $N \in \boldsymbol{N}$,
selection operator, variation operator,
stopping criterion $\iota : \mathbf{J}^N \mapsto \{\mathbf{true}, \mathbf{false}\}$
Output: The best individual of the population
fulfilling the stopping criterion

$k \leftarrow 0$
$\vec{J}(0) \leftarrow$ initial_population(N)
while $(\iota(\vec{J}(k)) \neq \mathbf{true})$ **do**
  for $i \leftarrow 1$ **to** N **do**
    $\phi_i(k) \leftarrow f(J_i(k))$
  od
  $\vec{J'}(k) \leftarrow$selection$(\vec{J}(k), \vec{\phi})$
  $\vec{J}(k+1) \leftarrow$variation$(\vec{J'}(k))$
  $k \leftarrow k+1$
od
return best individual of $\vec{J}(k)$

2.4 Summary

In this chapter the probabilistic optimization method *Evolutionary Algorithms* was embedded in the framework of global optimization. A definition of the global optimization problem was given and several aspects of the problem have been discussed. In particular, the characteristics and desired properties of optimization algorithms have been addressed. Furthermore, some popular global optimization algorithms for the unconstrained optimization problem have been reviewed.

Part I

Theory of Evolutionary Algorithms for Optimization

Chapter 3

Evolutionary Algorithms for Optimization

Part I of the thesis will address Evolutionary Algorithms (EAs) from a theoretical point of view. In the following section the basic principles of Evolutionary Algorithms are explained. The term "Evolutionary Algorithm" is relatively new and describes a set of different optimization methods having the same basic principle. Some approaches are briefly introduced in Section 3.2. Next EAs are examined as a global optimization tool. The remainder of Part I deals with a mathematical analysis of the selection operator (Chapter 4) and recombination operator (Chapter 5).

3.1 Basic Principles of Evolutionary Algorithms

The basic concepts of EAs have briefly been mentioned in the previous chapter. Now a more formal description is given that lays the ground for a mathematical analysis of these algorithms.

Evolutionary Algorithms can be classified as probabilistic search algorithms characterized by the fact that a number N of potential solutions (called *individuals* $J_i \in \mathbf{J}$) of the optimization problem simultaneously sample the search space . Here for simplicity it is assumed that each individual directly represents a point in the search space. In practice often a coding and decoding process is necessary to map the search space with the representation space. The *population* $\vec{J} = (J_1, J_2, ..., J_N)$ is modified according to the natural evolutionary process: after initialization, selection $\omega : \mathbf{J}^N \times \boldsymbol{R}^N \mapsto \mathbf{J}^N$, recombination $\Xi_r : \mathbf{J}^N \mapsto \mathbf{J}^N$ and mutation $\Xi_m : \mathbf{J}^N \mapsto \mathbf{J}^N$ are executed in a loop until some termination criterion $\iota : \mathbf{J}^N \mapsto \{\mathbf{true}, \mathbf{false}\}$ is reached. Each run of the loop is called a *generation* and $\vec{J}(k)$ denotes the population at generation k.

The selection operator is intended to improve the average quality of the population by giving individuals of higher quality a higher probability to be copied into the next generation. Selection thereby focuses the search on promising regions in the search space. The individuals themselves are not modified, only copies (clones) of the individuals are created. The quality of an individual is measured by a fitness function $f : \mathbf{J} \mapsto \boldsymbol{R}$. In the most general form the fitness function depends on the objective function h, the constraints g_i, and the point in the search space. In the simplest case the fitness function might relate directly to the objective function ($f = h$). However if, for example, constraints have to be considered the fitness function may become more complicated. This is discussed in detail in Section 3.3. In general, the fitness function can be an algorithm that computes the fitness value.

In order to explore new points in the search space the "genetic material" in the population is changed either by recombination (also called crossover) or by mutation. The detailed working mechanism of these variation methods depends on the representation of the individual. To each operator a probability is associated that determines the application of this operator, i.e., p_c gives the probability of applying crossover to an individual and p_m gives the probability of applying mutation.

Algorithm 5: (Evolutionary Algorithm)

Input: Fitness function $f : \mathbf{J} \mapsto \boldsymbol{R}$,
population size $N \in \boldsymbol{N}$,
selection operator $\omega : \mathbf{J}^N \times \boldsymbol{R}^N \mapsto \mathbf{J}^N$,
crossover operator $\Xi_c : \mathbf{J}^N \mapsto \mathbf{J}^N$,
crossover probability $p_c \in [0, 1]$,
mutation operator $\Xi_m : \mathbf{J}^N \mapsto \mathbf{J}^N$,
mutation probability $p_m \in [0, 1]$
stopping criterion $\iota : \mathbf{J}^N \mapsto \{\mathbf{true}, \mathbf{false}\}$
Output: The best individual of the population fulfilling the stopping criterion

$k = 0$
$\vec{J}(0) \leftarrow$ initial_population(N)
while $(\iota(\vec{J}(k)) \neq \mathbf{true})$ **do**
  for $i \leftarrow 1$ **to** N **do**
    $\phi_i(k) \leftarrow f(J_i(k))$
  od
  $\vec{J}'(k) \leftarrow \omega(\vec{J}(k), \vec{\phi}(k))$
  $\vec{J}''(k) \leftarrow \Xi_c(\vec{J}'(k), p_c)$
  $\vec{J}(k+1) \leftarrow \Xi_m(\vec{J}''(k), p_m)$
  $k \leftarrow k + 1$
od
return best individual of $\vec{J}(k)$

In this thesis it is assumed that selection and variation are carried out sequentially: first a selection phase creates an intermediate population $\vec{J}'(k)$, next the crossover operator is applied yielding $\vec{J}''(k)$, and finally the mutation operator gives the population for the next generation.

This kind of description differs from other formulations where selection is made to obtain the individuals for recombination [Goldberg, 1989; Koza, 1992]. But it is mathematically equivalent and allows the selection method and recombination method to be analyzed independently. In his dissertation Bäck gives a similar but more detailed outline of the Evolutionary Algorithm [Bäck, 1994a].

3.1.1 Selection Mechanism

The basic idea of selection is to prefer "better" individuals to "worse" individuals, where "better" and "worse" are defined by the fitness function f. As only copies of existing individuals are created more individuals will be located at "good" positions in the search space. This is why selection is often compared with "exploitation", as known regions in the search space will be examined further. The assumption thereby is, that better individuals are more likely to produce better offspring, i.e., that there is a correlation between parental fitness and offspring fitness. In population genetics this correlation is named *heritability*. Without heritability, selection of better individuals makes no sense.

A nice feature of the selection mechanism is its independence of the representation of the individual, as only the fitness values of the individuals are taken into account. This simplifies the analysis of the selection methods and allows a comparison that can be used in all flavors of Evolutionary Algorithms.

Many selection mechanisms are known in Evolutionary Algorithms. Partly they are introduced according to a mathematical analysis (e.g. fitness-proportional selection), partly based on similarities in nature (e.g. tournament selection). The principle of selection is very general and a different methodologies of selection schemes can be observed:

- Generational and steady-state selection: The Evolutionary Algorithm outlined by Algorithm 5 describes a generational concept. First selection acts on the whole population, then the variation operators are applied to the whole population. However, the steady-state approach replaces only a few members in the population by applying selection and recombination. For example, two individuals are selected by a selection mechanism, then crossover is applied to these two individuals and finally the offspring is inserted back into the population [Syswerda, 1989; Whitley, 1989].

- Elitist and non-elitist selection: standard generational selection schemes do not guarantee that the current best individual will be contained in the next generation (non-elitist selection). This may happen either due to the probabilistic nature of a selection scheme or due to the fact that all best individuals are "destroyed" by recombination or mutation. Consequently elitist selection schemes have been proposed. They simply copy the best individual of the current generation unconditionally into the next generation. This idea was originally suggested in [Jong, 1975].

- Sharing or crowding: Simple selection schemes are designed to concentrate on *one* optimum. In several cases it may be desirable to obtain several (equally good) optima. In particular, if one is interested in the optimization of multimodal functions, one might want to end up with a population distributed over all (or many) of the optima. This is usually achieved by replacing a new individual by the most "similar" individual out of a randomly chosen *crowd* of the population. The "similarity" introduces a new selection criterion and makes a metric necessary to define similarity. For example, in the case of bit-string representation this can be the Hamming-distance. Crowding has been proposed by [Jong, 1975]. A related method was introduced for tournament selection in [Harik, 1995].

In fitness-*sharing* the metric is used to force individuals in a population to share their fitness assignments with neighboring individuals. The fitness value of an individual is reduced if other individuals exist in its neighborhood (according to the metric). A problem with this method is that the calculation of the reduced fitness values introduces two or more additional parameters that need to be adjusted. Furthermore, the number of optima of the objective function has to be known in advance.

Comparisons of several multi-modal search techniques for Evolutionary Algorithms can be found in [Mahfoud, 1995; Harik, 1994]. It should be mentioned that multi-modal techniques may be useful even if unimodal functions are optimized. This is due to the maintaining of diversity in population.

A large fraction of this first part will be devoted to the derivation of a framework to analyze generational, unimodal, and non-elitist selection schemes and the application of this framework to some popular selection schemes (Chapter 4). In particular, tournament selection, truncation selection, linear and exponential ranking selection, and proportionate selection will be considered.

3.1.2 Crossover and Mutation Mechanism

Variation is introduced into the population by crossover or mutation. As these operators usually create offspring at new positions in the search space, they are also called "explorative" operators. Algorithm 5 only gives an outline of the principle of an Evolutionary Algorithm. The several instances of the EA differ in the way individuals are represented and in the realization of the recombination operator. Common representations are, for example, bit strings, vectors of real or integer values (for parameter optimization), trees (for function optimization), graphs or any problem-dependent data-structure. Based on information-theoretical considerations Holland suggests that the bit-string representation is optimal [Holland, 1975]. But for practical issues application-specific representations have revealed many advantages. These so-called hybrid approaches are for example promoted in [Michalewicz, 1992; Davis, 1991].

Along with a particular data-structure variation operators have to be defined that can be divided in asexual and sexual variation operators. The asexual variation (*mutation*) consists in a random change of the information represented by an individual.

Example 3.1 *If the individual is represented as a vector, mutation is the random change of an element of the vector (Fig. 3.1). How this change is performed depends on the type of the vector-elements. If the vector is a simple bit-string, mutation is to toggle the bit or not (with equal probability). For real values or integer values more sophisticated mutation operators are necessary. The most general approach is to define a probability distribution over the domain of possible values for a particular vector element. A new value is then chosen according to this distribution.*

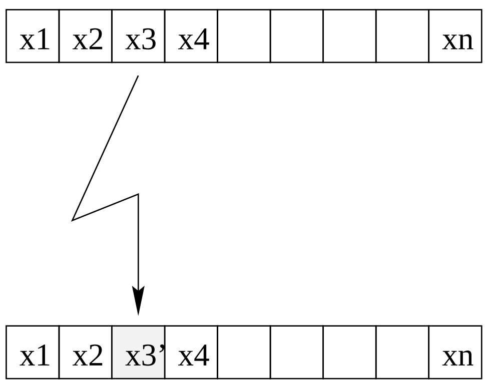


Figure 3.1: *Mutation operator for vector-representation - a randomly chosen element of the vector is changed.*

Example 3.2 *The mutation operator for tree representations works as follows: A randomly chosen mutation site (an edge in the tree) is* *selected and the subtree attached to this edge is replaced by a new, randomly created tree (see Fig. 3.2). Usually some parameters are additionally defined that restrict the maximum size or depth of a tree.*

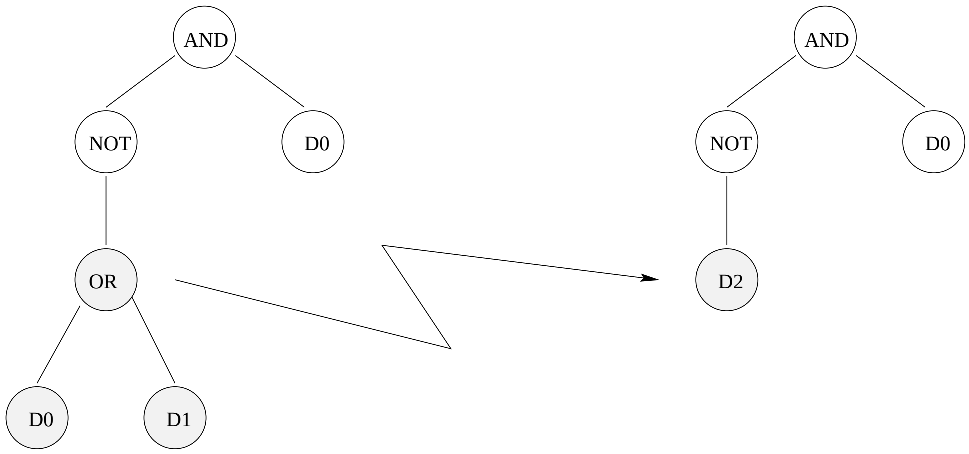


Figure 3.2: *Mutation operator for tree-representation - a randomly chosen subtree is replaced by a randomly created new tree (in this example only a single leaf).*

During sexual variation (*crossover* or *recombination*) two individuals exchange part of their information. Two individuals are chosen from the population and named *parents*. How the exchange of information is performed depends on the chosen representation. Again the method will be elucidated using two examples, the vector representation and the tree representation.

Example 3.3 *The simplest crossover operator for vector representations is shown in Fig. 3.3. In one-point crossover, a position in the vector is randomly chosen and the remaining positions of the vectors are simply swapped. This is the operator originally proposed by Holland [*Holland, 1975*] for Genetic Algorithms , where the vector elements consist of bits. The idea can be easily extended to two-point or n-point (uniform) crossover by choosing more than one swap-position [*Syswerda, 1989*].*

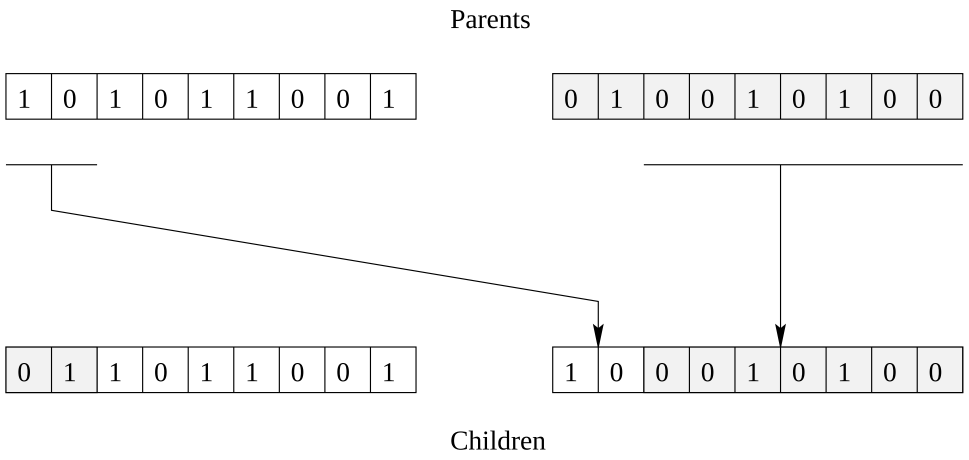


Figure 3.3: *One-point crossover for vector representations.*

Example 3.4 *The crossover operator for tree representations reproduces two offspring from two parents in the following way: In each tree an edge is randomly chosen as* crossover site *and the subtree attached to this edge is cut from the tree. The two subtrees are swapped and combined with the old trees at the crossover sites to form the children (Fig 3.4). In general, this results in two new trees, even if the two parents are identical.*

There is no need to restrict the number of parents for crossover to two. Recent research shows that increasing the number of mates leads to an increased performance [Syswerda, 1993; Moran *et al.*, 1995; van Kemenade *et al.*, 1995]. The special case of using the whole population as parents is also interesting from a theoretical point of view as the equations describing crossover simplify significantly [Mühlenbein and Voigt, 1995; Beyer, 1995b].

There is an ongoing debate between different communities which operator - mutation or crossover - is more important. Some researchers found evidence that the crossover operator might be "simulated" by mutation [Beyer, 1995b].

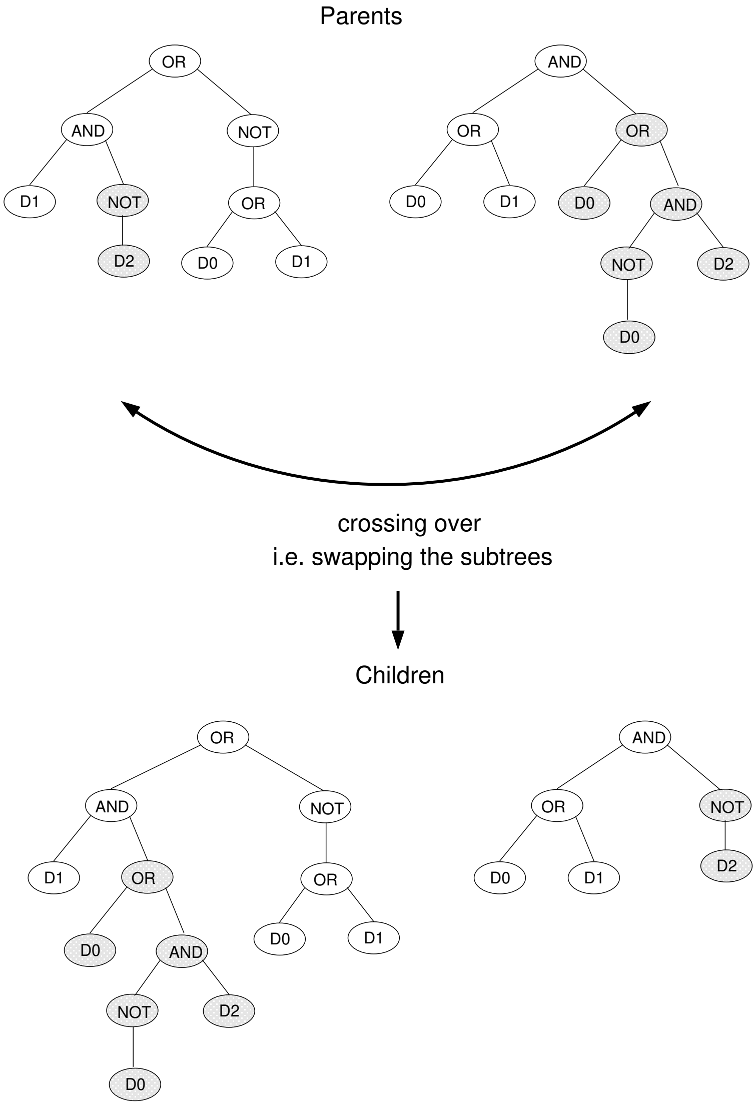


Figure 3.4: *Crossover operator for tree representations: two randomly chosen subtrees are exchanged to form the offspring.*

3.2 Variants of Evolutionary Algorithms

The variety of data-structures, selection and recombination mechanisms give possible ways of classifying Evolutionary Algorithms. However, the different terms are mostly historical. Moreover, the differences between the variants are fluid. Here the intention is to give a brief overview of the mainstream in EA research. A more detailed discussion of the similarities and differences of the flavors of EA can be found for example in [Bäck, 1994a; Nissen, 1994; Bäck and Schwefel, 1993].

3.2.1 Genetic Algorithms

Genetic Algorithms have been introduced by Holland in 1970s [Holland, 1975]. He used a population based algorithm to evolve rules for a classifier system. The "original" Genetic Algorithm uses *fitness-proportionate selection*, *bitstrings* of a fixed length l to represent the individuals (i.e., $\mathbf{J} = \{0,1\}^l$) and *one-point crossover*.

The design of the algorithm was based on a mathematical theory developed by Holland. The most famous result of this theory is the so-called *Schema Theorem* [Holland, 1975], also stated to be the "Fundamental Theorem of Genetic Algorithms" [Goldberg, 1989]. From this theorem "design rules" have been derived like the *builing-block- hypothesis (BBH)*. It states that a GA works by combining small, good parts of a solution (the building-blocks) to larger parts by the crossover-operator. As the crossover-operator may disrupt such building-blocks if they are spread across the gene, another design rule suggest to code information that belong together next to each other in the string. Another result from this theory concerns the selection method in GAs. The fitness-proportionate selection was designed to have optimal trade-off between exploration and exploitation.

Several researchers have drawn attention to weak points of the

Schema Theorem [Mühlenbein, 1991; Altenberg, 1995; Vose, 1991; Beyer, 1995a; Grefenstette and Baker, 1989]. Mühlenbein points out

> "that the question of why genetic algorithm builds better and better substrings by crossing-over is ignored" [Mühlenbein, 1991].

Beyer claims

> "if the BBH is regarded as the *reason* why the GA works, then *cause and effect are reversed*. Furthermore, the observation that leads to the BBH is *not* a feature *exclusively observed* in GAs" [Beyer, 1995a].

Altenberg summarizes the criticism of the Schema Theorem as

> "the mistake is to conclude that the growth of schemata has any implications for the quality of the search carried out by GA. [...] The common interpretation of the Schema Theorem implicitly assumes that any member of an above-average schema is likely to produce offspring of above-average fitness, i.e., that there is a correlation between membership in an above-average schema and production of fitter offspring. But the existence of such a correlation is logically independent of the validity of the Schema Theorem"[Altenberg, 1995].

Another important objection is the fact that the Schema Theorem predicts a very low increase of above average schema if uniform crossover [Syswerda, 1989] is used, as the disruption in the strings is very high. But

> "testing shows, however, that uniform crossover performs significantly better than either one or two-point crossover on most problems" [Syswerda, 1989].

Also several objections were raised that proportional-selection is an unsuited selection operator for an optimization method. This point will be discussed in detail in Section 4.2.

De Jong argues in two identical articles that Genetic Algorithms are not designed to be function optimizers but as a robust adaptive system [Jong, 1992; Jong, 1993]. Modifications such as replacing proportional selection with a rank-based selection scheme are necessary if a GA is to used for function optimization. This is not due to any insufficiency of the Schema Theorem.

Regardless of this dispute many successful practical applications of Genetic Algorithms have been published (see, e.g. [Bäck *et al.*, 1993; Alander, 1995a])

3.2.2 Genetic Programming

Genetic Programming can be viewed as a Genetic Algorithm using trees to represent individuals. This enables special application fields to be addressed, e.g. function optimization where the shape of a function is evolved, not only the constants. Another application area is the optimization of programs. As programs can be represented as trees this allows the "automatic programming of computers". The original idea was first proposed by [Cramer, 1985] but Koza elaborated it and made it popular [Koza, 1992]. Since then a continuously increasing community activity devoted to GP has evolved. It has recently reached another peak by a dedicated annual conference on GP.

The first approaches of GP used proportionate selection, crossover, and no mutation. Currently, the preferred selection scheme is tournament selection, that has been proposed being superior by some researchers [Blickle and Thiele, 1994] and been found empirically to be superior by others [Koza, 1994b].

Although the tree-based representation has many advantages, the need for hierarchy in the trees has also been identified. Hierarchy allows the GP to decompose a task in several subtasks and to use these subtasks several times in a program. The most successful approach has been given by Koza, the so-called automatically defined functions (ADFs). Whereas it was necessary in the initial approaches to define the number of sub-functions and their arguments in advance, new ideas make these requirements obsolete. The adaption of the number of subroutines and their parameters is achieved by so-called *architecture-altering operations* [Koza, 1994a].

A well-known problem due to the variable length of the tree-representation is the so-called *bloating phenomenon*. It describes the fact that the size of tree continues to grow despite the quality of a solution (individual) not improving. This problem will be discussed in detail in Section 5.2.

3.2.3 Evolution Strategies

Evolution Strategies (ES) were developed in the 1960s at the Technical University of Berlin by Rechenberg and Schwefel ([Schwefel, 1975; Rechenberg, 1973]). From the beginning ES have been designed as a numerical function optimization method. Both real and integer optimization is used but most commonly each individual is represented by an l-dimensional vector $J \in \boldsymbol{R}^l$. The selection method and population concept is described by two variables μ and λ. μ gives the number of parents (corresponding to the population size) whilst λ describes the number of offspring produced in each generation. The selection scheme is deterministic and the notation $(\mu + \lambda)$-ES describes an algorithm where parents and offspring together compete to reach the next generation. In other words, only the μ best individuals from the $\mu + \lambda$ individuals are selected for the next generation. The notation (μ, λ)-ES, however, describes a selection scheme where only the μ best individuals of the λ offspring are transferred to the next generation. Obviously, this requires $\mu \leq \lambda$. However, the case $\mu = \lambda$ leads to a random walk behaviour (no selection), hence $\mu < \lambda$ is required. The mutation of a vector element J^i is performed by adding a randomly chosen value according to a normal distribution with zero mean and standard deviation σ^i. A peculiarity of ES is the *self-adaptation* of the standard deviations σ^i, i.e., the standard deviations are also optimized according to the actual topology of the objective function. [Schwefel, 1975] showed that crossover is necessary to achieve this self-adaptation.

3.2.4 Evolutionary Programming

Evolutionary Programming (EP) is in many aspects similar to Evolution Strategies, elaborated independently by L.J. Fogel in the early 1960s. The current state-of-the-art in EP started with its application to optimization problems and is termed *meta-EP* [Fogel, 1991; Fogel, 1992]. EP exclusively relies on mutation as variation operator; no crossover is used at all. The selection mechanism is a mixture of tournament selection and truncation selection. Meta-EP incorporates the variances of the mutation rates in the genotype. Hence, self-adaptation is possible, similar to ES.

3.3 Constraint-Handling

If Evolutionary Algorithms are to be used for global optimization, a particular question arising is the handling of the constraints g_i. As all information of the quality of a solution is condensed to a single value (the fitness value) the information of constraint violation and the objective function h must be merged into the fitness value f. Up to now, no universal method for constraint handling has been developed for Evolutionary Algorithms. The main approaches proposed so far are summarized in the following. A comparison of several approaches can also be found in [Michalewicz, 1995b; Michalewicz, 1995a].

- **Constraint-handling by rejecting infeasible individuals:** Whenever an infeasible individual is obtained, the creation process of this individual is repeated, until a feasible individual is found. Unfortunately, this is a very inefficient method if the problem is highly constrained. In this case, most time is spend in (randomly) creating and rejecting individuals. Nevertheless, this method is used in Evolution Strategies, but here self-adaptation can reduce the amount of time spend in creating infeasible offspring.

- **Constraint-handling based on intelligent coding:**
A straight-forward idea to take care of constraints is to use a coding of the individual that allows only feasible solutions to be represented. But this is more a "general rule" how to code individuals and such a representation might not exist. Furthermore, most numerical optimization problems can not be handled with this approach.

- **Constraint-handling by repair:**
In this approach infeasible solutions are transformed into feasible ones with a special repair algorithm. However, this repair algorithm is problem specific and restoring the feasibility may be as difficult as the optimization problem. Nevertheless, this approach can be very helpful if a good repair heuristic is available.

- **Constraint-handling based on penalty functions:**
The most widely used approach to constraint-handing in Evolutionary Algorithms is the use of penalty functions. Some representatives are for example [Joines and Houck, 1994; Michalewicz and Attia, 1994; Powell and Skolnick, 1993]. In this case the fitness function is a sum of the objective function h and a penalty function p that reflects the violation of the constraints, i.e., (using the notation of Def. 2.1)

$$f(J) = \begin{cases} h(J) & : J \in \tilde{\mathbf{J}} \\ h(J) + p(g_1, g_2, \ldots, g_q, J) & : J \in \mathbf{J} \setminus \tilde{\mathbf{J}} \wedge h(J) \text{exists} \\ p(g_1, g_2, \ldots, g_q, J) & : J \in \mathbf{J} \setminus \tilde{\mathbf{J}} \\ & \wedge h(J) \text{does not exist} \end{cases} \tag{3.1}$$

The penalty is zero, if the individual represents a point $J \in \tilde{\mathbf{J}}$, i.e., J lies in the feasible region where no constraint is violated.

If one or more constraints are violated a factor p is added to the objective value leading to a "penalized" fitness value. If the objective value is not computable at that point in the search space the fitness value consists only of the penalty term.

By this the constrained optimization problem is transformed into an unconstrained one with a different objective function $h'(J) = f(J)$. A problem exists in determining the strength of the penalty. If one imposes a high degree of penalty more emphasis is placed on obtaining feasibility and also the EA will move quickly to the feasible region it is likely to converge to a point far from the optimum. However, if a low degree of penalty is used the EA may never reach the feasible region [Joines and Houck, 1994].

[Richardson *et al.*, 1989] gave the first detailed study on the design of penalty functions. They examined the set covering problem and other constrained numerical optimization tasks with different penalty functions. They hypothesized from their experiments that:

> "penalties which are functions of the distance from the feasibility region are better performers than those which are merely functions of the number of violated constraints"[Richardson *et al.*, 1989].

Furthermore, they propose penalty terms based on "expected completion costs", i.e., the expected cost to transform the infeasible solution into a feasible one. However, as discussed above this cost measure might not exist or not be computational tractable. They also ensure in their experiments that all infeasible solutions have a worse fitness value than any feasible one. This is the basic idea used in [Powell and Skolnick, 1993], too. Furthermore, Powell and Skolnick propose the use of rank-based selection schemes as they avoid scaling problems with the penalty functions.

[Michalewicz, 1995b; Michalewicz and Attia, 1994] introduce an algorithm that starts with a single (potentially randomly chosen) point $\hat{J}$. A new point $\hat{J}'$ is obtained out of $\hat{J}$ using an

Evolutionary Algorithm with a dynamic penalty function of all violated constraints at point $\hat{J}$, i.e.,

$$p(g_1, \ldots, g_q, J) = \frac{1}{2m} \sum_{l=1,\ldots,q: g_l(\hat{J})<0 \wedge g_l(J)<0} g_l^2(J),$$

where m determines the degree of penalty. m is called "temperature" and adjusted according to some "cooling scheme".

[Joines and Houck, 1994] also make use of dynamic penalty functions. As penalty term the number of violated constraints is used, weighted by a factor p_k that increases linearly with the generation count k of the EA. In this approach, all constraints may be active. The penalty function can be stated as

$$p(g_1, \ldots, g_q, J) = p_k \sum_{l=1,\ldots,q: g_l(J)<0} |g_l(J)|$$

- **Other methods of constraint-handling:**
  A special system for linear-constrained problems has been proposed through the Genocop system [Michalewicz and Janikow, 1991]. Due to the restriction to linear (equation and inequality) constraints the feasible region is always convex. This fact is used to construct special genetic operators that guarantee that the offspring remain in the feasible region. Hence, no penalty functions are necessary. In [Janikow, 1995] this method is used to optimize fuzzy decision trees.

  [Schoenauer and Xanthakis, 1993] use some pre-calculation to obtain an initial population for an Evolutionary Algorithm that consists almost only of individuals of the feasible region. This is obtained by evolving an initial random population with an objective function related to constraint satisfaction. If a reasonable percentage of the population fulfills the constraints, the objective function is switched to the original objective function h with a rigorous penalty applied whenever an individual does not satisfy a constraint. To make this mechanism work, sharing methods for the selection scheme are necessary in order to avoid premature convergence.

- **Individual objective switching (IOS):**
  Reflecting the previous methods on constraint-handling a combination of the penalty-based approach and the objective function switching used by [Schoenauer and Xanthakis, 1993] can be created. This new approach, called individual objective switching (IOS), defines a constant fitness function for the whole optimization process as used in the penalty-based approach. However, the fitness function is not a sum of penalty terms but a stepwise defined function, each step corresponding to a constraint violation. First, the objective is to minimize the degree of violation of constraint g_1, i.e., to minimize $-g_1(J)$, if $g_1(J) < 0$ (see Definition 2.1). If the individual fulfills this constraint, the objective function is switched to minimize the violation of g_2. This process continuous until all constraints are fulfilled and the original objective function is active. Note, that the fitness function changes individually for each individual and not simultaneously for the whole population. To guarantee that an individual violating less constraints has a better fitness value than an individual violating more constraints an appropriate transformation is necessary. For example, the arctan function can be used, to scale the range for each constraint violation into $[0, 1[$. The adding of a certain offset leads to an appropriate fitness function (assuming fitness minimization):

$$f(J) = \left\{ \begin{array}{rcl} q - \frac{2}{\pi}\arctan g_1(J) & : & g_1(J) < 0 \\ q - 1 - \frac{2}{\pi}\arctan g_2(J) & : & g_1(J) \geq 0 \wedge g_2 < 0 \\ q - 2 - \frac{2}{\pi}\arctan g_3(J) & : & g_1(J), g_2(J) \geq 0 \wedge g_3 < 0 \\ \dots & : & \dots \\ 1 - \frac{2}{\pi}\arctan g_q(J) & : & g_1(J), \dots, g_{q-1}(J) \geq 0 \\ & : & \wedge g_q < 0 \\ \frac{2}{\pi}\arctan h(J) & : & g_1(J), \dots, g_q(J) \geq 0 \end{array} \right. \tag{3.2}$$

  If a scaling and translation invariant selection method (such as tournament selection) is used in the EA, the transformation of the objective function h in (3.2) has no impact on the optimization process.

The main drawback of the method is the fact that it needs a gradual function to specify the degree of constraint-violation. A typical example is to consider the distance of the current point from the feasible region (with respect to this particular constraint). However, it may be impossible to calculate such a distance if the constraint is a Boolean variable. For example, the constraint "J is a prime number" can only be expressed as a Boolean statement. In this case the Evolutionary Algorithm has to solve a "needle-in-the-haystack" problem. Another potential problem lies in the sequence the constraints are considered. The ordering might have a strong impact on the performance of the optimization.

The method has been successfully used to obtain a controller function for a real-world broom-balancing problem [Stebler, 1996]. It is also applied to the GP method evolving short and accurate solutions for a regression problem (Chapter 5.2) and experimentally compared with other approaches (Chapter 5.4). Furthermore, it is used in Part II of this thesis for solving the system-synthesis optimization problem (Chapter 9).

In summary, the most popular method to handle constraints is the introduction of penalty-terms. But if a problem is highly constrained this may lead to poor performance of the optimization method. Hence, one should at first try to repair infeasible solutions with some knowledge-based heuristic. This heuristic should balance the tradeoff between computational complexity and success. The remaining infeasible solutions that are not captured by the repair-heuristic are then reflected by penalty-terms.

3.4 Disadvantages of Evolutionary Algorithms

- *High computational demand:* The modest demands on the objective function is paid with a relatively high computational demand. This demand not only arises form the population concept but also from the difficulty of the problems. An application specific heuristic that makes use of domain-knowledge is likely to outperform an Evolutionary Algorithm.

- *Difficult adjustment of parameters:* This is from the author's point of view the main drawback of Evolutionary Algorithms. A large number of parameters need to be adjusted, for example the kind of selection and crossover operator to use, the population size, the probabilities of applying a certain operator, and the form of the fitness function. Due to this fact successful applications are often the result of a lengthy trial-and-error procedure whose purpose is to adjust the parameters of the Evolutionary Algorithm for the particular problem class or even problem instance. Furthermore, EAs are often very sensitive to the fitness function such that slight changes in the fitness function may lead to completely different behaviour.

- *Heuristic principle:* EAs do not guarantee to find the global optimum. The theoretical proofs of global convergence are useless from a practical point of view as they assume infinite computation time. Under this premise even random search can reach the global optimum. Of more importance is the fact that for most instances of EA the accuracy of a solution obtained in a limited amount of computation time can not be predicted or guaranteed.

3.5 Advantages of Evolutionary Algorithms

- *Large application domain:* Evolutionary Algorithms have been used successfully used for a wide range of applications [Bäck *et al.*, 1993; Alander, 1995a]. One reason for this might be the intuitive concept of evolution and the modesty of the EA with regard to the structure of the specific optimization problem.

Especially the intuitive concept makes it easy to implement an algorithm "that works".

- *Suitable for complex search spaces:* It is exceedingly difficult to construct heuristics for complex combinatorial problems. In these problems the choice of one variable may change the meaning or quality of an other, i.e., there are high correlations between variables (often also called *epistasis*). EAs have been successfully applied to solving epistatic problems albeit with more effort. Obviously, the success of the EA depends on the particular implementation and not all flavors of EAs are equally well suited. As a rule of thumb it is always good to combine an EA with available (problem-dependent) optimization heuristics.

- *Easy to parallelize:* The population concept of Evolutionary Algorithms makes parallelization easy. By this, the execution time of the algorithm can be reduced. In principle, one can distinguish two different philosophies of parallelization: *paradigm-preserving* and *paradigm-changing* parallelization.

  In paradigm-preserving parallelization only the computation of the fitness values is parallelized. The individuals are sent to a number of *slaves* that only perform the fitness calculation and return the result. This kind of parallelization is especially suited to a cluster of workstations where each processor can communicate with all other processors via a global network [Stanley and Mudge, 1995]. An implementation in YAGPLIC [Blickle, 1995] showed that the overhead is independent of the particular problem and a reasonable speedup can be achieved for the fitness computation [Frey and Vaterlaus, 1996].

  A paradigm-changing parallelization is the formation of subpopulations. A subpopulation is assigned to each processor which then evolves almost independently of the other populations. Furthermore, a topology of the population is defined such that each subpopulation has only a few "neighbors". A few individuals *migrate* between neighbors and form a loose coupling between the subpopulations. For concrete implementations several details have to be clarified:

- the number of individuals to exchange
- which individuals to exchange (the best ones or arbitrary ones)
- which individuals to replace (the worst ones or arbitrary ones)
- synchronous or asynchronous exchange of individuals
- exchange only between neighbors or between arbitrary subpopulations.

Many implementations exist that exploit this migration-model (e.g., [Gorges-Schleuter, 1989; Tanese, 1989; Tamaki and Nishikawa, 1992; Gordon *et al.*, 1992]). The reported superlinear speedup of these implementation is due to the modified paradigm.

- *Robustness:* Robustness means that different runs of an EA for the same problem give similar results, i.e., that there is no great deviation in the quality of the solution. This has been observed, for example, in [Rogenmoser *et al.*, 1996]. They considered the transistor size optimization of high-speed CMOS circuits and compared several stochastic optimization techniques. Although a Monte-Carlo-based optimizer performed in average as good as a Genetic Algorithm the variation in the results was much higher.

In summary, EAs can be recommended for constrained global optimization, if

- the problem is too complex to be handled by an exact method or no exact method is available for the problem,
- there exists no problem-specific heuristic,
- a problem-specific heuristic exists and can be included into the EA.

Chapter 4

Selection

The selection operator is one of the main operators of Evolutionary Algorithms. In the following a comparison of the most prominent generational, unimodal, and non-elitist selection schemes is carried out. A large part of the analysis is also applicable to steady-state selection schemes as the selection probabilities are the same as in the generational case.

4.1 Description of Selection Schemes

In this section a description of selection schemes is introduced that will be used in the subsequent sections to analyze and compare tournament selection, truncation selection, linear and exponential ranking selection and fitness proportional selection. The description is based on the fitness distribution of the population before and after selection as introduced in [Blickle and Thiele, 1995].

4.1.1 Related Work

Work has been done to classify the different selection schemes such as *proportionate selection, ranking selection, tournament selection.* As the focus of this work is on a theoretical investigation, empirical comparisons are neglected.

Goldberg [Goldberg and Deb, 1991] introduced the term *takeover time*. The takeover time is the number of generations that is needed for a single best individual to fill up the whole generation if no crossover or mutation is used. Hence, this analysis is restricted to the behaviour of the best individual. Bäck has analyzed the most prominent selection schemes used in Evolutionary Algorithms with respect to their takeover time [Bäck, 1994b; Bäck, 1994a].

In [Mühlenbein and Schlierkamp-Voosen, 1993] the *selection intensity* in the so called *Breeder Genetic Algorithm (BGA)* is used to measure the progress in the population. The selection intensity describes the change in the average fitness of the population due to selection. An exact definition of the selection intensity as used in this thesis will be given in Def. 4.10. In [Mühlenbein and Schlierkamp-Voosen, 1993] the selection intensity is derived for proportional selection and truncation selection. Here this work will be extended to tournament selection, linear and exponential ranking selection.

An analysis based on the behaviour of the best individual (as done by Goldberg and Bäck) or on the average population fitness (as done by Mühlenbein) only describes one aspect of a selection method. Herein a selection scheme is described by its interaction on the distribution of fitness values. Out of this description several properties can be derived, e.g. the behavior of the best or average individual.

The characterization of the population by its fitness distribution has also been used by other researchers, but in a more informal way than it will be done in the subsequent sections. In [Mühlenbein and Schlierkamp-Voosen, 1993] the fitness distribution is used to calculate some properties of truncation selection. In [Shapiro *et al.*, 1994] a statistical mechanics approach is taken to describe the dynamics of a Genetic Algorithm that also makes use of fitness distributions.

The consequent use of fitness distributions has many advantages:

- Fitness distributions are a powerful framework giving an unified view of the selection schemes and allowing several up to now independently and isolated obtained aspects of these selection schemes (such as selection intensity or takeover time) to be derived with one single methodology.
- Several new properties of selection schemes can be derived for the first time, e.g.
  - the expected fitness distribution of tournament selection (Theorem 4.5), truncation selection (Theorem 4.9), linear ranking selection (Theorem 4.13), and exponential ranking selection (Theorem 4.18).
  - a new important characteristic of selection schemes, called *loss of diversity* (Definition 4.8)
  - an important new property of tournament selection (the concatenation of tournament selection - Theorem 4.7).
  - new approximations for properties of tournament selection (selection intensity and the selection variance)
  - new properties of exponential ranking selection (selection intensity, selection variance and loss of diversity).
- The convergence of an Evolutionary Algorithm optimizing the ONEMAX function can be predicted for several selection schemes.
- The unified description enables a new kind of comparison of selection schemes. In Chapter 4.7 for the first time several properties can be compared simultaneously.

These results are presented in the next sections. First, all necessary definitions are introduced.

4.1.2 Fitness Distribution

For selection only the fitness values of the individuals are taken into account. Hence, the state of the population is completely described by the fitness values of all individuals. Note that the objective function h in Definition 2.1 is to be minimized but from convention the fitness value in Evolutionary Algorithms is to be maximized. Hence, in the following analysis larger fitness values are "better" fitness values and correspond to smaller objective values.

There exists only a finite number of different fitness values $f_1, \ldots, f_n$, $(n \leq N)$ and the state of the population can as well be described by the values $s(f_i)$ that represent the number of occurrences of the fitness value f_i in the population.

Definition 4.1 (Fitness distribution) *The function $s : \boldsymbol{R} \mapsto Z_0^+$ assigns to each fitness value $f \in \boldsymbol{R}$ the number of individuals in a population $\vec{J} \in \mathbf{J}^N$ carrying this fitness value. s is called the* fitness distribution *of a population $\vec{J}$.*

It is possible to describe a selection method as a function that transforms a fitness distribution into another fitness distribution.

Definition 4.2 (Selection method) *A* selection method *Ω is a function that transforms a fitness distribution s into an new fitness distribution s':*

$$s' = \Omega(s, par\_list) \tag{4.1}$$

par_list is an optional parameter list of the selection method.

As the selection methods are probabilistic the next definition is introduced.

Definition 4.3 *$\Omega^*(s, par\_list)$ denotes the* expected fitness distribution *after applying the selection method Ω to the fitness distribution s. The notation $s^* = \Omega^*(s, par\_list)$ will be used as abbreviation.*

It is interesting to note that it is also possible to calculate the variance of the resulting distribution.

Theorem 4.1 *The variance in obtaining the fitness distribution s' is*

$$\sigma_s^2 = s^* \left(1 - \frac{s^*}{N}\right) \tag{4.2}$$

Proof: $s^*(f_i)$ denotes the expected number of individuals with fitness value f_i after selection. It is obtained by doing N experiments "select an individual from the population using a certain selection mechanism". Hence the selection probability of an individual with fitness value f_i is given by $p_i = \frac{s^*(f_i)}{N}$. To each fitness value there exists a Bernoulli trial "an individual with fitness f_i is selected". As the variance of a Bernoulli experiment with N trials is given by $\sigma^2 = Np(1-p)$, (4.2) is obtained using $p = p_i$. □

The index s in σ_s stands for "sampling" as it is the mean variance due to the sampling of the finite population. The variance of (4.2) is obtained by performing the selection method in N independent experiments. It is possible to reduce the variance almost completely by using more sophisticated sampling algorithms to select the individuals. Baker's "stochastic universal sampling" algorithm (SUS) [Baker, 1987], which is an optimal sampling algorithm, is introduced in Section 4.7 when the different selection schemes are compared.

Definition 4.4 (Cumulative fitness distribution) *Let n be the number of unique fitness values and $f_1 < ... < f_{n-1} < f_n$ $(n \leq N)$ the ordering of the fitness values with f_1 denoting the worst fitness occurring in the population and f_n denoting the best fitness in the population.*

$S(f_i)$ denotes the number of individuals with fitness value f_i or worse and is called cumulative fitness distribution, *i.e.,*

$$S(f_i) = \begin{cases} 0 & : \quad i < 1 \\ \sum_{j=1}^{j=i} s(f_j) & : \quad 1 \leq i \leq n \\ N & : \quad i > n \end{cases} \tag{4.3}$$

Example 4.1 *As an example of a discrete fitness distribution the initial fitness distribution of the 6-multiplexer problem is used (see also Example 2.1 on page 13). Worse individuals have lower fitness values. Figure 4.1 shows the distribution $s(f)$.*

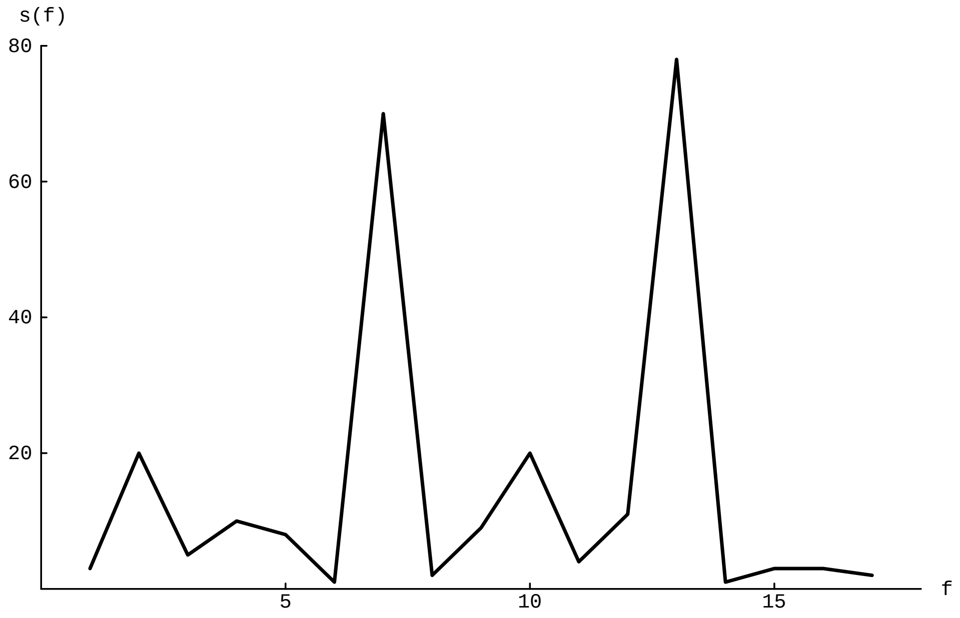


Figure 4.1: *The fitness distribution $s(f)$ for the 6-multiplexer problem.*

Next, the distribution $s(f)$ will be described as a continuous distribution $\bar{s}(f)$ allowing the following properties to be easily derived. The range of the function $\bar{s}(f)$ is $f_0 < f \leq f_n$, using the same notation as in the discrete case.

All functions in the continuous case will be denoted with a bar, e.g. $\bar{s}(f)$ instead of $s(f)$. Sums are replaced by integrals, for example

$$\bar{S}(f) = \int_{f_0}^{f} \bar{s}(x)\, dx \tag{4.4}$$

denotes the continuous cumulative fitness distribution.

Example 4.2 *As an example for a continuous fitness distribution the Gaussian distribution $\mathcal{G}(\mu, \sigma)$ with*

$$\mathcal{G}(\mu, \sigma)(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{4.5}$$

is chosen.

Now the aspects of the fitness distribution that will be compared are introduced. The definitions given all refer to continuous distributed fitness values.

4.1.3 Average Fitness

Definition 4.5 (Average fitness) $\bar{M}$ *denotes the average fitness of the population before selection and* $\bar{M}^*$ *denotes the expected average fitness after selection:*

$$\bar{M} = \frac{1}{N} \int_{f_0}^{f_n} \bar{s}(f)\, f\, df \tag{4.6}$$

$$\bar{M}^* = \frac{1}{N} \int_{f_0}^{f_n} \bar{s}^*(f)\, f\, df \tag{4.7}$$

4.1.4 Fitness Variance

Definition 4.6 (Fitness variance) *The* fitness variance $\bar{\sigma}^2$ *denotes the variance of the fitness distribution* $\bar{s}(f)$ *before selection and* $(\bar{\sigma}^*)^2$ *denotes the variance of the fitness distribution* $\bar{s}^*(f)$ *after selection:*

$$\bar{\sigma}^2 = \frac{1}{N} \int_{f_0}^{f_n} \bar{s}(f)\, (f - \bar{M})^2\, df = \frac{1}{N} \int_{f_0}^{f_n} f^2 \bar{s}(f)\, df - \bar{M}^2 \tag{4.8}$$

$$(\bar{\sigma}^*)^2 = \frac{1}{N} \int_{f_0}^{f_n} \bar{s}^*(f)\, (f - \bar{M}^*)^2\, df = \frac{1}{N} \int_{f_0}^{f_n} f^2 \bar{s}^*(f)\, df - \bar{M}^{*2} \tag{4.9}$$

Note the difference of this variance to the variance in obtaining a certain fitness distribution characterized by Theorem 4.1

4.1.5 Reproduction Rate

Definition 4.7 (Reproduction rate) *The* reproduction rate $\bar{R}(f)$ *denotes the ratio of the number of individuals with a certain fitness value f after and before selection*

$$\bar{R}(f) = \begin{cases} \frac{\bar{s}^*(f)}{\bar{s}(f)} & : \quad \bar{s}(f) > 0 \\ 0 & : \quad \bar{s}(f) = 0 \end{cases} \tag{4.10}$$

A reasonable selection method should favor good individuals by assigning them a reproduction rate $\bar{R}(f) > 1$ and punish bad individuals by a ratio $\bar{R}(f) < 1$.

4.1.6 Loss of Diversity

During every selection phase bad individuals are lost and replaced by copies of better individuals. Thereby a certain amount of "genetic material" is lost that was contained in the bad individuals. The number of individuals that are replaced corresponds to the strength of the "loss of diversity". This leads to the following definition.

Definition 4.8 (Loss of diversity) *The* loss of diversity D *is the proportion of individuals of a population that is not selected during the selection phase.*

Theorem 4.2 *If the reproduction rate $\bar{R}(f)$ increases strictly monotonously in f, the loss of diversity of a selection method is*

$$D = \frac{1}{N}\left(\bar{S}(f_z) - \bar{S}^*(f_z)\right) \tag{4.11}$$

where f_z denotes the fitness value such that $\bar{R}(f_z) = 1$.

Proof: As $\bar{R}(f)$ increases strictly monotonously in f, f_z is uniquely defined. For all fitness values $f \in (f_0, f_z]$ the reproduction rate is less than one. Hence the number of individuals that are not selected during selection is given by $\int_{f_0}^{f_z} (\bar{s}(x) - \bar{s}^*(x))\, dx$. It follows that

$$\begin{aligned} D &= \frac{1}{N} \int_{f_0}^{f_z} (\bar{s}(x) - \bar{s}^*(x))\, dx \\ &= \frac{1}{N} \left(\int_{f_0}^{f_z} \bar{s}(x)\, dx - \int_{f_0}^{f_z} \bar{s}^*(x)\, dx \right) \\ &= \frac{1}{N} \left(\bar{S}(f_z) - \bar{S}^*(f_z) \right) \end{aligned}$$

□

A small loss of diversity should reduce the risk of premature convergence as more unique genetic material is preserved for the next generation.

In his dissertation [Baker, 1989], Baker has introduced a similar measure called “reproduction rate RR”. RR gives the percentage of individuals that is selected to reproduce, hence $RR = 100(1 - D)$. Baker used this measure as a dynamic convergence measure. By observing the RR during the run of an Genetic Algorithm he tried to extract the state of convergence of the population. Note that in this thesis “reproduction rate” is used in the sense of Def. 4.7.

4.1.7 Selection Intensity

The term “selection intensity” or “selection pressure” is often used in different contexts and for different properties of a selection method. Goldberg and Deb [Goldberg and Deb, 1991] and Bäck [Bäck, 1994b] use the “takeover time” to define the selection pressure. Whitley calls the parameter c (see Section 4.5) of his ranking selection method selection pressure.

In this thesis the term “selection intensity” is used in the same way as in population genetics [Bulmer, 1980]. Mühlenbein has adopted the definition and applied it to Genetic Algorithms [Mühlenbein and Schlierkamp-Voosen, 1993]. Recently more and more researchers are using this term to characterize selection schemes [Thierens and Goldberg, 1994a; Thierens and Goldberg, 1994b; Bäck, 1995; Miller and Goldberg, 1995].

The change of the average fitness of the population due to selection is a reasonable measure for selection intensity. In population genetics the term selection intensity was introduced to obtain a normalized and dimensionless measure. The idea is to measure the progress due to selection by the so called “selection differential”, i.e., the difference between the average population fitness after and before selection. Dividing this selection differential by the mean variance of the population fitness leads to the desired dimensionless measure called selection intensity.

Definition 4.9 (Selection intensity) *The* selection intensity *of a selection method Ω for the fitness distribution $\bar{s}(f)$ is the standardized quantity*

$$I = \frac{\bar{M}^* - \bar{M}}{\bar{\sigma}} \tag{4.12}$$

By this, the selection intensity depends on the fitness distribution of the initial population. Hence, different fitness distributions will in general lead to different selection intensities for the same selection method. For comparison it is necessary to restrict oneself to a certain initial distribution. Using the normalized Gaussian distribution $\mathcal{G}(0,1)$ as initial fitness distribution leads to the following definition.

Definition 4.10 (Standardized selection intensity) *The* standardized selection intensity I_Ω *is the expected average fitness value of the population after applying the selection method Ω to the normal-*

ized Gaussian distribution $\mathcal{G}(0,1)(f) = \frac{1}{\sqrt{2\pi}} e^{-\frac{f^2}{2}}$*:*

$$I_\Omega = \int_{-\infty}^{\infty} f \, \overline{\Omega}^* (\mathcal{G}(0,1))(f) \, df \tag{4.13}$$

The "effective" average fitness value of a Gaussian distribution with mean μ and variance σ^2 can easily be derived as $\bar{M}^* = \sigma I_\Omega + \mu$. Note that this definition of the standardized selection intensity can only be applied if the selection method is scale and translation invariant. This is the case for all selection schemes examined in this thesis except proportional selection. Likewise this definition has no equivalent in the case of discrete fitness distributions. If the selection intensity for a discrete distribution has to be calculated, one must refer to Definition 4.9. In the remainder of this chapter the term "selection intensity" is used as equivalent for "standardized selection intensity" as the intention is to compare selection schemes.

4.1.8 Selection Variance

In addition to selection intensity the term of "selection variance" is introduced. The definition is analogous to the definition of the selection intensity, but describes the new variance of the fitness distribution after selection.

Definition 4.11 (Selection variance) *The* selection variance Φ *is the normalized expected variance of the fitness distribution of the population after applying the selection method* Ω *to the fitness distribution* $\bar{s}(f)$*, i.e.*

$$\Phi = \frac{(\bar{\sigma}^*)^2}{\bar{\sigma}^2} \tag{4.14}$$

For comparison the standardized selection variance is used.

Definition 4.12 (Standardized selection variance) *The* standardized selection variance Φ_Ω *is the normalized expected variance of* *the fitness distribution of the population after applying the selection method* Ω *to the normalized Gaussian distribution* $\mathcal{G}(0,1)$.

$$\Phi_\Omega = \int_{-\infty}^{\infty} (f - I_\Omega)^2 \, \overline{\Omega}^* (\mathcal{G}(0,1))(f) \, df \tag{4.15}$$

that is equivalent to

$$\Phi_\Omega = \int_{-\infty}^{\infty} f^2 \, \overline{\Omega}^* (\mathcal{G}(0,1))(f) \, df - I_\Omega^2 \tag{4.16}$$

Note that there is a difference between the selection variance and the loss of diversity. The loss of diversity gives the proportion of individuals that are not selected, regardless of their fitness value. The standardized selection variance is defined as the new variance of the fitness distribution assuming a Gaussian initial fitness distribution. Hence a selection variance of one means that the variance is not changed by selection. A selection variance less than one corresponds to a decrease in variance. The lowest possible value of Φ_Ω is zero, which means that the variance of the fitness values of the population after selection is itself zero. Again the term the "selection variance" is used as equivalent for "standardized selection variance".

4.2 Proportional Selection

Proportional selection is the original selection method proposed for genetic algorithms by Holland [Holland, 1975]. The probability of an individual to be selected is simply proportionate to its fitness value, i.e.,

$$p_i = \frac{f_i}{NM} \tag{4.17}$$

Algorithm 6 displays in pseudo code this method. The time complexity of the algorithm is $\mathcal{O}(N)$.

Algorithm 6: (Proportional Selection)

Input: The population $\vec{J} = (J_1, \ldots, J_N)$,
the fitness values of the population $\vec{\phi} = (\phi_1, \ldots, \phi_N)$.
Output: The population after selection $\vec{J'} = (J'_1, \ldots, J'_N)$

```
proportional(J⃗, φ⃗):
    sum_0 ← 0
    for i ← 1 to N do
        sum_i ← sum_{i-1} + φ_i
    od
    for i ← 1 to N do
        r ← random[0,sum_N[
        { determine l, such that sum_{l-1} ≤ r < sum_l }
        l ← calculate_index(r)
        J'_i ← J_l
    od
    return J⃗'
```

Obviously, this mechanism will only work if all fitness values are greater than zero. Furthermore the selection probabilities strongly depend on the scaling of the fitness function. As an example, assume a population of 10 individuals with the fitness values $\vec{\phi} = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)$. The selection probability for the best individual is hence $p_b \approx 18.2\%$ and for the worst $p_w \approx 1.8\%$. If the fitness function is shifted by 100, i.e., the constant value 100 is added to every fitness value, one calculates $p'_b \approx 10.4\%$ and $p'_w \approx 9.6\%$. The selection probabilities of the best and the worst individual are now almost identical. This undesirable property arises from the fact that proportional selection is not translation invariant (see e.g. [de la Maza and Tidor, 1993]). Because of this several *scaling methods* have been proposed to ensure that proportional selection works, e.g. linear static scaling, linear dynamic scaling, exponential scaling, logarithmic scaling [Grefenstette and Baker, 1989]; sigma truncation [Brill *et al.*, 1992]. Another method to improve proportional selection is the "over selection" of a certain percentage of the best individuals, e.g., to force that 80 % of all individuals are taken from the best 20 % of the population. This method was used in [Koza, 1992]. In [Mühlenbein and Schlierkamp-Voosen, 1993] it is already stated that "these modifications are necessary, not tricks to speed up the algorithm". The analysis in the remainder of this section will confirm this statement.

First, the expected fitness distribution after selection is calculated.

Theorem 4.3 *The expected fitness distribution after performing proportional selection on the distribution s is*

$$\Omega_P^*(s)(f_i) = s^*(f_i) = s(f_i)\frac{f}{M} \tag{4.18}$$

Proof: This follows immediately from (4.17). □

4.2.1 Reproduction Rate

Corollary 4.1 *The reproduction rate of proportional selection is*

$$\bar{R}_P(f) = \frac{f}{\bar{M}} \tag{4.19}$$

This is directly obtained by substituting (4.18) in (4.10).

The reproduction rate is proportionate to the fitness value of an individual. If all fitness values are close together (as it was in the example at the beginning of this section) all individuals have almost the same reproduction rate $R \approx 1$.

4.2.2 Selection Intensity

As proportional selection is not translation invariant the definition of standardized selection intensity (Def. 4.10) cannot be applied.

Hence, the results obtained by Mühlenbein and Schlierkamp-Voosen [Mühlenbein and Schlierkamp-Voosen, 1993] are cited.

Theorem 4.4 [Mühlenbein and Schlierkamp-Voosen, 1993] *The standardized selection intensity of proportional selection is*

$$I_P = \frac{\bar{\sigma}}{\bar{M}} \tag{4.20}$$

where $\bar{\sigma}$ is the mean variance of the fitness values of the population before selection.

Proof: See [Mühlenbein and Schlierkamp-Voosen, 1993]. □

The other properties like the selection variance an the loss of diversity are difficult to investigate for proportional selection. The crucial point is the explicit occurrence of the fitness value in the expected fitness distribution after selection (4.18). Hence an analysis is only possible if some further assumptions on the initial fitness distribution are made. Due to this other work on proportional selection assume some special functions to be optimized (e.g. [Goldberg and Deb, 1991]).

Another weak point is that the selection intensity even in the early stage of the optimization (when the variance is high) is too low. Measurements on a broad range of problems sometimes showed a negative selection intensity. This means that in some cases (due to sampling) there is a *decrease* in average population fitness. Seldom a very high selection intensity occurred ($I \approx 1.8$) if a *super-individual* was created. But the measured average selection intensity was in range of 0.1 to 0.3.

All the undesired properties together lead to the conclusion that proportional selection is an unsuitable selection scheme for optimization.

4.3 Tournament Selection

In tournament selection a group of t individuals is randomly chosen from the population. They may be drawn from the population with or without replacement. This group takes part in a "tournament", i.e., a winning individual is determined depending on its fitness value. The best individual having the highest fitness value is usually chosen deterministically though occasionally a stochastic selection may be made. In both cases only the winner is inserted into the next population and the process is repeated N times to obtain a new population. Often, tournaments are held between two individuals (binary tournament). However, this can be generalized to an arbitrary group size t called *tournament size*.

The following description assumes that the individuals are drawn with replacement and the winning individual is deterministically selected (Algorithm 7).

Algorithm 7: (Tournament Selection)

Input: The population $\vec{J} = (J_1, \ldots, J_N)$,
the fitness values of the population $\vec{\phi} = (\phi_1, \ldots, \phi_N)$.
the tournament size $t \in \{1, 2, ..., N\}$
Output: The population after selection $\vec{J'} = (J'_1, \ldots, J'_N)$

tournament(t,$\vec{J}$, $\vec{\phi}$):
  for $i \leftarrow 1$ **to** N **do**
    $J'_i \leftarrow$ best fit individual out of t randomly picked
    individuals from $\vec{J}$
  od
  return $\vec{J'}$

The outline of the algorithm shows that tournament selection can be implemented very efficiently as no sorting of the population is required. Implemented in the way above it has the time complexity $\mathcal{O}(N)$.

Using the notation introduced in the previous section, the entire fitness distribution after selection can be predicted. The prediction is made for the discrete (exact) fitness distribution as well as for a continuous fitness distribution. These results are published in [Blickle and Thiele, 1995]. The calculations assume that tournament selection is done with replacement.

Theorem 4.5 *The expected fitness distribution after performing tournament selection with tournament size t on the distribution s is*

$$\Omega_T^*(s,t)(f_i) = s^*(f_i) = N\left(\left(\frac{S(f_i)}{N}\right)^t - \left(\frac{S(f_{i-1})}{N}\right)^t\right) \tag{4.21}$$

Proof: First the expected number of individuals with fitness f_i or worse, i.e. $S^*(f_i)$, is calculated. An individual with fitness f_i or worse can only win the tournament if all other individuals in the tournament have a fitness of f_i or worse. This means to calculate the probability that all t individuals have also a fitness of f_i or worse. As the probability to choose an individual with fitness f_i or worse is given by $\frac{S(f_i)}{N}$ it follows that

$$S^*(f_i) = N\left(\frac{S(f_i)}{N}\right)^t \tag{4.22}$$

Using this equation and the relation $s^*(f_i) = S^*(f_i) - S^*(f_{i-1})$ (see Definition 4.4) (4.21) is obtained. □

Equation (4.21) shows the strong influence of the tournament size t on the behavior of the selection scheme. Obviously for $t = 1$ (in average) the unchanged initial distribution is obtained as $\Omega_T^*(s,1)(f_i) = N\left(\frac{S(f_i)}{N} - \frac{S(f_{i-1})}{N}\right) = S(f_i) - S(f_{i-1}) = s(f_i)$.

In [Bäck, 1994b] the probability of individual number i to be selected by tournament selection is given by $p_i = N^{-t}((N-i+1)^t -$

$(N-i)^t)$, under the assumption that the individuals are ordered according to their fitness value $f(J_1) \leq f(J_2) \leq ... \leq f(J_N)$. Note that Bäck uses an "reversed" fitness function where the best individual has the lowest index. For comparison with the results above the transformation $j = N - i + 1$ is used:

$$p_j = N^{-t}(j^t - (j-1)^t) \qquad 1 \leq j \leq N \tag{4.23}$$

This formula is a special case of (4.21) with *all individuals having different fitness values.* Then $s(f_i) = 1$ for all $i \in [1, N]$ and $S(f_i) = i$ and $p_i = \frac{s^*(f_i)}{N}$ yields the same equation as given by Bäck. Note that (4.23) is not valid if some individuals have the same fitness value.

Example 4.3 *Using the discrete fitness distribution from Example 4.1 (Figure 4.1) the fitness distribution shown in Figure 4.2 is obtained after applying tournament selection with a tournament size $t = 3$.*

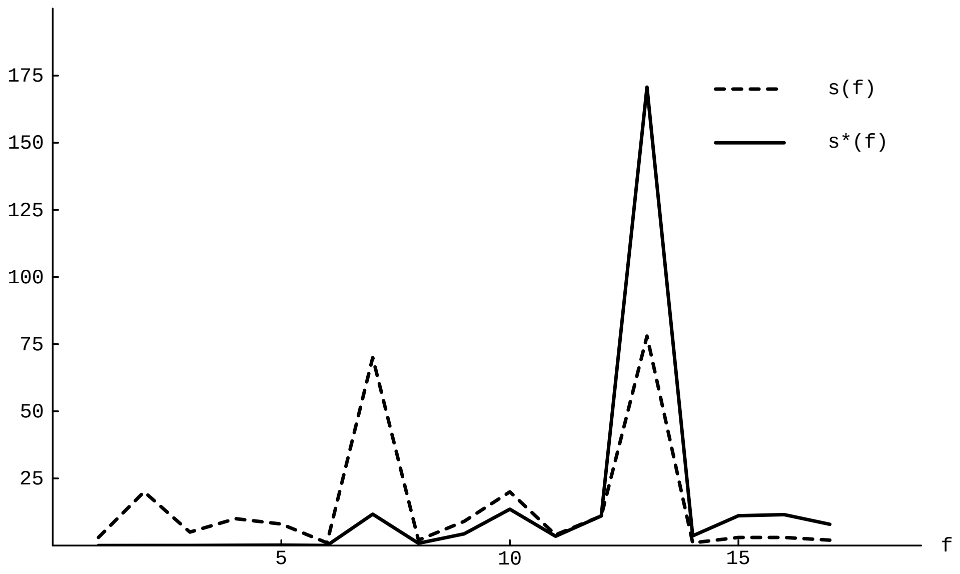


Figure 4.2: *The initial fitness distribution (dashed) and the resulting expected fitness distribution after applying tournament selection with a tournament size of 3.*

Next the expected fitness distribution for the continuous case is derived.

Theorem 4.6 *Let $\bar{s}(f)$ be the continuous fitness distribution of the population. Then the expected fitness distribution after performing tournament selection with tournament size t is*

$$\overline{\Omega}^*_T(\bar{s},t))(f) = \bar{s}^*(f) = t\bar{s}(f)\left(\frac{\bar{S}(f)}{N}\right)^{t-1} \tag{4.24}$$

Proof: Analogous to the proof of the discrete case the probability of an individual with fitness f or worse to win the tournament is given by

$$\bar{S}^*(f) = N\left(\frac{\bar{S}(f)}{N}\right)^t \tag{4.25}$$

As $\bar{s}^*(f) = \frac{d\bar{S}^*(f)}{df}$, (4.24) is obtained. □

4.3.1 Concatenation of Tournament Selection

An interesting property of tournament selection is the concatenation of several selection phases. Assuming an arbitrary population with the fitness distribution $\bar{s}$, a tournament selection with tournament size t_1 is applied to this population and then on the resulting population tournament selection with tournament size t_2. The obtained fitness distribution is the same as if only one tournament selection with tournament size $t_1 \cdot t_2$ is applied to the initial distribution $\bar{s}$.

Theorem 4.7 *Let $\bar{s}$ be a continuous fitness distribution and $t_1, t_2 \geq 1$ two tournament sizes. Then the following equation holds*

$$\overline{\Omega}^*_T(\overline{\Omega}^*_T(\bar{s},t_1),t_2)(f) = \overline{\Omega}^*_T(\bar{s},t_1\cdot t_2)(f) \tag{4.26}$$

Proof:

$$\overline{\Omega}_T^*(\overline{\Omega}_T^*(\bar{s},t_1),t_2)(f) = t_2\overline{\Omega}_T^*(\bar{s},t_1)(f)\left(\frac{1}{N}\int_{f_0}^{f}\overline{\Omega}_T^*(\bar{s},t_1)(x)\,dx\right)^{t_2-1}$$

$$= t_2 t_1 \bar{s}(f)\left(\frac{1}{N}\int_{f_0}^{f}\bar{s}(x)\,dx\right)^{t_1-1}$$

$$* \left(\frac{1}{N}\int_{f_0}^{f} t_1\bar{s}(x)\left(\frac{1}{N}\int_{f_0}^{x}\bar{s}(y)\,dy\right)^{t_1-1}dx\right)^{t_2-1}$$

As

$$\int_{f_0}^{f} t_1\bar{s}(x)\left(\frac{1}{N}\int_{f_0}^{x}\bar{s}(y)\,dy\right)^{t_1-1}dx = N\left(\frac{1}{N}\int_{f_0}^{f}\bar{s}(x)\,dx\right)^{t_1}$$

it follows that

$$\begin{aligned}
&\overline{\Omega}_T^*(\overline{\Omega}_T^*(\bar{s},t_1),t_2)(f)\\
&= t_2 t_1 \bar{s}(f)\left(\frac{1}{N}\int_{f_0}^{f}\bar{s}(x)\,dx\right)^{t_1-1}\left(\left(\frac{1}{N}\int_{f_0}^{f}\bar{s}(x)\,dx\right)^{t_1}\right)^{t_2-1}\\
&= t_2 t_1 \bar{s}(f)\left(\frac{1}{N}\int_{f_0}^{f}\bar{s}(x)\,dx\right)^{t_1-1}\left(\frac{1}{N}\int_{f_0}^{f}\bar{s}(x)\,dx\right)^{t_1(t_2-1)}\\
&= t_2 t_1 \bar{s}(f)\left(\frac{1}{N}\int_{f_0}^{f}\bar{s}(x)\,dx\right)^{t_1 t_2-1} = \overline{\Omega}_T^*(\bar{s},t_1\,t_2)(f)
\end{aligned}$$

□

In [Goldberg and Deb, 1991] the proportion P_k of best-fit individuals after k selections with tournament size t (without recombination) is given as

$$P_k = 1 - (1-P_0)^{t^k} \tag{4.27}$$

This can be obtained as a special case from Theorem 4.7, if only best-fit individuals are considered.

Corollary 4.2 *Let $\bar{s}(f)$ be a fitness distribution representable as*

$$\bar{s}(f) = \rho g(f) \left(\frac{\int_{f_0}^{f} g(x)\,dx}{N} \right)^{\rho - 1} \tag{4.28}$$

with $\rho \geq 1$ and $\int_{f_0}^{f_n} g(x)\,dx = N$. Then the expected distribution after tournament selection with tournament size t is

$$\bar{s}^*(f) = \rho\, t\, g(f) \left(\frac{\int_{f_0}^{f} g(x)\,dx}{N} \right)^{\rho t - 1} \tag{4.29}$$

Proof: If $\bar{s}(f)$ is assumed to be the result of applying tournament selection with tournament size ρ on the distribution $g(f)$, (4.29) is directly obtained using Theorem 4.7. $\square$

4.3.2 Reproduction Rate

Corollary 4.3 *The reproduction rate of tournament selection is*

$$\bar{R}_T(f) = \frac{\bar{s}^*(f)}{\bar{s}(f)} = t \left(\frac{\bar{S}(f)}{N} \right)^{t-1} \tag{4.30}$$

This is directly obtained by substituting (4.24) in (4.10).

Individuals with the lowest fitness have a reproduction rate of almost zero and the individuals with the highest fitness have a reproduction rate of t.

4.3.3 Loss of Diversity

Theorem 4.8 *The loss of diversity D_T of tournament selection is*

$$D_T(t) = t^{-\frac{1}{t-1}} - t^{-\frac{t}{t-1}} \tag{4.31}$$

Proof: $\bar{S}(f_z)$ can be determined using (4.30) (refer to Theorem 4.2 for the definition of f_z):

$$\bar{S}(f_z) = N\, t^{-\frac{1}{t-1}} \tag{4.32}$$

Using Definition 4.8 and (4.32) one obtains:

$$\begin{aligned} D_T(t) &= \frac{1}{N}\left(\bar{S}(f_z) - \bar{S}^*(f_z)\right) \\ &= \frac{\bar{S}(f_z)}{N} - \left(\frac{\bar{S}(f_z)}{N}\right)^t \\ &= t^{-\frac{1}{t-1}} - t^{-\frac{t}{t-1}} \end{aligned}$$

□

It turns out that the number of individuals lost increases with the tournament size (see Fig. 4.3). About the half of the population is lost at tournament size $t = 5$. Note that the loss of diversity is independent of the initial fitness distribution.

4.3.4 Selection Intensity

To calculate the selection intensity the average fitness of the population after applying tournament selection on the normalized Gaussian distribution $\mathcal{G}(0,1)$ is needed. Using Definition 4.5 leads to

$$I_T(t) = \int_{-\infty}^{\infty} t\, x\, \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \left(\int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{y^2}{2}}\, dy\right)^{t-1} dx \tag{4.33}$$

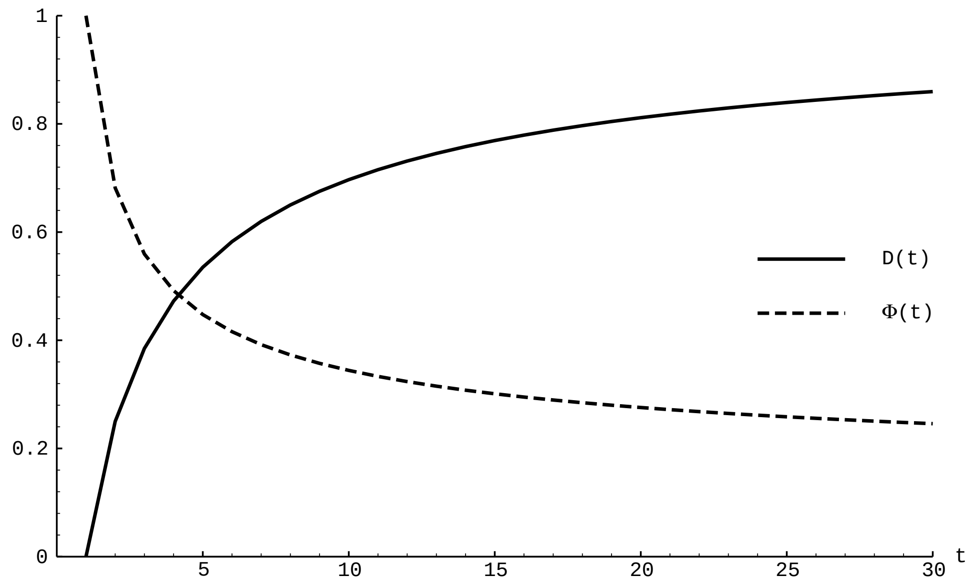


Figure 4.3: *The loss of diversity $D_T(t)$ (solid line) and the selection variance $\Phi_T(t)$ (broken line) of tournament selection.*

These integral equations can be solved analytically for the cases $t = 1, \ldots, 5$ ([Blickle and Thiele, 1995; Bäck, 1995; Arnold *et al.*, 1992]):

$$
\begin{aligned}
I_T(1) &= 0 \\
I_T(2) &= \frac{1}{\sqrt{\pi}} \\
I_T(3) &= \frac{3}{2\sqrt{\pi}} \\
I_T(4) &= \frac{6}{\pi\sqrt{\pi}} \arctan\sqrt{2} \\
I_T(5) &= \frac{10}{\sqrt{\pi}}\left(\frac{3}{2\pi}\arctan\sqrt{2} - \frac{1}{4}\right)
\end{aligned}
$$

For a tournament size of two Thierens and Goldberg derive the same average fitness value [Thierens and Goldberg, 1994a] in a completely different manner. However their formulation can not be extended to other tournament sizes.

For larger tournament sizes (4.33) can be accurately evaluated by numerical integration. The result is shown in Fig. 4.4 for a tournament size from 1 to 30. But an explicit expression of (4.33) may not exist. Using the steepest descent method (see, e.g. [Henrici, 1977]) an approximation for large tournament sizes can be given. But even for small tournament sizes this approximation gives acceptable results.

The calculations lead to the following recursion equation:

$$I_T(t)^i \approx \sqrt{c_i(\ln(t) - \ln(I_T(t)^{i-1}))} \tag{4.34}$$

with $I_T(t)^0 = 1$ and i the recursion depth. The calculation of the constants c_i is difficult. Taking a rough approximation with $i = 2$ the following equation is obtained that approximates (4.33) with an relative error of less than 2.4% for $t \in [2, 5]$, for tournament sizes $t > 5$ the relative error is less than 1%:

$$I_T(t) \approx \sqrt{2(\ln(t) - \ln(\sqrt{4.14\ \ln(t)}))} \tag{4.35}$$

4.3.5 Selection Variance

To determine the selection variance the following equation needs to be solved:

$$\Phi_T(t) = \int_{-\infty}^{\infty} t\ (x - I_T(t))^2\ \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \left(\int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{y^2}{2}}\, dy \right)^{t-1} dx \tag{4.36}$$

For a binary tournament one calculates

$$\Phi_T(2) \quad = \quad 1 - \frac{1}{\pi}$$

Here again (4.36) can be solved by numerical integration. The dependence of the selection variance on the tournament size is shown in Fig. 4.3.

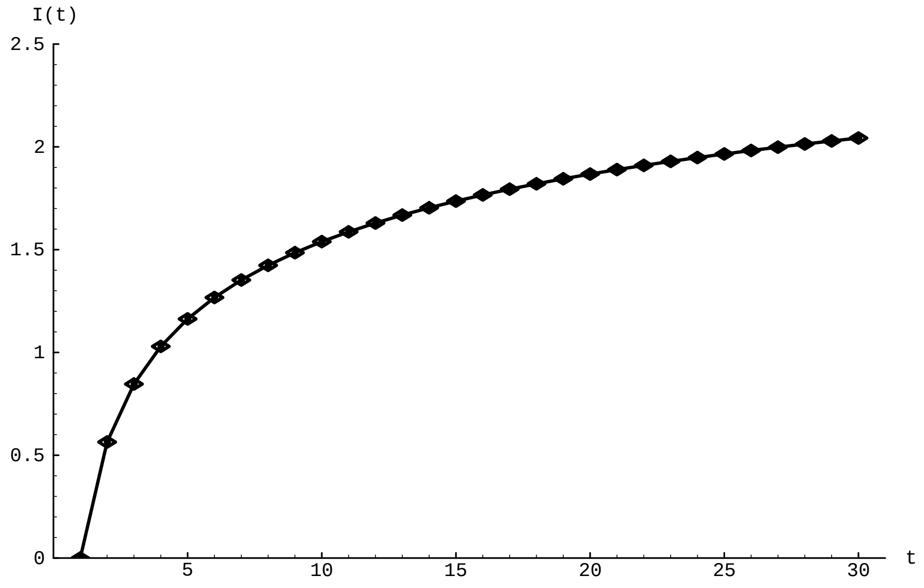


Figure 4.4: *Dependence of the selection intensity $I_T(t)$ on the tournament size t.*

To obtain a useful analytic approximation for the selection variance, symbolic regression using the Genetic Programming optimization method is used. Details about the way the data were computed can be found in Section 5.4.3 and Appendix B. The following expression approximates the selection variance with an relative error of less than 1.6% for $t \in \{1, \ldots, 30\}$:

$$\Phi_T(t) \approx \frac{0.918}{\ln(1.186 + 1.328t)} \qquad ; t \in \{1, \ldots, 30\} \tag{4.37}$$

4.4 Truncation Selection

In Truncation selection with threshold T only the fraction T best individuals are selected and they all have the same selection probability. This selection method is often used by breeders and in population genetics [Bulmer, 1980; Crow and Kimura, 1970]. Mühlenbein introduced this selection scheme to the domain of genetic algorithms

[Mühlenbein and Schlierkamp-Voosen, 1993]. This method is equivalent to (μ, λ)-selection used in evolution strategies with $T = \frac{\mu}{\lambda}$ [Bäck, 1995].

The outline of the algorithm is given by Algorithm 8.

Algorithm 8: (Truncation Selection)

Input: The population $\vec{J} = (J_1, \ldots, J_N)$,
the fitness values of the population $\vec{\phi} = (\phi_1, \ldots, \phi_N)$,
the truncation threshold $T \in [0, 1]$
Output: The population after selection $\vec{J'} = (J'_1, dots, J'_N)$

```
truncation(T, J⃗, φ⃗):
    J⃗* ← sorted population J⃗ according fitness φ⃗
        with least fit individual at the first position
    for i ← 1 to N do
        r ← random{ ⌈(1 − T)N⌉, ..., N}
        J'_i ← J*_r
    od
    return J⃗'
```

As a sorting of the population is required, truncation selection has a time complexity of $\mathcal{O}(N \log N)$.

Although this selection method has been investigated several times it will be described using the methods derived here, as additional important properties are derived.

Theorem 4.9 *The expected fitness distribution after performing truncation selection with threshold T on the distribution s is*

$$\Omega^*_\Gamma(s, T)(f_i) = \begin{cases} 0 & : \quad S(f_i) \leq (1-T)N \\ \frac{S(f_i)-(1-T)N}{T} & : \quad S(f_{i-1}) \leq (1-T)N < S(f_i) \\ \frac{s(f_i)}{T} & : \quad else \end{cases} \tag{4.38}$$

Proof: The first case in (4.38) gives zero offspring to individuals with a fitness value below the truncation threshold. The second case reflects the fact that threshold may lie within $s(f_i)$. Then only the fraction above the threshold $(S(f_i) - (1 - T)N)$ is selected. These fraction is in average copied $\frac{1}{T}$ times. The last case in (4.38) gives all individuals above the threshold the multiplication factor $\frac{1}{T}$, necessary to keep the population size constant. $\square$

Example 4.4 *Using the discrete fitness distribution from Example 4.1 the fitness distribution shown in Figure 4.5 is obtained after applying truncation selection with threshold $T = 0.47$.*

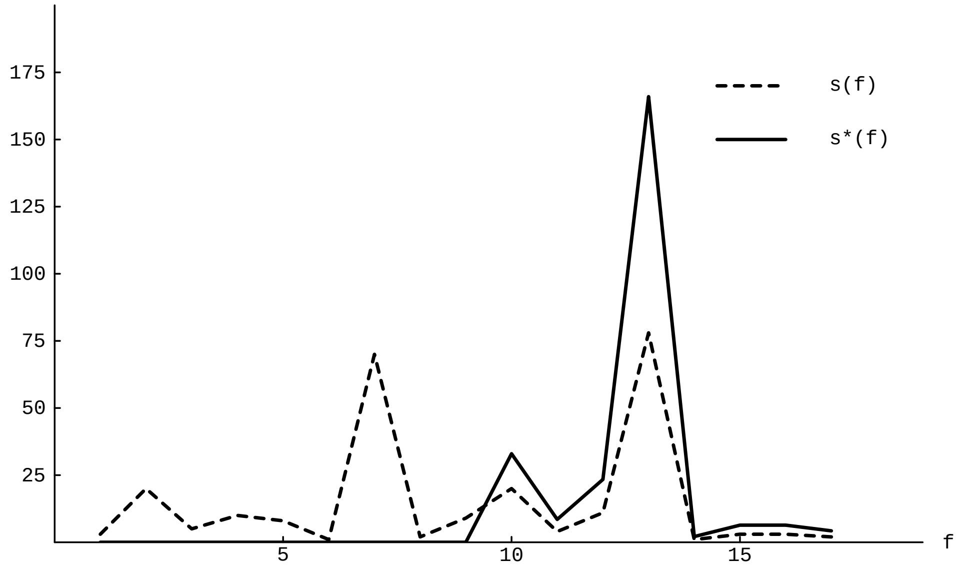


Figure 4.5: *The initial fitness distribution (dashed) and the resulting expected fitness distribution after applying truncation selection with truncation threshold $T = 0.47$.*

Theorem 4.10 *Let $\bar{s}(f)$ be the continuous distribution of the population. Then the expected fitness distribution after performing trunca-*

tion selection with threshold T is

$$\overline{\Omega}^*_\Gamma(\bar{s},T)(f) = \begin{cases} \frac{\bar{s}(f)}{T} & : \quad \bar{S}(f) > (1-T)N \\ 0 & : \quad else \end{cases} \tag{4.39}$$

Proof: As $\bar{S}(f)$ gives the cumulative fitness distribution, it follows from the construction of truncation selection that all individuals with $\bar{S}(f) < (1-T)N$ are truncated. As the population size is kept constant during selection, all other individuals must be copied in average $\frac{1}{T}$ times. □

4.4.1 Reproduction Rate

From the construction of the selection method the reproduction rate can easily be derived.

Corollary 4.4 *The reproduction rate of truncation selection is*

$$\bar{R}_\Gamma(f) = \begin{cases} \frac{1}{T} & : \quad \bar{S}(f) > (1-T)N \\ 0 & : \quad else \end{cases} \tag{4.40}$$

This is directly obtained by substituting (4.39) in (4.10).

4.4.2 Loss of Diversity

By construction of the selection method only the fraction T of the population will be selected, i.e. the loss of diversity is

$$D_\Gamma(T) = 1 - T \tag{4.41}$$

The loss of diversity is depicted in Fig. 4.6.

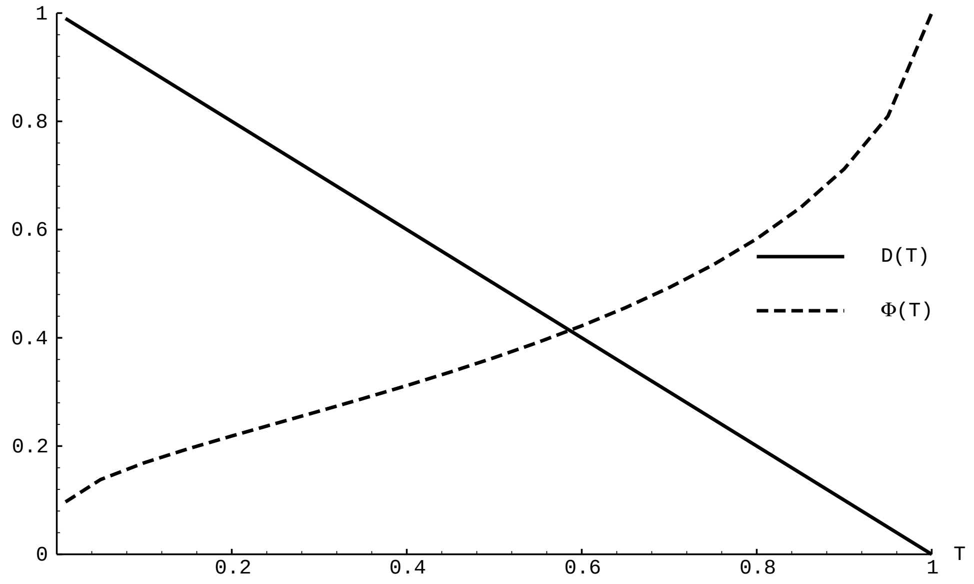


Figure 4.6: *The loss of diversity $D_\Gamma(t)$ (solid line) and the selection variance $\Phi_\Gamma(t)$ (broken line) of truncation selection.*

4.4.3 Selection Intensity

The results presented in this subsection were also derived by [Crow and Kimura, 1970].

Theorem 4.11 *The selection intensity of truncation selection is*

$$I_\Gamma(T) = \frac{1}{T}\frac{1}{\sqrt{2\pi}}e^{-\frac{f_c^2}{2}} \tag{4.42}$$

where f_c is determined by

$$T = \int_{f_c}^{\infty} \frac{1}{\sqrt{2\pi}}e^{-\frac{f^2}{2}}\,df. \tag{4.43}$$

Proof: The selection intensity is defined as the average fitness of the population after selection assuming an initial normalized Gaussian distribution $\mathcal{G}(0,1)$, hence $I = \int_{-\infty}^{\infty} \bar{\Omega}(\mathcal{G}(0,1))(f)\ f\ df$. As no individual with a fitness value worse than f_c will be selected, the lower integration bound can be replaced by f_c. Here f_c is determined by

$$\bar{S}(f_c) = (1-T)N = 1-T \tag{4.44}$$

because $N = 1$ for the normalized Gaussian distribution.

Hence, I_Γ can be computed as

$$\begin{aligned} I_\Gamma(T) &= \int_{f_c}^{\infty} \frac{1}{T}\frac{1}{\sqrt{2\pi}} e^{-\frac{f^2}{2}} f\ df \\ &= \frac{1}{T}\frac{1}{\sqrt{2\pi}} e^{-\frac{f_c^2}{2}} \end{aligned}$$

Here f_c is determined by (4.44). Solving (4.44) for T yields

$$\begin{aligned} T &= 1 - \int_{-\infty}^{f_c} \frac{1}{\sqrt{2\pi}} e^{-\frac{f^2}{2}} df \\ &= \int_{f_c}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-\frac{f^2}{2}} df \end{aligned}$$

$\square$

A lower bound for the selection intensity reported by [Nagaraja, 1982] is $I_\Gamma(T) \leq \sqrt{\frac{1-T}{T}}$.

Figure 4.7 shows the selection intensity in dependence of parameter T.

4.4.4 Selection Variance

Theorem 4.12 *The selection variance of truncation selection is*

$$\Phi_\Gamma(T) = 1 - I_\Gamma(T)(I_\Gamma(T) - f_c) \tag{4.45}$$

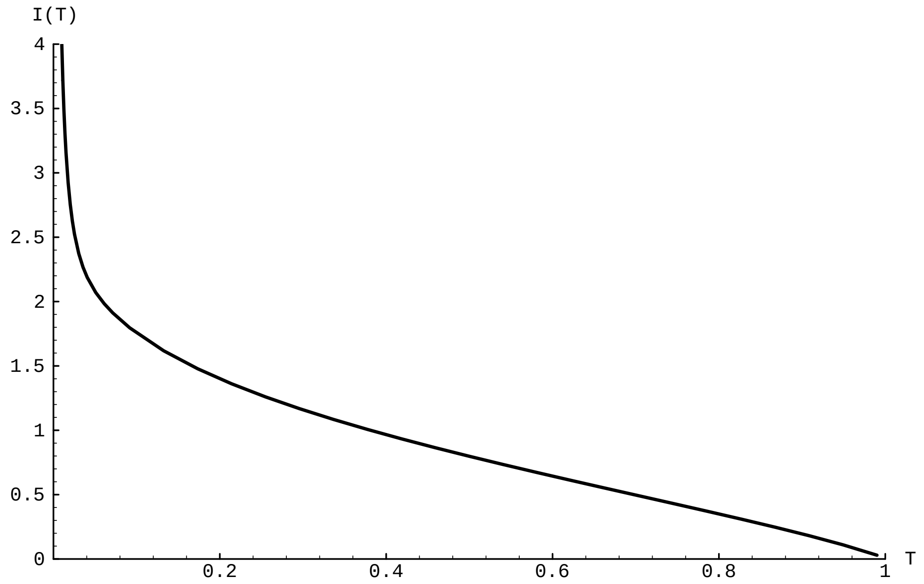


Figure 4.7: *The Selection intensity $I_\Gamma(T)$ of truncation selection.*

Proof: The substitution of (4.39) in the definition equation (4.16) gives

$$\Phi_\Gamma(T) = \int_{f_c}^{\infty} f^2 \frac{1}{T} \frac{1}{\sqrt{2\pi}} e^{-\frac{f^2}{2}} \, df - (I_\Gamma(T))^2$$

A partial integration with $u(f) = f$ and $\frac{\partial v}{\partial f} = f e^{-\frac{f^2}{2}}$ yields

$$\Phi_\Gamma(T) = \frac{f_c}{T\sqrt{2\pi}} e^{-\frac{f_c^2}{2}} + \frac{1}{T\sqrt{2\pi}} \int_{f_c}^{\infty} e^{-\frac{f^2}{2}} \, df - (I_\Gamma(T))^2$$

Substituting (4.42) and (4.43) simplifies this this equation to (4.45). □

The selection variance is plotted in Fig. 4.6. Equation (4.45) has also been derived in [Bulmer, 1980].

4.5 Linear Ranking Selection

Linear ranking selection was first suggested by Baker to eliminate the serious disadvantages of proportionate selection [Grefenstette and Baker, 1989; Whitley, 1989]. For linear ranking selection the individuals are sorted according their fitness values and the rank N is assigned to the best individual and the rank 1 to the worst. The selection probability is linearly assigned to the individuals according to their rank:

$$p_i = \frac{1}{N}\left(\eta^- + (\eta^+ - \eta^-)\frac{i-1}{N-1}\right) \;; \qquad i \in \{1, \ldots, N\} \tag{4.46}$$

Here $\frac{\eta^-}{N}$ is the probability of the worst individual to be selected and $\frac{\eta^+}{N}$ the probability of the best individual to be selected. As the population size is held constant, the conditions $\eta^+ = 2 - \eta^-$ and $\eta^- \geq 0$ must be fulfilled. Note that all individuals get a different rank, i.e., a different selection probability, even if they have the same fitness value.

Koza [Koza, 1992] determines the probability by a multiplication factor r_m that determines the gradient of the linear function. A transformation into the form of (4.46) is possible by $\eta^- = \frac{2}{r_m+1}$ and $\eta^+ = \frac{2r_m}{r_m+1}$.

Whitley [Whitley, 1989] describes the ranking selection by transforming an equally distributed random variable $\chi \in [0, 1]$ to determine the index of the selected individual

$$j = \lfloor \frac{N}{2(c-1)}\left(c - \sqrt{c^2 - 4(c-1)\chi}\right)\rfloor \tag{4.47}$$

where c is a parameter called "selection bias". Bäck has shown that for $1 < c \leq 2$ this method is almost identical to the probabilities in (4.46) with $\eta^+ = c$ [Bäck, 1994b].

The pseudo-code implementation of linear ranking selection is given by Algorithm 9. The method requires the sorting of the population, hence the complexity of the algorithm is dominated by the complexity of sorting, i.e. $\mathcal{O}(N \log N)$.

Algorithm 9: (Linear Ranking Selection)

Input: The population $\vec{J} = (J_1, \ldots, J_N)$,
the fitness values of the population $\vec{\phi} = (\phi_1, \ldots, \phi_N)$,
the reproduction rate of the worst individual $\eta^- \in [0, 1]$
Output: The population after selection $\vec{J'} = (J'_1, \ldots, J'_N)$

linear_ranking(η^-,$\vec{J}$, $\vec{\phi}$):
  $\vec{J^*} \leftarrow$ sorted population $\vec{J}$ according fitness $\vec{\phi}$
    with least fit individual at the first position
  $sum_0 \leftarrow 0$
  for $i \leftarrow 1$ **to** N **do**
    $sum_i \leftarrow sum_{i-1} + p_i$ (Equation 4.46)
  od
  for $i \leftarrow 1$ **to** N **do**
    $r \leftarrow$ random[0,1[
    $J'_i \leftarrow J^*_l$ such that $sum_{l-1} \leq r < sum_l$
  od
  return $\vec{J'}$

Theorem 4.13 *The expected fitness distribution after performing ranking selection with η^- on the distribution s is*

$$\Omega^*_R(s, \eta^-)(f_i) = s^*(f_i) = s(f_i)\frac{N\eta^- - 1}{N - 1} + \frac{1 - \eta^-}{N - 1}\left(S(f_i)^2 - S(f_{i-1})^2\right) \tag{4.48}$$

Proof: First the expected number of individuals with fitness f_i or worse is calculated, i.e. $S^*(f_i)$. As the individuals are sorted according to their fitness value this number is given by the sum of the probabilities of the $S^*(f_i)$ least fit individuals:

$$S^*(f_i) \quad = \quad N \sum_{j=1}^{S(f_i)} p_j$$

$$
\begin{aligned}
&= \eta^- S(f_i) + \frac{\eta^+ - \eta^-}{N-1} \sum_{j=1}^{S(f_i)} j - 1 \\
&= \eta^- S(f_i) + \frac{\eta^+ - \eta^-}{N-1} \frac{1}{2} S(f_i)\left(S(f_i) - 1\right)
\end{aligned}
$$

With $\eta^+ = 2 - \eta^-$ and $s^*(f_i) = S^*(f_i) - S^*(f_{i-1})$ follows

$$
\begin{aligned}
s^*(f_i) &= \eta^- \left(S(f_i) - S(f_{i-1})\right) \\
&\quad + \frac{1-\eta^-}{N-1} \left(S(f_i)(S(f_i) - 1) - S(f_{i-1})(S(f_{i-1}) - 1)\right) \\
&= \eta^- s(f_i) + \frac{1-\eta^-}{N-1} \left(S(f_i)^2 - S(f_{i-1})^2 - s(f_i)\right) \\
&= s(f_i) \frac{N\eta^- - 1}{N-1} + \frac{1-\eta^-}{N-1} \left(S(f_i)^2 - S(f_{i-1})^2\right)
\end{aligned}
$$

$\square$

Example 4.5 *Using the discrete fitness distribution from Example 4.1 the fitness distribution shown in Figure 4.8 is obtained after applying linear ranking selection with $\eta^- = 0.1$.*

Theorem 4.14 *Let $\bar{s}(f)$ be the continuous fitness distribution of the population. Then the expected fitness distribution after performing ranking selection $\overline{\Omega}_R$ with η^- on the distribution $\bar{s}$ is*

$$
\overline{\Omega}^*_R(\bar{s}, \eta^-)(f) = \bar{s}^*(f) = \eta^- \bar{s}(f) + 2\frac{1-\eta^-}{N} \bar{S}(f)\bar{s}(f) \tag{4.49}
$$

Proof: As the continuous form of (4.46) is given by $\bar{p}(x) = \frac{1}{N}(\eta^- + \frac{\eta^+ - \eta^-}{N} x)$ one calculates $\bar{S}(f)$ using $\eta^+ = 2 - \eta^-$:

$$
\bar{S}^*(f) = N \int_0^{\bar{S}(f)} \bar{p}(x)\, dx
$$

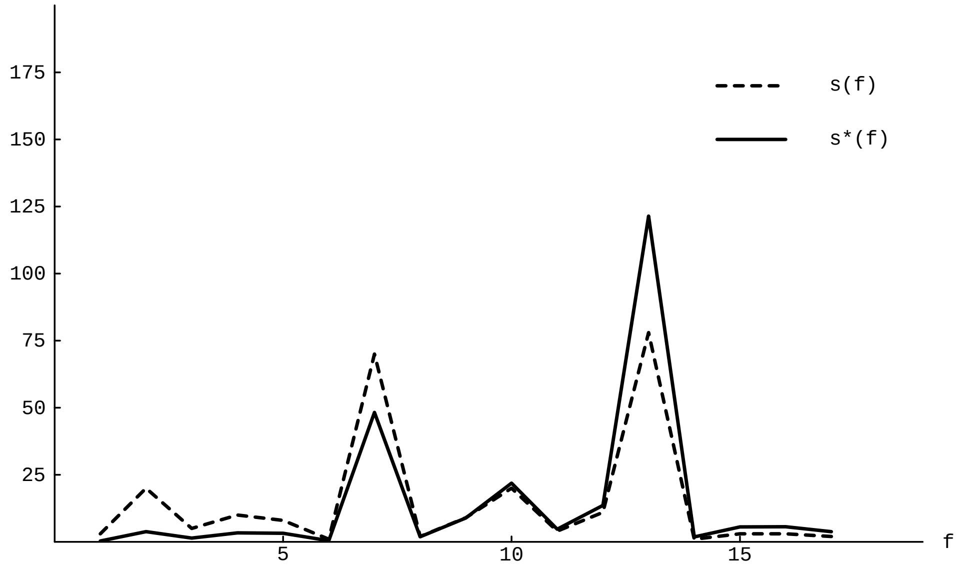


Figure 4.8: *The initial fitness distribution (dashed) and the resulting expected fitness distribution after applying linear ranking selection with* $\eta^- = 0.1$.

$$\begin{aligned} &= \eta^- \int_0^{\bar{S}(f)} dx + 2\frac{1-\eta^-}{N} \int_0^{\bar{S}(f)} x \, dx \\ &= \eta^- \bar{S}(f) + \frac{1-\eta^-}{N} \bar{S}(f)^2 \end{aligned}$$

As $\bar{s}^*(f) = \frac{d\bar{S}^*(f)}{df}$, (4.49) follows. $\square$

4.5.1 Reproduction Rate

Corollary 4.5 *The reproduction rate of ranking selection is*

$$\bar{R}_R(f) = \eta^- + 2\frac{1-\eta^-}{N}\bar{S}(f) \tag{4.50}$$

This is directly obtained by substituting (4.49) in (4.10).

Equation (4.50) shows that the worst individuals have the lowest reproduction rate $\bar{R}(f_0) = \eta^-$ and the best individuals have the highest reproduction rate $\bar{R}(f_n) = 2 - \eta^- = \eta^+$. This can be derived from the construction of the method as $\frac{\eta^-}{N}$ is the selection probability of the worst individual and $\frac{\eta^+}{N}$ the one of the best individual.

4.5.2 Loss of Diversity

Theorem 4.15 *The loss of diversity $D_R(\eta^-)$ of ranking selection is*

$$D_R(\eta^-) = (1 - \eta^-)\frac{1}{4} \tag{4.51}$$

Proof: Using Theorem 4.2 and realizing that $S(f_z) = \frac{N}{2}$ it follows that:

$$\begin{aligned}
D_R(\eta^-) &= \frac{1}{N}\left(\bar{S}(f_z) - \bar{S}^*(f_z)\right) \\
&= \frac{1}{N}\left(\bar{S}(f_z) - \eta^- \bar{S}(f_z) - \frac{1-\eta^-}{N}\bar{S}(f_z)^2\right) \\
&= \frac{1}{N}\left(\frac{N}{2} - \eta^- \frac{N}{2} - \frac{1-\eta^-}{N}\frac{N^2}{4}\right) \\
&= \frac{1}{4}(1-\eta^-)
\end{aligned}$$

□

Baker has derived this result using his term of “reproduction rate” [Baker, 1989].

Note that the loss of diversity is again independent of the initial distribution. The loss of diversity is depicted in Fig. 4.9.

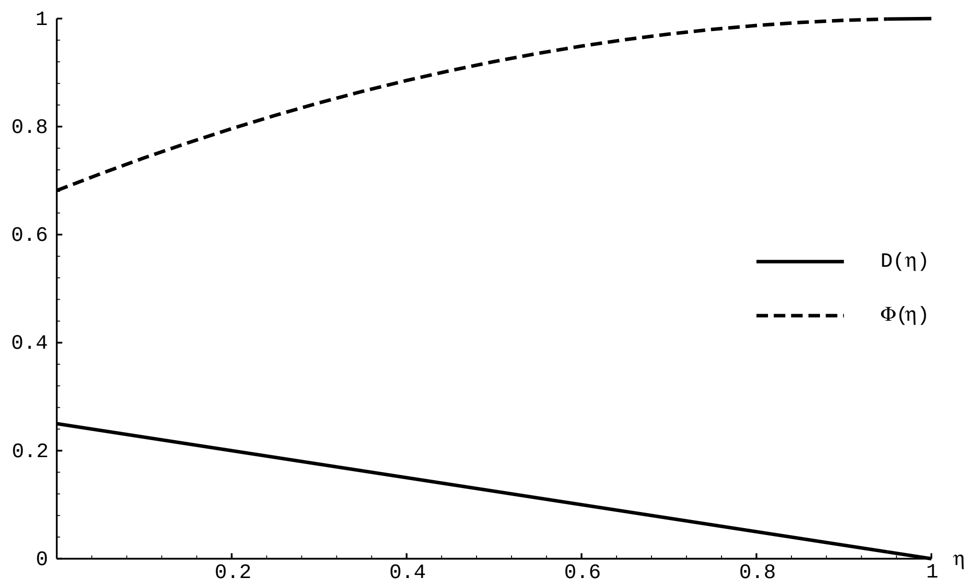


Figure 4.9: *The loss of diversity $D_R(\eta)$ (solid line) and the selection variance $\Phi_R(\eta)$ (broken line) of linear ranking selection.*

4.5.3 Selection Intensity

Theorem 4.16 *The selection intensity of ranking selection is*

$$I_R(\eta^-) = (1-\eta^-)\frac{1}{\sqrt{\pi}} \tag{4.52}$$

Proof: Using the definition of the selection intensity (Definition 4.10) and the Gaussian function for the initial fitness distribution one obtains

$$\begin{aligned} I_R(\eta^-) &= \int_{-\infty}^{\infty} x \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \left(\eta^- + 2(1-\eta^-) \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-\frac{y^2}{2}} dy \right) dx \\ &= \frac{\eta^-}{\sqrt{2\pi}} \int_{-\infty}^{\infty} x e^{-\frac{x^2}{2}}\, dx + \frac{1-\eta^-}{\pi} \int_{-\infty}^{\infty} x e^{-\frac{x^2}{2}} \int_{-\infty}^{x} e^{-\frac{y^2}{2}} dy\, dx \end{aligned}$$

Using (D.2) and (D.4) (Appendix D), (4.52) follows. □

The selection intensity of ranking selection is shown in Figure 4.10 in dependence of the parameter η^-.

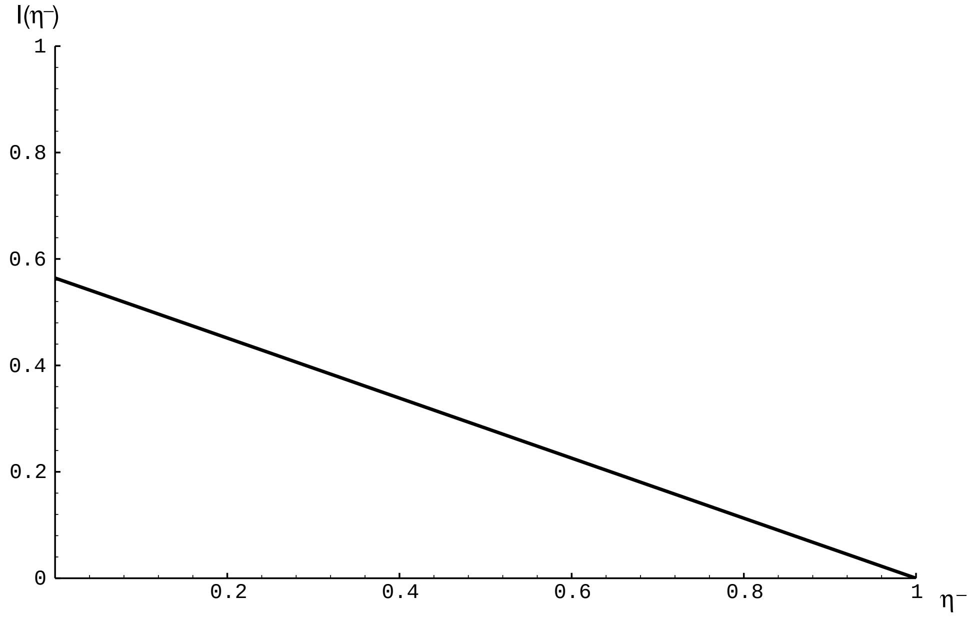


Figure 4.10: *The selection intensity $I_R(\eta^-)$ of ranking selection.*

4.5.4 Selection Variance

Theorem 4.17 *The selection variance of ranking is*

$$\Phi_R(\eta^-) = 1 - \frac{(1-\eta^-)^2}{\pi} = 1 - I_R(\eta^-)^2 \tag{4.53}$$

Proof: Substituting (4.49) into the definition equation (4.16) leads to

$$\Phi_R(\eta^-) \quad = \quad \int_{-\infty}^{\infty} f^2 \frac{1}{\sqrt{2\pi}} e^{-\frac{f^2}{2}} \left(\eta^- + 2(1-\eta-) \int_{-\infty}^{f} \frac{1}{\sqrt{2\pi}} e^{-\frac{y^2}{2}} dy \right) df$$

$$
\begin{aligned}
& -I_R(\eta^-)^2 \\
= \quad & \frac{\eta^-}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f^2 e^{-\frac{f^2}{2}}\, df \\
+ \quad & \frac{1-\eta^-}{\pi} \int_{-\infty}^{\infty} f^2 e^{-\frac{f^2}{2}} \int_{-\infty}^{f} e^{-\frac{y^2}{2}}\, dy\, df \\
- \quad & I_R(\eta^-)^2
\end{aligned}
$$

Using the relations (D.5) and (D.6) leads to

$$
\begin{aligned}
\Phi_R(\eta^-) &= \eta^- + (1-\eta^-) - I_R(\eta^-)^2 \\
&= 1 - I_R(\eta^-)^2
\end{aligned}
$$

□

The selection variance of ranking selection is plotted in Fig. 4.9.

4.6 Exponential Ranking Selection

Exponential ranking selection differs from linear ranking selection in that the probabilities of the ranked individuals are exponentially weighted. The base of the exponent is the parameter $0 < c < 1$ of the method. The closer c to 1 the lower is the "exponentiality" of the selection method. The meaning and the influence of this parameter will be discussed in detail. Again the rank N is assigned to the best individual and the rank 1 to the worst individual. Hence the probabilities of the individuals are given by

$$
p_i = \frac{c^{N-i}}{\sum_{j=1}^{N} c^{N-j}}\,; \qquad i \in \{1, ..., N\} \tag{4.54}
$$

The sum $\sum_{j=1}^{N} c^{N-j}$ normalizes the probabilities to ensure

$\sum_{i=1}^{N} p_i = 1$. As $\sum_{j=1}^{N} c^{N-j} = \frac{c^N-1}{c-1}$ the above equation can be rewritten:

$$p_i = \frac{c-1}{c^N-1} c^{N-i} \,; \qquad i \in \{1, ..., N\} \tag{4.55}$$

The algorithm for exponential ranking (Algorithm 10) is similar to the algorithm for linear ranking. The only difference lies in the calculation of the selection probabilities.

Theorem 4.18 *The expected fitness distribution after performing exponential ranking selection with parameter c on the distribution s is*

$$\Omega_E^*(s, c, N)(f_i) = s^*(f_i) = N \frac{c^N}{c^N-1} c^{-S(f_i)} \left(c^{s(f_i)} - 1\right) \tag{4.56}$$

Algorithm 10: (Exponential Ranking Selection)

Input: The population $J = (J_1, \ldots, J_N)$,
the fitness values of the population $\vec{\phi} = (\phi_1, \ldots, \phi_N)$,
the ranking base $c \in]0, 1]$
Output: The population after selection $\vec{J'} = (J'_1, \ldots, J'_N)$

```
exponential_ranking(c, J⃗, φ⃗):
    J⃗* ← sorted population J⃗ according to fitness
        with least fit individual at the first position
    sum_0 ← 0
    for i ← 1 to N do
        sum_i ← sum_{i-1} + p_i (Equation 4.55)
    od
    for i ← 1 to N do
        r ← random[0,1[
        J'_i ← J*_l such that sum_{l-1} ≤ r < sum_l
    od
    return J⃗'
```

Proof: First the expected number of individuals with fitness f_i or worse is calculated, i.e. $S^*(f_i)$. As the individuals are sorted according to their fitness value this number is given by the sum of the probabilities of the $S^*(f_i)$ least fit individuals:

$$\begin{aligned} S^*(f_i) &= N \sum_{j=1}^{S(f_i)} p_j \\ &= N \frac{c-1}{c^N-1} \sum_{j=1}^{S(f_i)} c^{N-j} \end{aligned}$$

and with the substitution $k = N - j$

$$\begin{aligned} S^*(f_i) &= N \frac{c-1}{c^N-1} \sum_{k=N-S(f_i)}^{N-1} c^k \\ &= N \frac{c-1}{c^N-1} \left(\sum_{k=0}^{N-1} c^k - \sum_{k=0}^{N-S(f_i)-1} c^k \right) \\ &= N \frac{c-1}{c^N-1} \left(\frac{c^N-1}{c-1} - \frac{c^{N-S(f_i)}}{c-1} \right) \\ &= N \left(1 - \frac{c^N}{c^N-1} c^{-S(f_i)} \right) \end{aligned}$$

As $s^*(f_i) = S^*(f_i) - S^*(f_{i-1})$ it follows that

$$\begin{aligned} s^*(f_i) &= N \frac{c^N}{c^N-1} \left(c^{-S(f_{i-1})} - c^{-S(f_i)} \right) \\ &= N \frac{c^N}{c^N-1} c^{-S(f_i)} \left(c^{s(f_i)} - 1 \right) \end{aligned}$$

□

Example 4.6 *Using the discrete fitness distribution from Example 4.1 the fitness distribution shown in Figure 4.11 is obtained after applying exponential ranking selection with $c = 0.986$ ($N = 250$).*

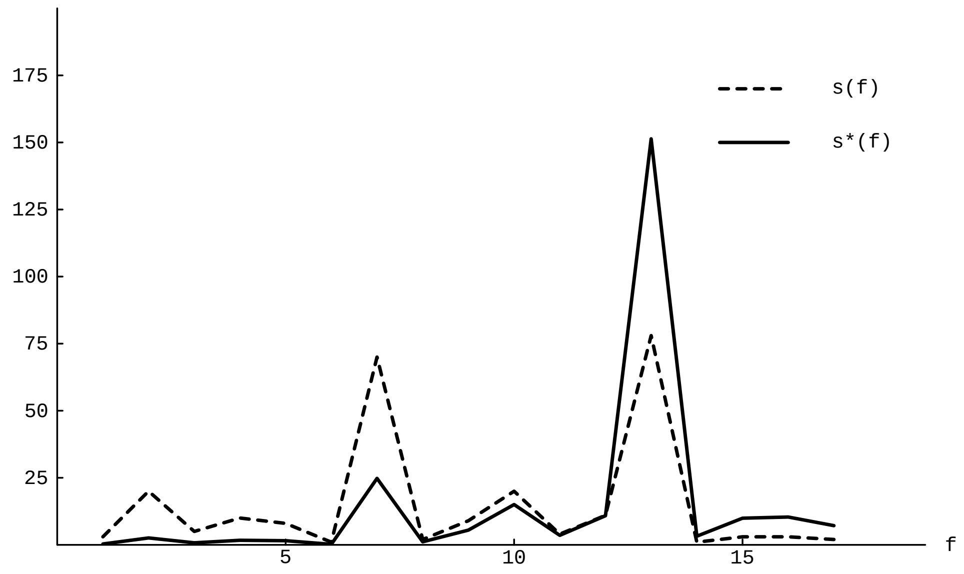


Figure 4.11: *The initial fitness distribution (dashed) and the resulting expected fitness distribution after applying exponential ranking selection with $c = 0.986$ ($N = 250$).*

Theorem 4.19 *Let $\bar{s}(f)$ be the continuous fitness distribution of the population. Then the expected fitness distribution after performing exponential ranking selection $\overline{\Omega}_E$ with parameter c on the distribution $\bar{s}$ is*

$$\overline{\Omega}^*_E(\bar{s}, c)(f) = \bar{s}^*(f) = N \frac{c^N}{c^N - 1} \ln c \; \bar{s}(f) \; c^{-\bar{S}(f)} \tag{4.57}$$

Proof: As the continuous form of (4.55) is given by $\bar{p}(x) = \frac{c^{N-x}}{\int_0^N c^{N-x}}$ and $\int c^x = \frac{1}{\ln c} c^x$ one calculates:

$$\begin{aligned}
\bar{S}^*(f) &= N \frac{c^N \ln c}{c^N - 1} \int_0^{\bar{S}(f)} c^{-x} \, dx \\
&= -N \frac{c^N}{c^N - 1} [c^{-x}]_0^{\bar{S}(f)}
\end{aligned}$$

$$= \quad -N\frac{c^N}{c^N-1}\left(1+c^{-S(f)}\right)$$

As $\bar{s}^*(f) = \frac{d\bar{S}^*(f)}{df}$, (4.57) follows. □

It is useful to introduce a new variable $\kappa = c^N$ to eliminate the explicit dependence on the population size N:

$$\overline{\Omega}^*_E(s,\kappa)(f) = \bar{s}^*(f) = \frac{\kappa\ln\kappa}{\kappa-1}\bar{s}(f)\kappa^{-\frac{\bar{S}(f)}{N}} \tag{4.58}$$

The meaning of κ will become apparent in the next section.

4.6.1 Reproduction Rate

Corollary 4.6 *The reproduction rate of exponential ranking selection is*

$$\bar{R}_E(f) = \frac{\kappa\ln\kappa}{\kappa-1}\kappa^{-\frac{\bar{S}(f)}{N}} \tag{4.59}$$

This is directly obtained by substituting (4.58) in (4.10).

Equation (4.59) shows that the worst individual has the lowest reproduction rate $\bar{R}(f_0) = \frac{\kappa\ln\kappa}{\kappa-1}$ and the best individual has the highest reproduction rate $\bar{R}(f_n) = \frac{\ln\kappa}{\kappa-1}$. Hence a natural explanation of the variable κ is obtained, as $\frac{\bar{R}(f_0)}{\bar{R}(f_n)} = \kappa$: it describes the ratio of the reproduction rate of the worst and the best individual. Note that $c < 1$ and hence $c^N \ll 1$ for large N, i.e., the interesting region of values for κ is in the range from $[10^{-20}, 1]$.

4.6.2 Loss of Diversity

Theorem 4.20 *The loss of diversity $D_E(\kappa)$ of exponential ranking selection is*

$$D_E(\kappa) = \frac{1 - \ln \frac{\kappa-1}{\kappa \ln \kappa}}{\ln \kappa} - \frac{\kappa}{\kappa - 1} \tag{4.60}$$

Proof: From the demand $R(f_z) = 1$ one calculates:

$$\frac{\bar{S}(f_z)}{N} = -\frac{\ln \frac{\kappa-1}{\kappa \ln \kappa}}{\ln \kappa} \tag{4.61}$$

Using Theorem 4.2 leads to:

$$\begin{aligned}
D_E(\kappa) &= \frac{1}{N}\left(\bar{S}(f_z) - \bar{S}^*(f_z)\right) \\
&= -\frac{\ln \frac{\kappa-1}{\kappa \ln \kappa}}{\ln \kappa} - \frac{\kappa}{\kappa - 1}\left(1 - \kappa^{\frac{\ln \frac{\kappa-1}{\kappa \ln \kappa}}{\ln \kappa}}\right) \\
&= -\frac{\ln \frac{\kappa-1}{\kappa \ln \kappa}}{\ln \kappa} - \frac{\kappa}{\kappa - 1}\left(1 - \frac{\kappa - 1}{\kappa \ln \kappa}\right) \\
&= \frac{1 - \ln \frac{\kappa-1}{\kappa \ln \kappa}}{\ln \kappa} - \frac{\kappa}{\kappa - 1}
\end{aligned}$$

□

The loss of diversity is shown in Fig. 4.12.

4.6.3 Selection Intensity and Selection Variance

Analytical expressions of the selection intensity and the selection variance are difficult to calculate for exponential ranking. Recalling the definition of the selection intensity (Definition 4.10) one sees that the integral of the Gaussian function occurs as exponent in an indefinite integral. Hence the selection intensity as well as the selection variance will be only numerically calculated. The selection intensity is shown in Fig. 4.13, the selection variance in Fig. 4.12.

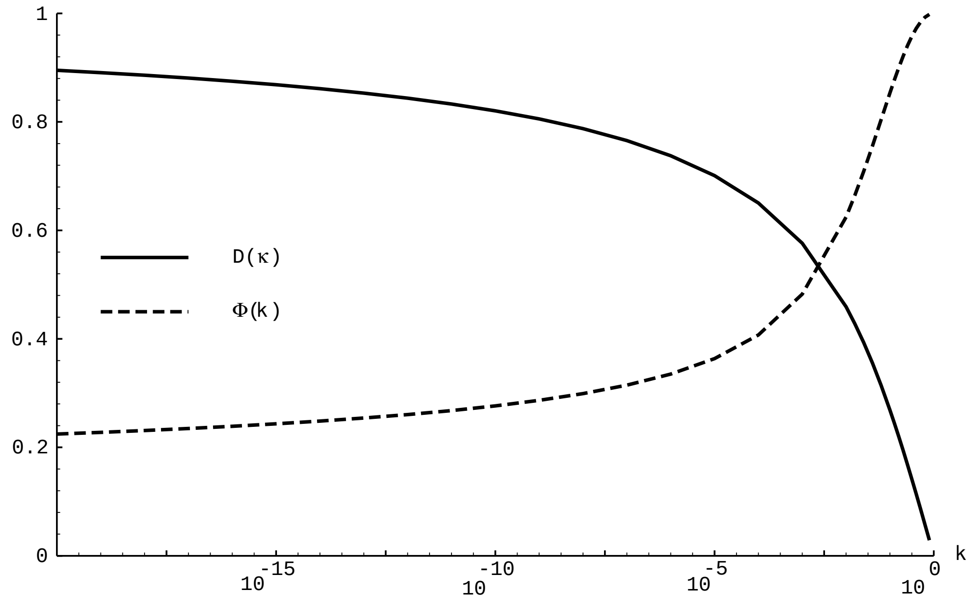


Figure 4.12: *The loss of diversity $D_E(\kappa)$ (solid line) and the selection variance $\Phi_E(\kappa)$ (broken line) of exponential ranking selection. Note the logarithmic scale of the κ-axis.*

However, an approximation formula can be derived using the Genetic Programming optimization method (see Appendix B). The selection intensity of exponential ranking selection can be approximated with a relative error of less than 1.81% for $\kappa \in [10^{-20}, 0.8]$ by

$$I_E(\kappa) \approx \frac{\ln \kappa}{-2.548 - 1.086\sqrt{\kappa} + 0.4028 \ln \kappa} \tag{4.62}$$

Similar, an approximation for the selection variance of exponential ranking selection can be found. The following formula approximates the selection variance with an relative error of less than 2.8% for $\kappa \in [10^{-20}, 0.8]$:

$$\Phi_E(\kappa) \approx 0.1 + \kappa^{\frac{1}{4}}(e^{\frac{-\kappa}{2}} + 0.2428\kappa^{-\frac{15}{64}}) \tag{4.63}$$

The selection variance is shown in Fig. 4.12.

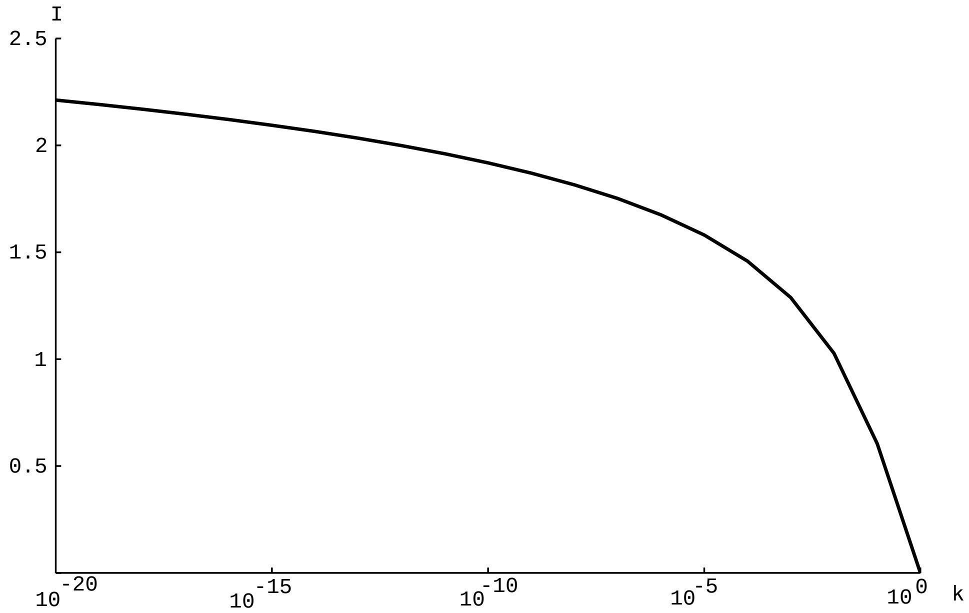


Figure 4.13: *The selection intensity $I_E(\kappa)$ of exponential ranking selection. Note the logarithmic scale of the κ-axis.*

4.7 Comparison of Selection Schemes

In this section the selection methods are compared according to their properties derived in the preceding sections. First the reproduction rates of selection methods are compared and an unified view of selection schemes is derived. Section 4.7.3 is devoted to the comparison of the selection intensity and gives a convergence prediction for simple Genetic Algorithm optimizing the ONEMAX function. The selection intensity is also used in the subsequent sections to compare the methods according to their loss of diversity and selection variance.

4.7.1 Reproduction Rate and Universal Selection

The reproduction rate gives the number of expected offspring of an individual with a certain fitness value after selection. In the preceding sections only the reproduction rate for the continuous case has been considered. Table 4.1 gives the equations for the discrete (exact) case. They have been derived using the exact offspring equations (4.21), (4.38), (4.48), (4.56) and (4.18) with some additional algebraic manipulations.

Selection Method Ω	Reproduction Rate $R_\Omega(f_i)$
Tournament	$\frac{N}{s(f_i)}\left(\left(\frac{S(f_i)}{N}\right)^t - \left(\frac{S(f_{i-1})}{N}\right)^t\right)$
Truncation	$\begin{cases} 0 & : \; S(f_i) \leq (1-T)N \\ \frac{S(f_i)-(1-T)N}{s(f_i)T} & : \; S(f_{i-1}) \leq (1-T)N < S(f_i) \\ \frac{1}{T} & : \; else \end{cases}$
Linear Ranking	$\frac{N\eta^- - 1}{N-1} + \frac{1-\eta^-}{N-1}\left(2S(f_i) - s(f_i)\right)$
Exp. Ranking	$\frac{N}{s(f_i)}\frac{\kappa}{\kappa-1}\kappa^{\frac{-S(f_i)}{N}}\left(\kappa^{\frac{s(f_i)}{N}} - 1\right)$
Proportional	$\frac{f_i}{M}$

Table 4.1: *Comparison of the reproduction rate of the selection methods for discrete distributions.*

The standard sampling mechanism uses one spin of a roulette wheel (divided into segments for each individual with an the segment size proportional to the reproduction rate) to determine one member of the next generation. Hence, N trials have to be performed to obtain an entire population. As these trials are independent of each other a relatively high mean variation in the outcome is observed (see also Chapter 4.1 and Theorem 4.1). This is also the case for tournament selection although there is no explicitly used roulette wheel sampling.

This mean variation can be almost completely eliminated by using the reproduction rate and the so called "stochastic universal sampling" (SUS) method [Baker, 1987]. In this method only a single spin of the wheel is necessary as the roulette has N markers for the "winning individuals" and hence all individuals are chosen at once.

The SUS algorithm can be regarded as an optimal sampling algorithm. It has zero bias, i.e., no deviation between the expected reproduction rate and the algorithmic sampling frequency. Furthermore, SUS has minimal spread, i.e., the range of the possible values for $s'(f_i)$ is

$$s'(f_i) \in \{\lfloor s^*(f_i) \rfloor, \lceil s^*(f_i) \rceil\} \tag{4.64}$$

The outline of the SUS algorithm is given by Algorithm 11.

Algorithm 11: (Stochastic Universal Sampling)

Input: The population $\vec{J}$ and the reproduction rate for each fitness value $\vec{R} = (R_1, \ldots, R_n)$
Output: The population after selection $\vec{J}'$

SUS($\vec{J}, \vec{R}$):
  $sum \leftarrow 0$
  $j \leftarrow 1$
  $ptr \leftarrow$ random[0,1)
  for $i \leftarrow 1$ **to** N **do**
    $sum \leftarrow sum + R_k$ where R_k is the reproduction rate of individual J_i
    while ($sum > ptr$) **do**
      $J'_j \leftarrow J_i$
      $j \leftarrow j + 1$
      $ptr \leftarrow ptr + 1$
    od
  od
  return $\{\vec{J}'\}$

By means of the SUS algorithm the outcome of a certain run of the selection scheme is as close as possible to the expected behavior, i.e., the mean variation is minimal. Even though it is not clear whether there are any performance advantages in using SUS, it makes the run of a selection method more "predictable".

To be able to apply SUS one has to know the expected number of offspring of each individual. Baker has applied this sampling method only to linear ranking selection as here the expected number of offspring is known by construction (see Section 4.5). As the offspring values have been derived for the selection methods discussed in the previous sections (see Table 4.1) it is possible to use stochastic universal sampling for all these selections schemes. This leads to an unified view of selection schemes and allows the construction of an "universal selection method" in the following way: First the fitness distribution of the population is computed. Next the expected reproduction rates are calculated using the equations derived in the proceeding sections (Table 4.1). In the last step SUS is used to obtain the new population after selection (Algorithm 12).

Algorithm 12: (Universal Selection Method)

Input: The population $\vec{J} = (J_1, \ldots, J_N)$,
the fitness values of the population $\vec{\phi} = (\phi_1, \ldots, \phi_N)$.
Output: The population after selection $\vec{J}'$

universal_selection($\vec{J}, \vec{\phi}$):
$\vec{s} \leftarrow$ fitness_distribution($\vec{\phi}$)
$\vec{r} \leftarrow$ reproduction_rate($\vec{s}$)
$\vec{J}' \leftarrow$ SUS($\vec{J}, \vec{r}$)
return $\vec{J}'$

The time complexity of the universal selection method is $\mathcal{O}(N \ln N)$ as the fitness distribution has to be computed. Hence, if "tournament selection" is performed with this algorithm the lower mean variation is paid with a higher computational complexity.

4.7.2 The Complement Selection Schemes: Tournament and Linear Ranking

If several properties of tournament selection and linear ranking selection are compared one observes that binary tournament behaves similar to a linear ranking selection with a very small η^-. And indeed it is possible to prove that binary tournament and linear ranking with $\eta^- = \frac{1}{N}$ have identical expected behavior.

Theorem 4.21 *The expected fitness distributions of linear ranking selection with $\eta^- = \frac{1}{N}$ and tournament selection with $t = 2$ are identical, i.e.*

$$\Omega_R^*(s, \frac{1}{N}) = \Omega_T^*(s, 2) \tag{4.65}$$

Proof:

$$\begin{aligned} \Omega_R^*(s, \frac{1}{N})(f_i) &= s(f_i)\frac{N\frac{1}{N} - 1}{N-1} + \frac{1 - \frac{1}{N}}{N-1}\left(S(f_i)^2 - S(f_{i-1})^2\right) \\ &= \frac{1}{N}\left(S(f_i)^2 - S(f_{i-1})^2\right) \\ &= \Omega_T^*(s, 2)(f_i) \end{aligned}$$

□

Goldberg and Deb [Goldberg and Deb, 1991] have also shown this result, but only for the behavior of the best fit individual.

By this the complementary character of the two selection schemes is revealed. For lower selection intensities ($I \leq \frac{1}{\sqrt{\pi}}$) linear ranking selection is the appropriate selection mechanism as for selection intensities ($I \geq \frac{1}{\sqrt{\pi}}$) tournament selection is better suited. At the border the two section schemes are identical.

4.7.3 Comparison of the Selection Intensity

Selection Method	Selection Intensity
Tournament	$I_T(t) \approx \sqrt{2(\ln t - \ln(\sqrt{4.14\ \ln t}))}$
Truncation	$I_\Gamma(T) = \frac{1}{T}\frac{1}{\sqrt{2\pi}}e^{-\frac{f_c^2}{2}}$
Linear Ranking	$I_R(\eta^-) = (1-\eta^-)\frac{1}{\sqrt{\pi}}$
Exponential Ranking	$I_E(\kappa) \approx \frac{\ln\kappa}{-2.548-1.086\sqrt{\kappa}+0.4028\ln\kappa}$
Fitness Proportionate	$I_P = \frac{\bar{\sigma}}{\bar{M}}$

Table 4.2: *Comparison of the selection intensity of the selection methods.*

As the selection intensity is a very important property of the selection method, Table 4.3 gives some settings for the selection methods that yield the same selection intensity.

The importance of the selection intensity is based on the fact that the behavior of a simple Genetic Algorithm can be predicted if the fitness distribution is normally distributed. In [Mühlenbein and Schlierkamp-Voosen, 1993] a prediction is made for a genetic algorithm optimizing the ONEMAX (or bit-counting) function. Here the fitness is given by the number of 1's in the binary string of length l. Uniform crossover is used and assumed to be random process which creates a binomial fitness distribution. As a result, after each recombination phase the input of the next selection phase approximates a Gaussian distribution. Hence, a prediction of this optimization using the selection intensity should be possible. For a sufficiently large population the authors calculate

$$p(k) = \frac{1}{2}\left(1 + \sin(\frac{I}{\sqrt{l}}k + \arcsin(2p_0 - 1))\right) \tag{4.66}$$

where p_0 denotes the fraction of 1's in the initial random population and $p(k)$ the fraction of 1's in generation k. Convergence is characterized by the fact that $p(k_c) = 1$ so the convergence time for the special

I	0.34	0.56	0.84	1.03	1.16
Ω_T:t	-	2	3	4	5
Ω_R:η^-	0.4	0	-	-	-
Ω_Γ:T	0.8	0.66	0.47	0.36	0.30
Ω_E:κ	0.29	0.12	0.032	$9.8 \cdot 10^{-3}$	$3.5 \cdot 10^{-3}$

I	1.35	1.54	1.87	2.16
Ω_T:t	7	10	20	40
Ω_Γ:T	0.22	0.15	0.08	0.04
Ω_E:κ	$4.7 \cdot 10^{-4}$	$2.5 \cdot 10^{-5}$	10^{-9}	$2.4 \cdot 10^{-18}$

Table 4.3: *Parameter settings for truncation selection Ω_Γ, tournament selection Ω_T, linear ranking selection Ω_R, and exponential ranking selection Ω_E to achieve the same selection intensity I.*

case of $p_0 = 0.5$ is given by $k_c = \frac{\pi}{2}\frac{\sqrt{n}}{I}$. Mühlenbein derived this formula for truncation selection, however only the selection intensity is used. Thereby it is straightforward to give the convergence time for any other selection method, by substituting I with the corresponding terms derived in the preceding sections.

For tournament selection one calculates

$$k_{T,c}(t) \approx \frac{\pi}{2}\sqrt{\frac{l}{2(\ln t - \ln\sqrt{4.14\ln t})}} \tag{4.67}$$

for truncation selection

$$k_{\Gamma,c}(T) = T\frac{\pi\sqrt{\pi l}}{\sqrt{2}}e^{\frac{f_c^2}{2}} \tag{4.68}$$

for linear ranking selection

$$k_{R,c}(\eta^-) = \frac{\pi\sqrt{\pi l}}{2(1-\eta^-)} \tag{4.69}$$

and for exponential ranking selection

$$k_{E,c}(\kappa) \approx \frac{\pi\sqrt{l}}{2}\,\frac{-2.548 - 1.086\sqrt{\kappa} + 0.4028\ln\kappa}{\ln\kappa} \tag{4.70}$$

4.7.4 Comparison of Loss of Diversity

Selection Method	Loss of Diversity
Tournament	$D_T(t) = t^{-\frac{1}{t-1}} - t^{-\frac{t}{t-1}}$
Truncation	$D_\Gamma(T) = 1 - T$
Linear Ranking	$D_R(\eta^-) = (1-\eta^-)\frac{1}{4}$
Exponential Ranking	$D_E(\kappa) = \frac{1-\ln\frac{\kappa-1}{\kappa\ln\kappa}}{\ln\kappa} - \frac{\kappa}{\kappa-1}$

Table 4.4: *Comparison of the loss of diversity of the selection methods*

Table 4.4 summarizes the loss of diversity for the selection methods. It is difficult to compare these relations directly as they depend on different parameters that are characteristic for the specific selection method, e.g., the tournament size t for tournament selection, the threshold T for truncation selection, etc. Hence, one has to look for an independent measure to eliminate these parameters and to be able to compare the loss of diversity. This measure is chosen to be the selection intensity. The loss of diversity of the selection methods is viewed as a function of the selection intensity. To calculate the corresponding graph one first computes the value of the parameter of a selection method (i.e., t for tournament selection, T for truncation selection, η^- for linear ranking selection, and κ for exponential ranking selection) that is necessary to achieve a certain selection intensity. With this

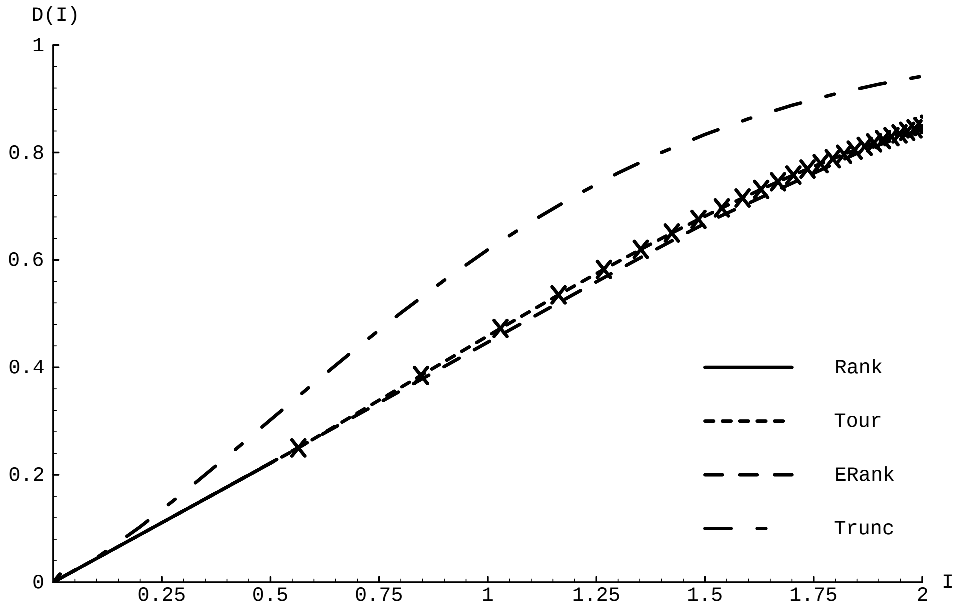


Figure 4.14: *The dependence of the loss of diversity D on the selection intensity I for tournament selection, truncation selection, linear ranking selection and exponential ranking selection. Note that for tournament selection only the points marked as crosses on the graph correspond to valid (integer) tournament sizes.*

value the loss of diversity is then obtained using the corresponding equations, i.e., (4.31), (4.41), (4.51), (4.60). Figure 4.14 shows the result of this comparison: the loss of diversity for the different selection schemes in dependence of the selection intensity. Tournament and exponential ranking behave almost identical. Ranking selection only covers the lower quarter of the I-axis, as the maximum selection intensity of ranking selection is $I_R(0) = \frac{1}{\sqrt{\pi}} \approx 0.56$. In this region it behaves also very similar to exponential ranking. To achieve the same selection intensity more bad individuals are replaced using truncation selection than using tournament selection or one of the ranking selection schemes, respectively. This means that more "genetic material" is lost using truncation selection.

The graph in Fig. 4.14 is based on the standardized selection intensity as given by Definition 4.10, i.e., a normal distribution of the initial fitness values has been assumed. In order to estimate the generality of the above results, the following experiment has been set up: 800 different fitness distribution have been randomly created and used as input to tournament selection and truncation selection. Eight different parameter settings for the selection methods have been used (corresponding to the first eight entries in Table 4.3). The *realized selection intensity* $\hat{I}$ and the loss of diversity have been measured for each fitness distribution. The realized selection intensity has been computed according to the general definition of selection intensity (Def. 4.9), i.e.,

$$\hat{I} = \frac{M^* - M}{\sqrt{\sum_{i=1}^{n} s(f_i)(f_i - M)^2}} \tag{4.71}$$

The difference between the loss of diversity of both methods for a given selection intensity is shown in Fig. 4.15. The solid line is the theoretical expected difference according to the standardized selection intensity (as in Fig. 4.14). The dots are the measured differences of the loss of diversity. As can be seen, in almost all cases this difference is positive, i.e., truncation selection has a higher loss of diversity as tournament selection *for arbitrary fitness distributions.* Furthermore, the computed loss of diversity seems to be an accurate approximation and upper bound for this difference.

Supposing that a lower loss of diversity is desirable as it reduces the risk of premature convergence, one expects that truncation selection should be outperformed by the other selection methods. But in general it depends on the problem and on the representation of the problem to be solved whether a low loss of diversity is "advantageous". But with Fig. 4.14 one has a useful tool at hand to make the right decision for a particular problem.

Another interesting fact can be observed looking at Table 4.4: The loss of diversity is independent of the initial fitness distribution. Nowhere in the derivation of these equations a particular fitness distribution is assumed and nowhere the fitness distribution $\bar{s}(f)$ occurs in the equations. In contrary, the (standardized) selection intensity and

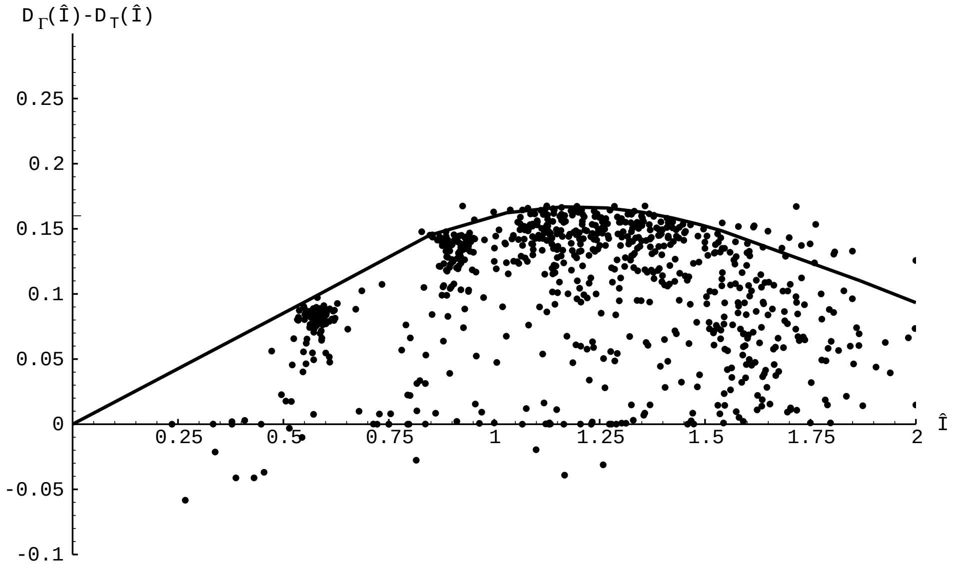


Figure 4.15: *The measured difference of the loss of diversity of truncations selection and tournament selection in dependence of the realized selection intensity. The solid line gives the difference according to the standardized selection intensity (as displayed in Fig. 4.14), the points give the differences measured for 800 randomly created fitness distributions.*

the (standardized) selection variance are computed for a certain initial fitness distribution (the normalized Gaussian distribution). Hence, the loss of diversity can be viewed as an inherent property of a selection method.

4.7.5 Comparison of the Selection Variance

The mechanism used in the preceding subsection is now applied to the selection variance, i.e., the selection variance is viewed as a function of the selection intensity.

Selection Method	Selection Variance
Tournament	$\Phi_T(t) \approx \frac{0.918}{\ln(1.186+1.328t)}$
Truncation	$\Phi_\Gamma(T) = 1 - I_\Gamma(T)(I_\Gamma(T) - f_c)$
Linear Ranking	$\Phi_R(\eta^-) = 1 - I_R^2(\eta^-)$
Exponential Ranking	$\Phi_E(\kappa) \approx 0.1 + \kappa^{\frac{1}{4}}(e^{\frac{-\kappa}{2}} + 0.2428\kappa^{-\frac{15}{64}})$

Table 4.5: *Comparison of the selection variance of the selection methods.*

It can be seen in Figure 4.16 that truncation selection leads to a lower selection variance than tournament selection. The highest selection variance is obtained by exponential ranking.

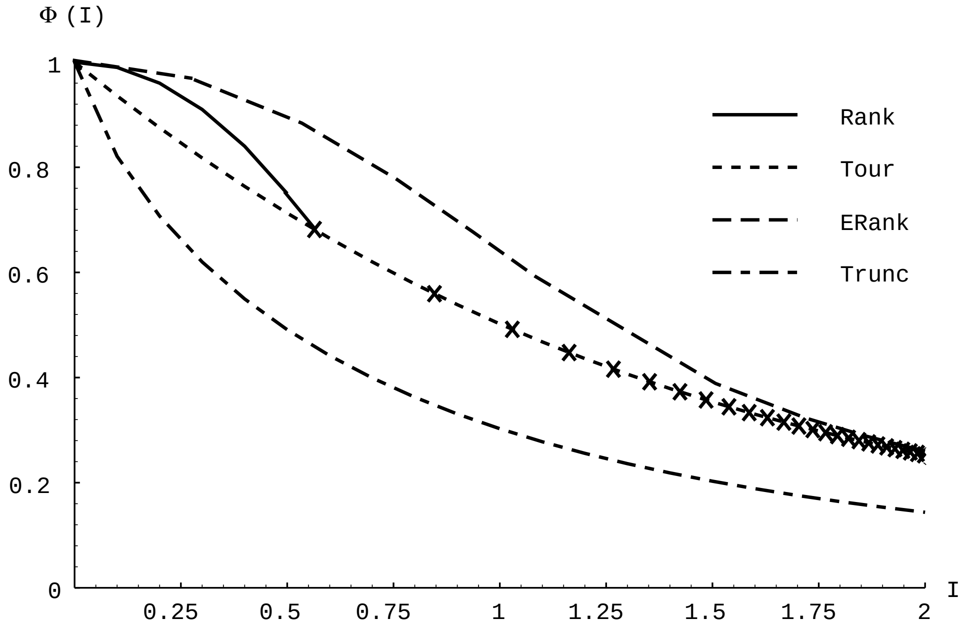


Figure 4.16: *The dependence of the selection variance Φ on the selection intensity I for the selection schemes. Note that for tournament selection only the points marked as crosses on the graph correspond to valid (integer) tournament sizes.*

An interpretation of the results may be difficult as it depends on the optimization task and the kind of problem to be solved whether a high selection variance is advantageous or not. But again this graph may help decide the “appropriate” selection method for a particular optimization problem. Under the assumption that a higher variance is advantageous to the optimization process, exponential ranking selection reveals itself to be the best selection scheme. In [Mühlenbein and Voigt, 1995] it is stated that

> “if two selection methods have the same selection intensity, the method giving the higher standard deviation of the selected parents is to be preferred”[Mühlenbein and Voigt, 1995].

Although this statement is too general, for particular problems this might be true and hence exponential ranking selection would be the “best” selection method under this premise.

Chapter 5

Recombination

In the preceding chapter a detailed analysis of several selection schemes was carried out. This chapter deals with the recombination method in Evolutionary Algorithms. As mentioned in Chapter 3 the various subclasses of EAs differ in the representation of the individuals and therefore in the implementation of the recombination mechanism. Hence, no global analysis of recombination is possible and one has to investigate the recombination mechanism individually for each representation. Much research has be done to analyze and predict the performance of Genetic Algorithms and Evolution Strategies. An excellent overview of this work can be found for example in [Bäck, 1994a].

In the remainder of this chapter the recombination in Genetic Programming is analyzed. The focus is on the crossover operator, as it is commonly used exclusively in Genetic Programming and no mutation is applied. First, a short description of the crossover operator is given. Section 5.2 describes the well-known problem of "bloating", i.e., the fact that the trees grow over the generations without improving the solution. Furthermore, several approaches to explain this phenomenon are introduced. In Section 5.3 a new approach to explain the cause of bloating is described. In Section 5.4 some experiments follow that compare the effectiveness of the different approaches on a discrete and a continuous problem.

5.1 Recombination in Genetic Programming

The overview of the variants of Evolutionary Algorithms in Chapter 3 introduced Genetic Programming and its peculiarity of tree-represented individuals.

The only recombination method that will be considered in this chapter is the random exchange of subtrees between two parents (see Fig. 3.4 on page 36). To form new offspring two trees are randomly chosen from the population (parents). In each tree an edge is randomly chosen as *crossover site* where the subtree attached is cut from the tree. The two subtrees are swapped and combined with the old trees at the crossover sites to form the children. In general, this results in two new trees, even if the two parents are identical.

Usually the crossover site is not chosen uniformly in the tree, but with a lower probability for the leaves and a higher probability for the nodes within the tree. This idea was introduced by Koza [Koza, 1992] as an uniform selection probability would lead to a frequent swapping of leaves.

5.2 Bloating

A crucial point of GP is the tree size explosion or bloating phenomenon. It is well known from several investigations that the tree size in GP tends grow even (or especially) if no improvement in the population is made [Tackett, 1994; Blickle and Thiele, 1994; Nordin and Banzhaf, 1995; Nordin *et al.*, 1995; Andre and Teller, 1996]. However, for many applications the tree size must be small in order to be able to interpret the evolved results. Furthermore, it has been observed that bloating is accompanied by a reduced performance of the optimization process [Blickle and Thiele, 1994; Andre and Teller, 1996].

There are several suggestions to tackle the problem of tree size explosion. The common approach is to introduce of some kind of parsimony pressure analogous to a penalty term. If the fitness is to be minimized this consists of adding a factor to the fitness value that is related to the size of the individual, i.e.,

$$f(J) = A(J) + \delta C(J) \tag{5.1}$$

where $A(J)$ gives the original value of the individual and $C(J)$ gives its complexity. The difficulty in this approach is the balancing of this parsimony pressure δ. Especially if the fitness function is real valued, one faces the problem that if the parsimony pressure is too high, the GP may favor a short solution to a better one.

5.2.1 Analysis Based on Introns

Nordin *et. al.* addressed the tree size problem in two studies [Nordin and Banzhaf, 1995; Nordin *et al.*, 1995].

In [Nordin and Banzhaf, 1995] a description of the crossover operator in GP is presented introducing the term "intron". An intron is defined as a "block that does not affect the behavior of the entire program for any of the fitness cases" [Nordin and Banzhaf, 1995]. The crossover effect in GP is described and it is observed that most crossovers performed in GP have a destructive effect, in that the fitness of the children decrease. Additionally, many crossover events are neutral, i.e., the fitness of the children remains the same as that of the parents. A mathematical analysis shows that an individual can protect itself against crossover by increasing its relative fraction of introns. The application of a parsimony pressure leads to an elimination of these introns and reveals a "short" solution. From this the authors argue that Evolutionary Algorithms have a built in property favoring short solutions and that a large fraction of the code of a solution in GP are "introns" and the effective size of the code is much smaller. Unfortunately, the authors have not solved the problem of an accurate parsimony pressure. The example demonstrating the reduction of introns from the individuals had a large parsimony pressure so that no solution to the problem was found.

This analysis of the crossover in GP is very similar to that performed by Blickle and Thiele in [Blickle and Thiele, 1994]. This approach will be presented in detail in Section 5.3. However, different conclusions are drawn from this analysis.

In the second publication ([Nordin *et al.*, 1995]) the term of Explicitly Defined Intron (EDI) is introduced:

> "An EDI is an instruction segment that is inserted between two nodes. It can, by definition, only act as an intron. It can *not* effect the calculated result of the individual. It does, however, effect the probability of crossover between the two nodes" [Nordin *et al.*, 1995].

A number is associated to each EDI representing the number of implicit introns that are simulated by this EDI. Hence, the EDI can be seen as a weight of the edges in the tree. The probability of crossover at a particular edge is proportional to the weight at that edge. During crossover the sum of the two weights involved in the crossover operation is randomly divided between the two new edges. The authors conclude that EDIs may replace and interact with implicit introns and under the presence of some parsimony pressure lead to more compact solutions. Another effect of the "weighted crossover" should be the potential emergence and encapsulation of useful building blocks possibly leading to a faster discover of high fit individuals. As an example they use a problem of symbolic regression on a second order polynomial with "large" constants.

5.2.2 Adaptive Parsimony Pressure

The problem of adjusting the parsimony pressure was addressed and elegantly solved by [Zhang and Mühlenbein, 1995]. They introduced a method called "adaptive balancing of accuracy and parsimony". The parsimony pressure δ (also called *Occam factor* in [Zhang and Mühlenbein, 1995]) is automatically adapted to the current quality of the best solution. It is interesting to note that this analysis does not require an analysis of the GP crossover operator. The proposed heuristic is derived from a detailed statistical analysis. The principle is a low parsimony in the beginning of the optimization, when the error $A(J)$ is above a user-specified accuracy limit ϵ. When this limit is reached, the parsimony pressure is increased in order to reduce the size of the solution.

The adaptive parsimony factor is used in the same way as in (5.1) and computed in each generation as follows.

$$\delta(k) = \begin{cases} \frac{1}{l^2}\frac{A_{best}(k-1)}{\hat{C}_{best}(k)} & : \text{ if } A_{best}(k-1) > \epsilon \\ \frac{1}{l^2}\frac{1}{A_{best}(k-1)\hat{C}_{best}(k)} & : \text{ else} \end{cases} \tag{5.2}$$

Here, l is the size of the training set for the particular problem, $A_{best}(k)$ is the raw fitness or accuracy of the current best individual in the population at generation k. $\hat{C}_{best}(k)$ is the estimated complexity of the best individual in the population in generation k. It is calculated on basis of the complexity $C_{best}(k-1)$ of the best individual of the previous generation:

$$\hat{C}_{best}(k) = C_{best}(k-1) + \Delta C_{sum}(k) \tag{5.3}$$

$\Delta C_{sum}(k)$ is a moving average that keeps track of the difference in the complexity between the best individual of one generation and the best individual of the next generation:

$$\Delta C_{sum}(k) = \frac{1}{2}\left(C_{best}(k) - C_{best}(k-1) + \Delta C_{sum}(k-1)\right) \tag{5.4}$$

A potential drawback of the method is the requirement of "global" information as the accuracy and the complexity of the best individual in the population is necessary. In a distributed Evolutionary Algorithm this imposes a global synchronization. Furthermore, this method will only work properly if the number of training cases l is sufficiently large.

The method is proposed in the context of evolving sigma-pi neural networks but can be used for arbitrary applications. The method seems to be very useful for many other applications. For example, in [Blickle, 1996] it enables short and accurate approximation formulas on complex data to be evolved. In Section 5.4 this approach is also compared and the effectiveness of the methodology is shown.

5.2.3 Trait Mining

A controversial approach is proposed by Tackett. He argues that

> "such strategies (using parsimony pressure) are sometimes beneficial but not consistently so, since they often interfere with population fitness" [Tackett, 1995]

and introduces a so called *trait-mining* approach that "determines saliency without intruding on the structure and process of the genetic-programming algorithm". The key idea is to introduce some mechanism to keep track of the frequency and saliency of subtrees (named "schema") [Tackett, 1994]. This information is used to select "better" subtrees for crossover to achieve a better convergence. However, this mechanism is associated with a very large computational overhead.

5.2.4 Individual Objective Switching

The idea of IOS introduced in Chapter 3.3 in connection with constraint-handling can also be used to address the problem of bloating. This can easily be seen if the problem of finding a short and accurate solution is stated as the following optimization problem:

$$\begin{gathered}\text{minimize } C(J),\\ \text{such that } A(J) < \epsilon\end{gathered}$$

In this case the objective is to minimize the tree size under the constraint that the approximation error is below a specified limit ϵ. Hence, the fitness function can be formulated as

$$f(J) = \begin{cases} 1 + A(J) & : \quad A(J) \geq \epsilon \\ \frac{C(J)}{C_{max}} & : \quad else \end{cases} \tag{5.5}$$

The great advantage of this approach is that no balancing of accuracy and size by the Occam factor δ is necessary. This new idea will be compared to other approaches in Section 5.4.

5.2.5 Other Approaches

A simple idea to address the size problem is to restrict the maximum tree size during the run of the GP, e.g., to set the maximum tree size to 10. However, one problem with this approach is the diversity of the population. In order to have sufficient genetic material in the population, one has to use large population sizes. More important is the fact that it is often unknown in advance which tree size is sufficient to solve a particular problem.

5.3 Analysis of Redundancy

In this section an explanation for the bloating phenomenon is introduced that is published in [Blickle and Thiele, 1994]. It is based on an analysis of redundancy in the trees. First, some definitions will be introduced. Note that the term "individual" is used as a synonym for "tree".

Definition 5.1 *The edge A in a individual J is called* redundant *if the fitness value of the individual is independent of the subtree located at edge A.*

Note that the redundancy defined in this way depends on the fitness function and on the context. It exactly reflects the view that the GP has of redundancy. It is also the same definition that [Nordin and Banzhaf, 1995] use for "introns". If the edge A is redundant it follows immediately that all edges in the subtree located at edge A are also redundant.

Definition 5.2 *The* proportion of redundant edges *in a individual J is given by*

$$p_r(J) = \frac{\textit{number of redundant edges in } J}{\textit{number of all edges in } J} \tag{5.6}$$

In order to observe the impact of redundancy on the crossover operation the term of *redundancy class* is introduced.

Definition 5.3 *The* redundancy class J^* *is the set of all individuals J that only differ from subtrees at redundant edges, i.e., for any two trees $J_1, J_2 \in J^*$, J_1 can be transformed into J_2 only by changing subtrees at redundant edges of J_1.*

An individual belongs only to one redundancy class, and all members of the class have the same fitness value denoted by $f(J^*)$. If the crossover site is chosen in the redundant part the represented solution has "survived" crossover. The probability for this event will be denoted by $p_s(J)$.

Theorem 5.1 *Let p_c be the probability of crossover and J an individual of the redundancy class J^*. The probability of individual J to remain in class J^* after crossover is given by*

$$p_s(J) \geq 1 - p_c + p_c p_r(J) \tag{5.7}$$

Proof: With probability $1-p_c$ the individual does not participate in crossover and with probability $p_c p_r(J)$ the crossover is performed at a redundant edge. In both cases the individual J remains in the class J^*. The "$\geq$" sign takes into account that an individual also survives if crossover is performed at a non-redundant edge and a "matching" subtree is inserted. However, this probability is very small. $\square$

The average probability of an arbitrary member of the redundancy class J^* to survive crossover is

$$p_s(J^*) \geq 1 - p_c + p_c p_r(J^*) \tag{5.8}$$

with $p_r(J^*) = \frac{\sum_{J \in J^*} p_r(J)}{|J^*|}$ the average proportion of redundant edges in the individuals of class J^*.

It follows from (5.7) that within a class more redundant individuals are more likely to survive. On the other hand it is very difficult to predict whether the redundancy of an individual will increase or decrease once a redundant crossover site has been chosen as this depends on the size of the new subtree being inserted. However, this size is independent of the redundancy, so, in average, the selection mechanism will dominate and the redundancy within a class is expected to increase.

Theorem 5.2 *Let $p_r(J^*)$ be the average redundancy of the redundancy class J^* before selection. The average redundancy $p_r'(J^*)$ after selection is in average independent of the selection method and $p_r'(J^*) = p_r(J^*)$.*

Proof: As the selection of any individual of the class T^* is independent of the redundancy and each individual has equal reproduction rate $R(f(J^*))$ no tree is preferred. Therefore, the redundancy will in average remain constant after selection. $\square$

5.3.1 Generational Behavior

Now the behavior of the Genetic Program over a whole generation is described.

Theorem 5.3 *Let $f(J^*)$ be the fitness value of the individuals of the redundancy class J^* with $|J^*|$ members and $R_\Omega(f)$ the reproduction rate of individuals with fitness f using the selection mechanism Ω. The expected number of members of J^* in the next generation is given by*

$$|J^*|^{k+1} \geq |J^*|^k R_\Omega(f(J^*))(1 - p_c + p_c p_r(J^*)) \tag{5.9}$$

Proof: $|J^*|^k R_\Omega(f(J^*))$ gives the expected number of members of the redundancy class J^* after selection and before crossover. Using this and combining (5.8) with Definition 5.2 Equation 5.9 is obtained. Actually $p_r(J^*)$ denotes the average redundancy in the class J^* before selection but it is used in Equation 5.9 as average redundancy after selection. However on average these two values are equal, as shown in Theorem 5.2. □

5.3.2 Effect of Redundancy

In order to evaluate the effect of redundancy the behaviour of the model derived in the previous section will be compared with a model that does not consider redundancy. In such a model, the expected number of offspring of an individual will only depend on the reproduction rate and on the crossover probability, i.e.,

$$|J^*|^{k+1} \geq |J^*|^k \tilde{R}(f(J^*))(1 - \tilde{p_c}) \tag{5.10}$$

A comparison with (5.9) can be made in two different ways. First, assume that the crossover probabilities are equal in both cases ($p_c = \tilde{p_c}$) and the effects of redundancy are reflected by a changed reproduction rate $\tilde{R}$:

$$|J^*|^k R(f(J^*))(1 - p_c + p_c p_r(J^*)) = |J^*|^k \tilde{R}(f(J^*))(1 - p_c)$$

Solving the above equation for $\tilde{R}$ yields:

$$\tilde{R}(f(J^*)) = R(f(J^*)) \left(1 + \frac{p_c p_r(J^*)}{1 - p_c}\right) \tag{5.11}$$

Here it is apparent how the redundancy increases the *effective reproduction rate* $\tilde{R}$. If, for example, a crossover probability of $p_c = 0.9$ and a redundancy of $p_r(J^*) = 0.11$ is assumed one calculates $\tilde{R}(f(J^*)) \approx 2R(f(J^*))$. This means that even a moderate redundancy of only 11% doubles the expected number of offspring in the next generation. Looking at redundancy from an individual's point of view it is advantageous to maximize the redundancy, because this increases the probability of the individual to survive.

An other way to observe the influence of redundancy is to calculate the *effective crossover probability* $\tilde{p_c}$ and assuming a constant reproduction rate $\tilde{R} = R$. Using the same approach as before one calculates:

$$\tilde{p_c} = p_c(1 - p_r(T^*)) \tag{5.12}$$

This equation shows that the effective crossover rate is decreased. The higher redundancy implies that it is less likely to chose a non-redundant edge as crossover site thereby hindering the evolution of new individuals. It follows that the probability to escape a potential local optimum decreases with time. From the viewpoint of GP as an optimization method, the redundancy is unfavorable because it aggravates the optimization process.

These considerations show the importance of redundancy for the performance of the Genetic Programming optimization method. One suggestion to control redundancy is given in the next section.

5.3.3 Marking Crossover

The idea of *marking crossover* is to avoid crossover at redundant edges by marking all edges that are traversed during evaluation of the fitness function: First, the marking flags of all nodes are cleared and if a node is evaluated during the fitness calculation the corresponding flag is set. After calculating the fitness function, only redundant nodes have cleared flags. The crossover is then restricted to edges with the flag set, i.e., non-redundant edges. By this useless crossover sites are avoided.

The additional time to set the marking flag is very low and the additional memory demand is one bit per node for the marking flag. The method implies that some subtrees are not evaluated if a "run-time" condition is met. That can only be achieved with a suitable implementation of the GP. Also the "total" redundancy can not be evaluated as it depends on the ordering of the evaluation of the subtrees. Consider, for example, the boolean `AND` function and the tree `AND(FALSE,SUBTREE)` where `FALSE` is a tree that always returns the value false and `SUBTREE` is an arbitrary tree . If the implementation evaluates the subtrees in order, the second subtree `SUBTREE` will never be traversed as the result always false. Here `SUBTREE` is recognized to be redundant. However, if the implementation evaluates the subtrees in reverse order, `SUBTREE` will in some cases be true and others false. So both subtrees will be evaluated and the redundancy will not be discovered.

5.3.4 Deleting Crossover

The marking method only avoids redundant crossover sites but does not address the bloating phenomenon directly as it leaves the redundant subtrees unchanged. A straightforward idea to reduce the tree size is to *delete* all redundant subtrees and replace each by a single randomly chosen terminal (*deleting crossover*).

Deleting crossover should also be able to clarify some points referred to "hitchhiking". Hitchhiking describes the phenomenon that unfit subtrees may be copied and propagated in the population by attaching themselves to highly fit trees. As long as the unfit subtree does not affect the fitness of the whole individual it can be seen as a redundant subtree. This is the reason why the frequency of a subtree is not related to its fitness or usefulness. This has been observed by several researchers [Tackett, 1994; Tackett, 1995; Rosca, 1995]. However the opinions differ in the valuation of hitchhiking. One hypothesis states the hitchhiking to be useful in that the "useless" subtrees may be helpful in later stages of the optimization and hitchhiking helps to keep the population diverse [Tackett, 1994]. From the analysis in the previous section one would suggest that the unfit (redundant) subtrees aggravate the optimization process. As the deleting crossover removes the redundant subtrees no hitchhiking is possible. Hence, if the performance increases using deleting crossover, hitchhiking is probably harmful for the optimization process. This point is addressed in the next section when several approaches to the bloating phenomenon are compared.

5.4 Experiments

In the previous section a mathematical description of the Genetic Programming was carried out by introducing the term of redundancy. Here some effects of this theory will experimentally verified. Two different experiments will be made. In the first experiment the 6-multiplexer problem will be used as an example for a discrete problem and illuminate the effects of redundancy. Second, the regression of a one-dimensional functional dependency obtained by a complex integral equation will be used as an example of a continuous problem. Both examples will also serve to compare the approaches described in Section 5.2.

5.4.1 The Setups

Besides the problem of tree size explosion, several items have to be considered in order to use GP successfully for symbolic regression. These are

- *The handling of mathematical exceptions during fitness evaluation:* As mathematical valid expressions are searched illegal mathematical operations are not permitted and considered by a penalty term for each violation.
- *The handling of floating point or integer constants:* All individuals with a fitness value better than a certain threshold f_{th} are selected for constant optimization. The optimization consists of sequentially changing all constants in the tree by a fixed amount $\xi = 0.1$ and accepting the new value as long as the fitness is improved or a maximum number of optimization steps s_{max} have been performed.
- *The choice of the fitness function:* The fitness function consists of three terms: The first term measures the error $A(J)$ of an individual J. The measure depends on the specific task and is introduced in the context of the experiments (Sect. 5.4). Furthermore, a penalty term PENALTY for mathematical exceptions is added in addition to the parsimony pressure that depends on the complexity $C(J)$ of the function. In summary, the fitness (that has to be minimized) of an approximation function J is given by:

$$f(J) = A(J) + \delta\, C(J) + j\mathrm{PENALTY} \tag{5.13}$$

where $j \in \{0, \ldots, l\}$ gives the number of occurrences of mathematical exceptions during evaluation of the fitness function.

Five basic setups are compared:

- **Simple** setup: Standard GP with a constant parsimony pressure δ.

- **EDI** setup: The usage of standard GP with EDIs and a constant parsimony pressure δ. The initialization range of the EDIs is [10, ..., 1010], as recommended in [Nordin *et al.*, 1995].

- **Deleting** setup: The use of standard GP with deleting crossover as described above and a constant parsimony pressure δ.

- **Adaptive** setup: The usage of standard GP with adaptive parsimony pressure as suggested in [Zhang and Mühlenbein, 1995], with the accuracy limit $\epsilon = 0.02$.

- **IOS** setup: The usage of standard GP with individual objective switching as described above, i.e., the fitness function is

$$f(J) = \begin{cases} 1 + A(J) + j\text{PENALTY} & : \quad A(J) + j\text{PENALTY} \geq \epsilon \\ \frac{v(J)}{v_{max}} & : \quad else \end{cases} \tag{5.14}$$

also with accuracy limit $\epsilon = 0.02$.

5.4.2 The 6-multiplexer problem

In the 6-multiplexer problem [Koza, 1992] the task is to find a boolean expression that performs the 6-multiplexer function using the function set $\mathcal{F} = \{And, Or, Not, If\}$ and the terminal set $\mathcal{T} = \{a_0,\ a_1,\ d_0,\ d_1,\ d_2,\ d_3\}$. For this discrete optimization problem the accuracy is measured by the number of incorrect classified data samples, i.e.,

$$A_d(J) = \sum_{i=0}^{63} \begin{cases} 0 & : \quad J(\mathbf{x_i}) = y_i \\ 1 & : \quad else \end{cases} \tag{5.15}$$

Hence, an optimal individual fulfills $A_d(J_{opt}) = 0$ (see also Example 2.1 on page 13). 100 runs were performed for each setup and each run was continued to the 50th generation even if a perfect individual was already found.

Population Size	$N = 200$
Selection Method	Tournament Selection 10
Crossover probability	$p_c = 0.9$
Mutation probability	$p_m = 0$
Crossover site at leaves	10%
Crossover site at internal nodes	90%
Complexity measure ($v(J)$= # of nodes)	$C(J) = \frac{v(J)}{v_{max}}$
No constant optimization	

Table 5.1: *The parameter setting for the 6-multiplexer problem.*

Experiment 1: Comparison of the Different Approaches

The first experiment compared the performance of the various methods to deal with the size problem. The results are summarized in Table 5.2. In all experiments a population size of $N = 200$, a tree size limit of $v_{max} = 100$ nodes, and a maximum of $k_{max} = 50$ generations is used. Note that the constant parsimony pressure is $\delta = 1$, as the smallest possible increase in accuracy is 1. p_{succ} gives the probability of success, i.e., the percentage of runs that found a perfect solution, v_{sol} reports the average complexity (in number of nodes) over all perfect individuals of the last generation and all runs, and v_{av} the average complexity over all individuals of the last generation and all runs.

First, the success-rates are compared. The **EDI** approach performs less well than the **Simple** approach. The reported benefits of this approach are not apparent for this particular problem. The **Adaptive** setup (46%) is clearly better than **Simple** (41%). Still a little better is the **IOS** setup (49%). To understand this one has to look at the success probability without parsimony pressure ($\delta = 0$). The **Simple** approach now has a 48% success probability. This means that a constant parsimony pressure seriously reduces the probability of success but reduces the complexity dramatically from an average tree size of $v_{sol} = 55.3$ nodes to $v_{sol} = 13.5$ nodes. Furthermore, the

Exp	Setup	δ	p_{succ} [%]	v_{sol}	v_{av}
	Simple	1.0	41	13.5	12.31
	Simple	0	48	55.3	63.77 (*)
	EDI	1.0	38	10.9	10.9
1	**Deleting**	1.0	57	13.1	12.00
	Deleting	0	62	46.4	57.73 (*)
	Adaptive	*var*	46	15.5	13.82
	IOS	-	49	17.77	49.7

Table 5.2: *Results of the experiments with four different setups (***Simple***,* **EDI***,* **Deleting***,* **Adaptive***) to determine a Boolean 6-multiplexer function. (*) marks runs that stopped after the occurrence of an optimal individual.*

Adaptive approach almost perfectly combines the advantages of a strong parsimony pressure and no parsimony pressure as the probability of success remains almost the same (46%) but the complexity is efficiently reduced. The **IOS** setup achieves the same performance as the unconstrained **Simple** approach. This is plausible, as the fitness functions are equal as long as the perfect solution has not been found. Afterwards, the **IOS** approach works towards smaller solutions. The average size of the population in last generation remains large ($v_{av} = 49,7$ nodes). However, the average size of solutions found is $v_{sol} = 17.77$ nodes, which is close to the values of the other approaches.

The best performer is the **Deleting** setup. With this approach the success rate is increased to 57% with and 62% without parsimony pressure. Obviously, due to the IF-function in the function set this approach exploits the redundancy very well. Furthermore, the increased success rate indicates that hitchhiking of useful subtrees does not occur in the 6-multiplexer problem. However, the **Deleting** setup is not able to reduce the tree size considerably without parsimony pressure. The behaviour of the **Deleting** setup and the underlying marking crossover is examined further in the next subsection.

Interesting to note is the fact that there is no need to use the **Adaptive** setup to reduce the complexity of the trees. All setups end up at almost the same average complexity. This is due to the fact that a relatively high constant parsimony pressure can be exceeded without having a negative effect on the accuracy (due to the discrete character of the error function for this problem). However, the great advantage of the **Adaptive** approach is that it simultaneously leads to almost the same high success rate as the unconstrained version. The **IOS** setup leads to an even better success-rate and gives also short solutions, although they are not as compact as the solution of the other approaches.

Experiment 2: The Redundancy Over Time

According to Theorem 5.3 the redundancy in the trees of a certain (not only the best) fitness value will increase over time, because the trees are treated equal during selection phase but more redundant trees have a higher chance to survive crossover. This expectation is verified by Fig. 5.1: in this single run, the GP was stuck at a local optima from generation 4 to 15 and increased redundancy is observed for all fitness values ($\delta = 0$).

Another way to observe the increasing redundancy is to continue a GP run even if a best solution is found. [Koza, 1992] reported the phenomenon that after some generations there are almost 70-80% best-fit individuals, but almost all best trees are different. This can be explained with redundancy: the trees are all different but belong to only a small number of different redundancy classes. A very high redundancy makes the trees robust against crossover. Figure 5.2 shows the course of redundancy versus time.

Experiment 3: Redundancy and Selection Intensity

For a given fitness value increasing redundancy can only be achieved by increasing the tree size, because some "minimal size" in the atomic

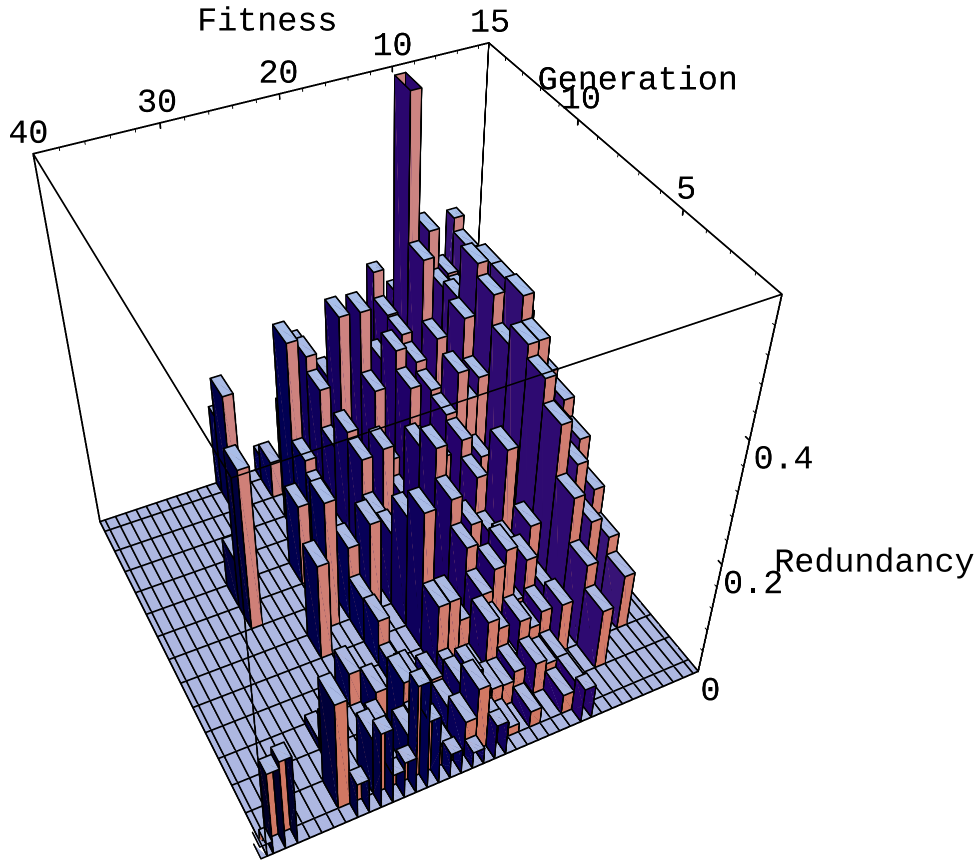


Figure 5.1: *The redundancy increases with time (6-multiplexer problem, tournament size $t = 10$, $\delta = 0$).*

part of the tree is necessary to reach this fitness. The tree size will grow with time and the "size problem" can be explained by redundancy. Figure 5.3 shows that the average tree size for a given fitness value is smaller if a higher reproduction rate is chosen. So if the redundancy can be reduced the average tree size will be reduced at the same time.

One way to achieve a reduction of redundancy is offered by Theorem 5.3: If the reproduction rate R_Ω is too low Theorem 5.3 implies that the probability that non-redundant trees survive is small and

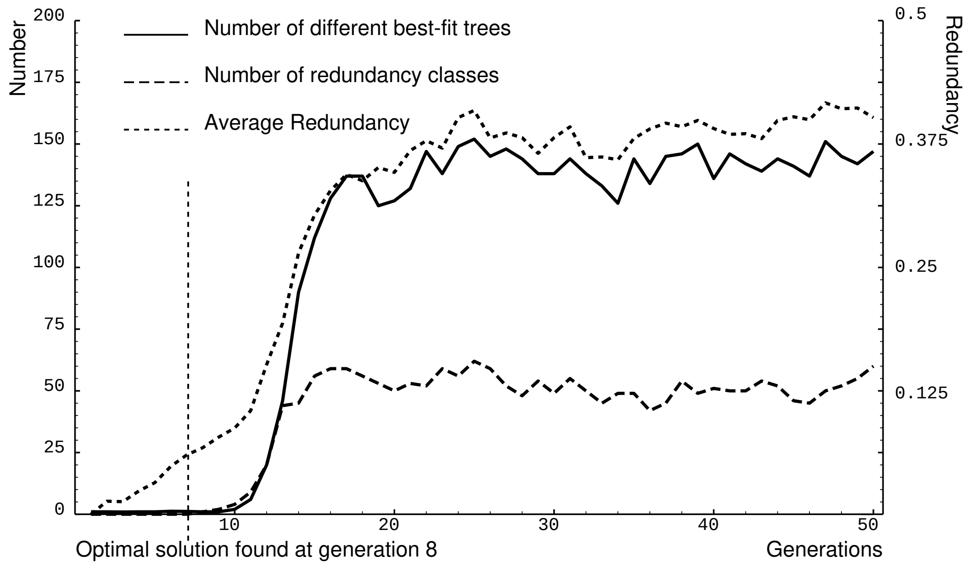


Figure 5.2: *Redundancy versus time after the occurrence of a best-fit individual ($\delta = 0$).*

redundancy-free trees will often vanish. Therefore a higher reproduction rate should lead to a lower redundancy. The results of the corresponding experiments with tournament selection are shown in Figure 5.4 for a tournament size t of 2, 5 and 10. It can be seen that redundancy decreases with increasing selection intensity, e.g. the redundancy at generation 50 for tournament size 2 is almost twice as high as the redundancy using tournament size 10. As the convergence time is very different for various selection intensity the relation between the average redundancy in a population versus the average population fitness gives a more meaningful description. In Figure 5.5 it can be seen that higher selection intensity implies lower redundancy.

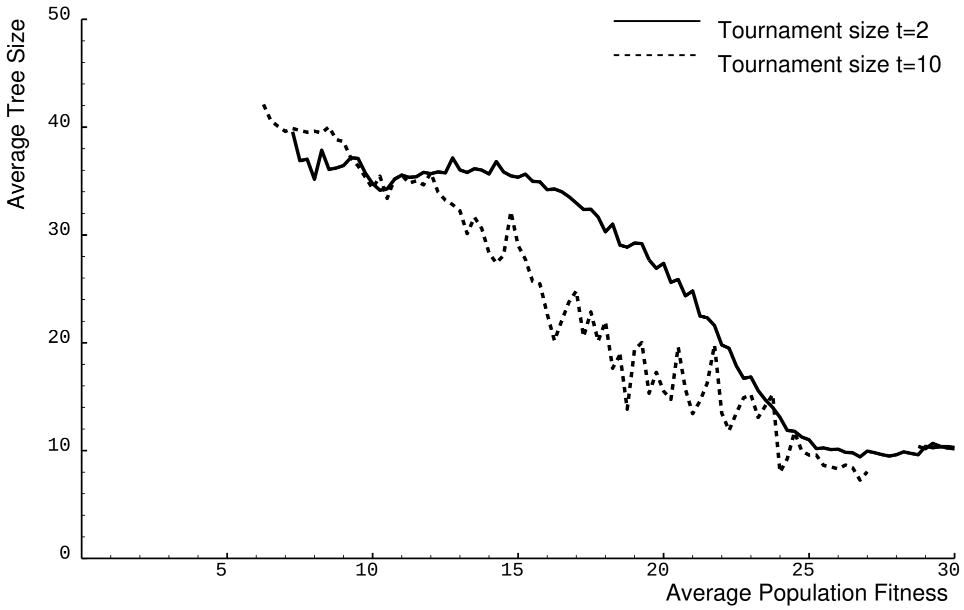


Figure 5.3: *Higher selection intensity leads to smaller tree sizes for the same fitness value (maxtreesize $= 50$, $\delta = 0$).*

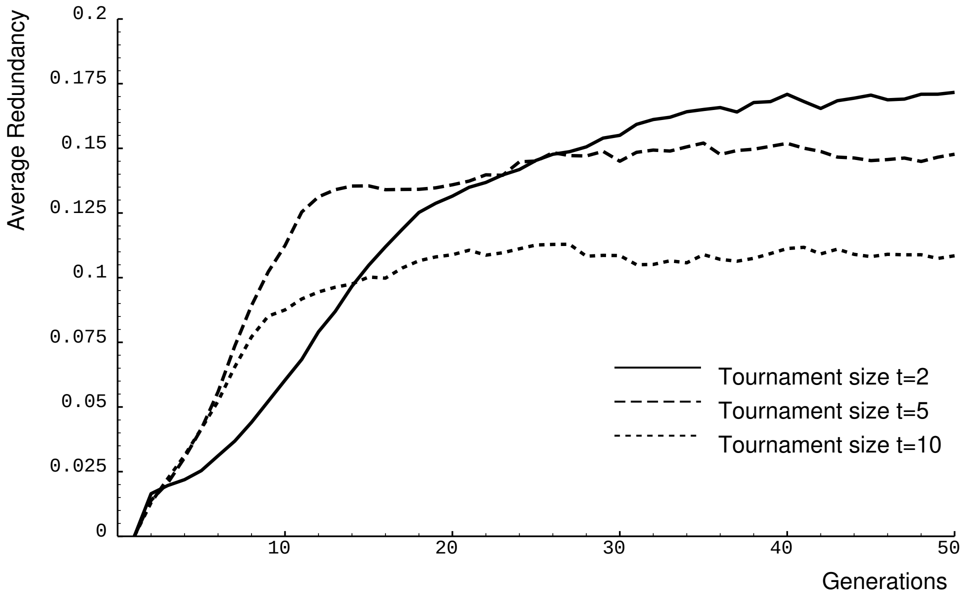


Figure 5.4: *Increasing selection intensity leads to decreasing redundancy ($\delta = 0$).*

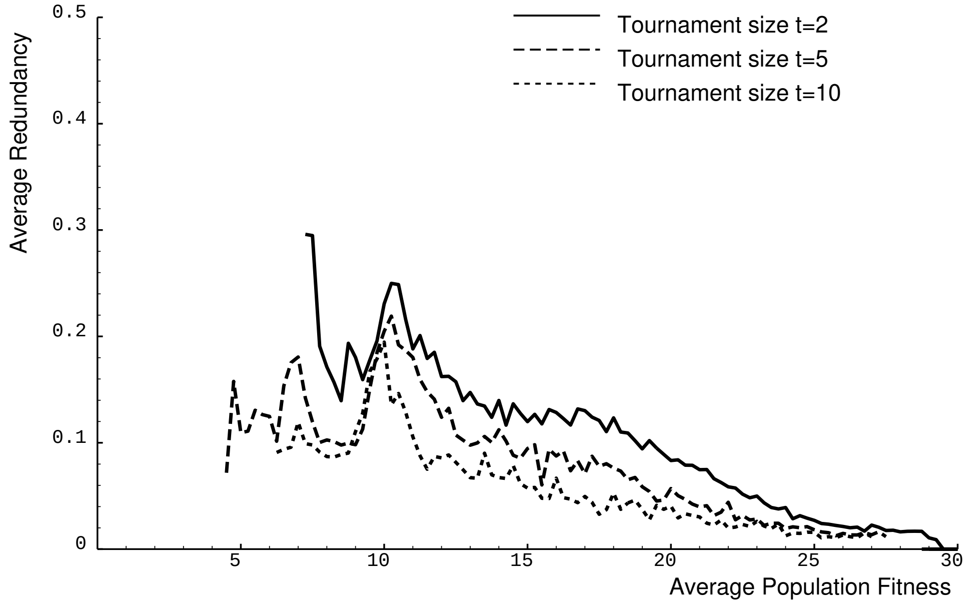


Figure 5.5: *Average population fitness versus redundancy with tournament selection ($\delta = 0$).*

5.4.3 Symbolic Regression

During the investigation of selection schemes in Chapter 4 several difficult integral equations were obtained. Also some approximation formulas were obtained by the GP method. In the following, one of these equations is used as an example for a continuous regression problem, namely the dependence of the selection variance $\Phi_T(t)$ on the tournament size t (Equation 4.36).

As the tournament size is a discrete parameter (only integer tournament sizes are possible) the data for approximating this equation consist of the tournament sizes and the corresponding selection variances, i.e., $\{\ (1, \Phi_T(1)),\ (2, \Phi_T(2)),\ \ldots,\ (30, \Phi_T(30))\ \}$. The values Φ_T have been computed by numerical integration. Values for larger tournament sizes ($t > 30$) are of no practical relevance. Hence only the interval $t = \{1, \ldots, 30\}$ is considered and the approximation formula needs not to be verified on unseen data, i.e., the generalization capabilities of the approximation formula are not taken into account. In Appendix B all numerical values of the selection variances are given (Table B.1).

The maximum relative error over all data points assures that for each data point the error is below a certain value:

$$A_c(J) = \max_{i=1,\ldots,l} |\frac{J(\vec{x}_i) - y_i}{y_i}| \tag{5.16}$$

For this experiment the function set was $\mathcal{F}$={Plus, Subtract, Times, Divide, Log, Sqrt, Exp} and the terminal set was $\mathcal{T}$ = {t, π, RFPC}, where RFPC symbolizes a random floating point constant. Twenty runs were carried out for each setup and each experiment. The parameter setting of the GP are summarized in Table 5.3.

Selection method	Tournament Selection 10
Crossover method	crossover limiting tree size
Crossover probability	$p_c = 0.9$
Constant-optimization threshold	$f_{th} = 1.0$
Maximum number of constant-optimization steps	$s_{max} = 10$
Complexity measure	$C(J) = \frac{v(J)}{v_{max}}$
Penalty value	$PENALTY = 100.000$

Table 5.3: *The global parameters of GP for the symbolic regression problem.*

The results of the experiments are shown in Table 5.4. The entry A_{best} gives the maximum relative error (in percent) of the best approximation formula found in the last generation. A_{av} gives the average error (in percent) of the best approximations found in the 20 runs. Similar v_{best} describes the complexity (in number of nodes) of the best solution found and v_{av} is the average complexity of all 20 best solutions.

Experiment 4: In this experiment the four setups **Simple**, **EDI**, **Deleting**, and **Adaptive** are compared with a very restricted maximum tree size (number of nodes in the trees) $v_{max} = 15$, a high population size of $N = 10000$, and an maximum of 30 generations ($k_{max} = 30$). This parameter setting is designed for the **Simple** setup. The first observation to make is that GP is well suited for this problem as short solutions with high accuracy have been found even with the **Simple** setup. This indicates that the choice of a maximum tree size of 15 is sufficient to solve this problem. The approximation accuracy was at least 4%. The best result found had an maximum relative error of 0.56% and can be simplified to:

$$\Phi_T(t) \approx \frac{0.918}{\ln(1.186 + 1.328t)} \qquad ; t \in \{1, \ldots, 30\} \qquad (4.37)$$

The results for the **EDI** and **Deleting** setup are of the same quality and there seems to be no advantage in using one of these more complicated setups. In the **Adaptive** setup the error is slightly increased and the complexity slightly decreased. The average error of the **IOS** setup lies in a similar range although the best approximation is not as good as for the other setups. But one has to consider that the approximation accuracy is set to $\epsilon = 2\%$. The small differences in the results seem reasonable as the restricted tree size and the small number of generations probably give no time and space for the more sophisticated approaches to evolve.

Experiment 5: Subsequently, another experiment was set up with a higher maximum number of generations ($k_{max} = 100$) and a higher maximum tree size ($v_{max} = 45$) but a smaller population size ($N = 3000$). This setup should give enough time (generations) for the **EDI** and **Adaptive** setup to work.

The quality of the best and average best solution for **Simple**, **EDI**, and **Deleting** setup improved dramatically. But in all cases this was at a cost of a more than doubled complexity of the solution compared to Experiment 4. This means that the optimization worked towards smaller approximation errors and not towards smaller complexity values. This is remarkable because of the relatively high parsimony pressure at the end of the optimization. If, for example, the

Exp	Setup	δ	$A_{best}[\%]$	$A_{av}[\%]$	v_{best}	v_{av}
	Simple	0.01	0.56	1.8	12	12.75
	EDI	0.01	0.30	1.6	12	12.38
4	**Deleting**	0.01	0.34	2.0	13	12.17
	Adaptive	*var*	0.98	1.8	11	11.85
	IOS	-	1.19	2.0	15	12.61
	Simple	0.01	0.08	0.56	29	26.92
	EDI	0.01	0.08	0.62*	21	26.23
5	**Deleting**	0.01	0.10	0.76	25	26.38
	Adaptive	*var*	0.84	1.6	8	13.5
	IOS	-	0.62	1.64	16	16.9

Table 5.4: *Results of the experiments with four different setups (***Simple***,* **EDI***,* **Deleting***,* **Adaptive***) to evolve an approximation formula for the integral equation (4.15). For the values marked with (*) refer to the text.*

best approximation found for the **Simple** setup is considered, one sees that $C_{best} = \frac{29}{45} \approx 0.64$ and $A_{best} = 0.0008$. As $\delta = 0.01$ this means that the contribution of the complexity to the fitness (according to equation 5.13) is about 8 times the contribution of the accuracy.

The **EDI** setup seems not to have any advantages over the **Simple** setup. In contrary, the average error of the best solutions is very high (152%, not shown in the table). This was the result of two runs for which the GP found very bad solutions. This phenomenon has also been reported in [Nordin *et al.*, 1995] where on the other hand good solutions were found very quickly. This could not be observed in experiments here. If the two "runaways" are neglected the average error is 0.62% (marked with (*) in the table).

The **Deleting** approach also seems to have no advantages. All results are more or less similar to those of the **Simple** setup.

The **Adaptive** setup seems to be very well suited to reduce the complexity of the solutions. Although the maximum tree size could grow up to 45 nodes, the average complexity of the solutions was only 13.5 nodes. The fact that the accuracy of approximation is one magnitude worse than those of the **Simple** , **EDI** , and **Deleting** setup is plausible as the accuracy limit was set to $\epsilon = 2\%$.

The **IOS** setup also works well on this example. The average accuracy is almost the same as for the **Adaptive** setup and the average complexity is only slightly increased (16.9 nodes compared to 13.5 nodes for the **Adaptive** setup). Consequently, the **IOS** setup is also well suited for this constrained problem. This is remarkable, as the application of this approach is much simpler and not restricted to this single domain as the **Adaptive** setup.

5.5 Conclusion

In summary, there seems to be no advantage in using the **EDI** or **Deleting** approach for the particular continuous problem in order to reduce the complexity of the solutions. They both work as well as the **Simple** setup. Probably, in a continuous problem, no subtrees are "completely" redundant or "complete" introns but may contribute only "very little" to the quality of an approximation. They are not recognized by these methods as introns and consequently do not have the intended effect. This conjecture is supported by the good performance of the **Deleting** setup on the (discrete) 6-multiplexer problem. However, it remains open as to why the **EDI** approach fails to work for this example.

The superior methods to obtain compact and accurate solutions are the method of adaptive parsimony pressure and the **IOS** approach that both worked well on both problems. Especially the good results for the **IOS** approach are remarkable as the principle is very general and not explicitly designed for evolving short solutions. The evolved solutions were compact and accurate within the user-specified limits.

For discrete problems there may be no need for this method as a constant parsimony pressure is sufficient to evolve compact solutions. But one has to consider that a constant parsimony pressure might reduce the probability of success.

Part II

Application of Evolutionary Algorithms to System Synthesis

Chapter 6

System-Level Synthesis

Part II of this thesis is concerned with the automated design and optimization of complex digital systems composed of hardware and software components. Although this type of system has been developed by engineers for many years only the use of computer aided design (CAD) tools allows today's complex designs to be handled in a rational way. Furthermore, CAD systems helped to shorten time-to-market and reduce costs of new products. Consequently, research in CAD methods is followed with interest by industry.

The complexity of the design is not only determined by the number of subcomponents but also by their *heterogeneity*. *Application specific systems* designed for specific tasks are a typical representative of heterogeneous systems and their efficient design will become increasingly important in the future. These systems are also termed *embedded systems* as they are usually embedded in a specific technical context. Heterogeneity manifests itself not only as hardware and software components, but also as analog/digital components and mechanical/electronical components (sensors and actors).

From a historical perspective one can see several stages in the development of CAD: The first step in design automation was automatic placement and routing (geometric and transistor layout) which became necessary as the number of components grew. The next step was *logic synthesis* allowing the automated generation of a circuit at gate level from a specification given as Boolean functions or finite state machines. Recently, the first commercial tools for a further level of abstraction became available. In *high-level synthesis* the behaviour of the system is described using logic and arithmetic operations such as multiplication and addition which are transformed into a structural description consisting of data and control paths.

The next level of abstraction will start with an algorithmic problem description, i.e., one specifies the task to be performed by an algorithm (and not by Boolean functions or transistor schematic). The algorithm has to be executed by a system that is unknown in advance where several constraints are given, for example, the maximum cost of the system to be designed or time constraints for critical operations. This level can be termed *system level* or *architecture level*. Although there is no common definition of what system-level synthesis is, the following characterization might be appropriate: System-level synthesis can be described as a mapping from a behavioral description where the functional objects possess the granularity of algorithms, tasks, procedures, or processes onto a structural specification with structural objects being general or special purpose processors, ASICs, buses and memories. Tasks can be quite complex, for example, a complete filter operation, matrix multiplication or Fourier transformation. The granularity of the model is very coarse, i.e., details of the realization and algorithm are not considered. Nevertheless, decisions made at this stage of a design have far reaching implications.

Three goals of system synthesis are:

- Short time-to-market:
  Automated design systems allow faster development times and higher re-utilization of designs.

- Error reduction:
  To reduce expensive iterations in the design process it is desirable to use design tools that guarantee provably correct designs. This can be achieved by a strictly formal description of the system behavior and the definition and the usage of formal verification methods.

- Design space exploration:
  Especially in the early stages of the design, basic decisions are made that affect the performance and costs of the system. Synthesis tools should suggest several implementations in reasonable time.

The main problems concerning the system-level design are:

- Specification:
  Basis for a formal treatment of any problem is a sound model capturing relevant properties of a system.

- Performance estimation:
  The cost, execution time and memory demands of a complex operation (such as matrix multiplication) performed by, for example, a RISC processor or a dedicated hardware chip, cannot be exactly known in advance, as the exact measure of these values implies the synthesis of these components. Nevertheless, estimations are possible, for example, if a library based approach is used where data of similar realizations are stored.

- Optimization:
  There are many tradeoffs which have to be examined, and the high mutual dependency of different design criteria make it difficult to manually determine the "best" architecture for a specific problem. Consequently, an automated optimization tool is of great interest.

The proposed methodology treats the problem of optimizing the mapping of an algorithm-level dataflow graph based specification onto a heterogeneous hardware/software architecture (*hardware/software codesign*). This problem requires:

1. the *selection of the architecture* (allocation) from a specified set of possible architectures,
2. the *mapping* of the algorithm onto a selected architecture in space (binding) and time (scheduling), and
3. the design space exploration with the goal of finding a set of implementations that satisfy a number of constraints on cost and performance.

The last point is of special interest if one is interested in the so called *Pareto points* of the design space. Usually, in system-synthesis many different criteria have to be optimized, for example, cost of an implementation, data-throughput, power consumption, maintainability. The concept of Pareto-optimality gives a measure of concurrently comparing implementations to several criteria.

Definition 6.1 *A point J_i is* dominated *by point J_k if point J_k is better than or equally good as point J_i on each criteria, denoted as $J_i \succ J_k$. A point (implementation) is said to be Pareto optimal if it is not dominated by any other point.*

Example 6.1 *Consider the trade-off between cost c and speed (period P) of an implementation. Fig. 6.1 shows the design-space of an example (the specification of Example 7.4 that will be introduced in the next chapter). The implementations are marked by crosses. Three Pareto points are shown with ($c = 303$, $P = 39$) and ($c = 353$, $P = 9$, and ($c = 403$, $P = 4$). The long horizontal and vertical lines spread out the regions where these points dominate all other implementations.*

An Evolutionary Algorithm allows to determine the Pareto set in a single optimization run as the EA operates on populations. The necessary preconditions (fitness function and selection scheme) will be

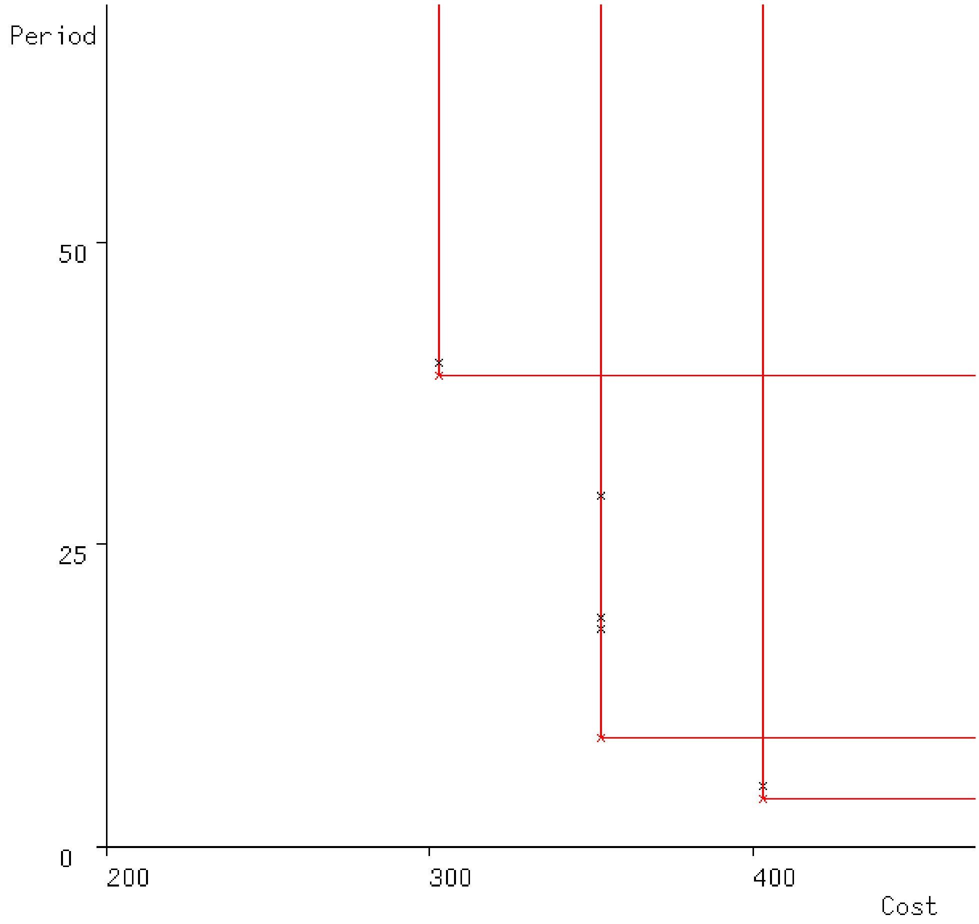


Figure 6.1: *Example of a design space and Pareto points.*

discussed in Chapter 8 and a design-space exploration will be carried out in Chapter 9.

The methodology concentrates on the optimization of dataflow-dominant systems. The other problems mentioned above will not be considered here. Nevertheless, solutions to these problems are required to use the proposed optimization methodology efficiently. For example, without accurate performance estimations a meaningful optimization can not be performed.

The novelty of the approach consists in:

- *providing a new system-level algorithm/architecture model* for heterogeneous hardware/software systems in which the underlying architecture is not fixed a priori but a set of architectures that should be investigated can be specified using a graph-theoretic framework,
- *formulating a new formal definition for system-level synthesis* including steps (1) and (2) leading to the specification of the synthesis problem as an optimization problem, and
- *applying Evolutionary Algorithms to system-level synthesis* and showing that they can perform steps (1)-(3) in a single optimization run.

From the analysis in Part I it is plausible that Evolutionary Algorithms are a good candidate for system-level synthesis because they

- iteratively improve a *population* (set) of implementations,
- they do not require the quality (cost) function to be linear,
- they are known to work "well" on problems with large and non-convex search spaces and
- they have a heuristic character, i.e. a moderate run-time. The guarantee of optimal solutions is not required as the optimization process uses estimated data.

The remainder of Part II is organized as follows. First, in this chapter a short summary of existing approaches to system-level synthesis and an overview of the proposed optimization methodology are given. In the next chapter the new specification model including the problem graph and the architecture graph will be formally defined. It is explained how to model costs, delays, bus- and pin- constraints, how to model single-chip systems, multiple chip solutions, communication, and resource sharing. The task of system-synthesis is given as a constrained global optimization problem as stated by Def. 2.1. The tasks of the Evolutionary Algorithm including the definition of appropriate fitness functions for evaluating the quality of an implementation are outlined in Chapter 8. The example of a video codec is used in Chapter 9 as a case study to explain the optimization methodology and to show its performance.

6.1 Existing Approaches to System-Level Synthesis

Many different approaches to system-level synthesis already exist that are classified according to their class of input specifications:

- *Control-dominant specification:* Methods in this class consider control-dominated specifications, e.g., communicating sequential processes [Buchenrieder *et al.*, 1995],[Thomas *et al.*, 1993], C-code [Hardt and Camposano, 1995] and extensions thereof [Ernst *et al.*, 1994] or finite-state-machine based specifications, e.g., [Ismail *et al.*, 1994]. In the mapping phase, (static) scheduling of tasks and communications cannot be done due to non-determinism of execution times. Here, estimation mainly depends on profiling and simulation.

- *Dataflow-dominant specification:* Methodologies in this class consider dataflow specifications, e.g. dataflow graphs, e.g. [Kalavade and Lee, 1994], precedence- and taskgraphs or parallel languages, e.g. UNITY [Barros and Rosenstiel, 1992]. The approach in [Gupta and Micheli, 1992] belongs to the same category, however, with extensions allowing operations with non-deterministic delays.

Tightly coupled with the class of input specifications is the scope of *target architectures*:

- *Dedicated control & data path in VLSI:* [McFarland, 1986], [Lagnese and Thomas, 1991], [Vahid and Gajski, 1992] are approaches to partition a functional specification for high-level synthesis. The target architecture is in most cases a dedicated hardware architecture including a control path and a data path.

- *Multi-chip dedicated VLSI architecture:* Some methodologies do focus on multichip VLSI solutions, e.g., [Küçükçakar and Parker, 1995]. All functionality is mapped onto a dedicated multi-chip architecture with busses.

- *Hardware/Software Architectures:* In the realm of hardware/software architectures, most approaches consider the target architecture to be fixed, e.g., [Kalavade and Lee, 1994] (one processor and custom hardware communicating via memory-mapped I/O), [Ernst *et al.*, 1994] (one RISC processor and one or more given custom blocks and predefined HW-modules that communicate mutually exclusive by memory coupling using a single CSP-like protocol), or [Gupta and Micheli, 1992] (one programmable component and multiple hardware modules communicating with each other using one system bus, the processor being the master). In [D'Ambrosio and Hu, 1994], also the allocation of components is considered as a task of the mapping process.

Finally, different approaches are classified according to their optimization model: Most of the mentioned methodologies consider system-level synthesis as a *partitioning* problem. Partitioning techniques have been applied for many different problems in hardware synthesis, e.g. [Camposano and Brayton, 1987], [McFarland, 1986], [Lagnese and Thomas, 1991], [Vahid and Gajski, 1992]. The goal is in most cases to meet chip capacity and time constraints. Most of these approaches present a clustering-based approach. [Küçükçakar and Parker, 1995] considers multichip modules, without programmable components. In some clustering based approaches like e.g., [Barros and Rosenstiel, 1992], only the space mapping (binding) is considered.

Finally, optimization methods can be classified into

- *exact methods:* These methods include enumerative search techniques and approaches based on integer linear programming. In [D'Ambrosio and Hu, 1994], e.g., a tool based on enumeration is used to find Pareto-implementations. These approaches suffer from long run-times and are only worth investigating if the estimation of costs and performance is proven to be highly accurate and the number of objects small.

- *heuristics:* Gupta in [Gupta and Micheli, 1992] present a heuristic for hardware/software partitioning where all functionality is initially mapped to hardware with the goal to move as much functionality as possible into the processor such that performance constraints remain guaranteed. [Ernst *et al.*, 1994] present an approach that starts with an initial partition in software and moves functionality to hardware until timing constraints are met. Simulated annealing is used in the outer optimization loop and adapted estimated parameters in the inner loop. Recently, the approach described in [D'Ambrosio *et al.*, 1995] has has been changed from using global enumeration techniques [D'Ambrosio and Hu, 1994] to applying Evolutionary Algorithms for design space exploration.

6.2 Overview of the Optimization Methodology

Figure 6.2 gives an overview of the new optimization methodology.

The specification consists of a dataflow graph based problem graph (see Fig. 6.2a), an architectural model (see Fig. 6.2c), user-defined mapping constraints (see Fig. 6.2b) and an optimization procedure (see Fig. 6.2d) using an Evolutionary Algorithm where each individual codes an implementation of the problem graph including an architecture and a mapping of nodes in the problem graph in space (binding) to that architecture.

Example 6.2 *Fig. 6.2e shows some individuals out of a population* $\vec{J}$ *of implementations. They correspond to the problem graph in Fig. 6.2a including the binding of functional nodes to functional resources (RISC, Hardware Modules 1 and 2) and the binding of communication nodes (shaded nodes) to bus resources (shard bus, point-to-point bus). The scheduling of nodes inside the functional resources is not shown. Note that each individual of a population may code a different architecture.*

The EA implicitly performs an *exploration of the design space*: it is likely that in the end, several optimal implementations are obtained with possibly completely different architectural features if an appropriate selection scheme is used. This fact is used to obtain the Pareto set of implementations.

As good heuristics for scheduling are available, the individuals code allocation and binding only. The scheduling is considered to be part of the fitness evaluation of an individual.

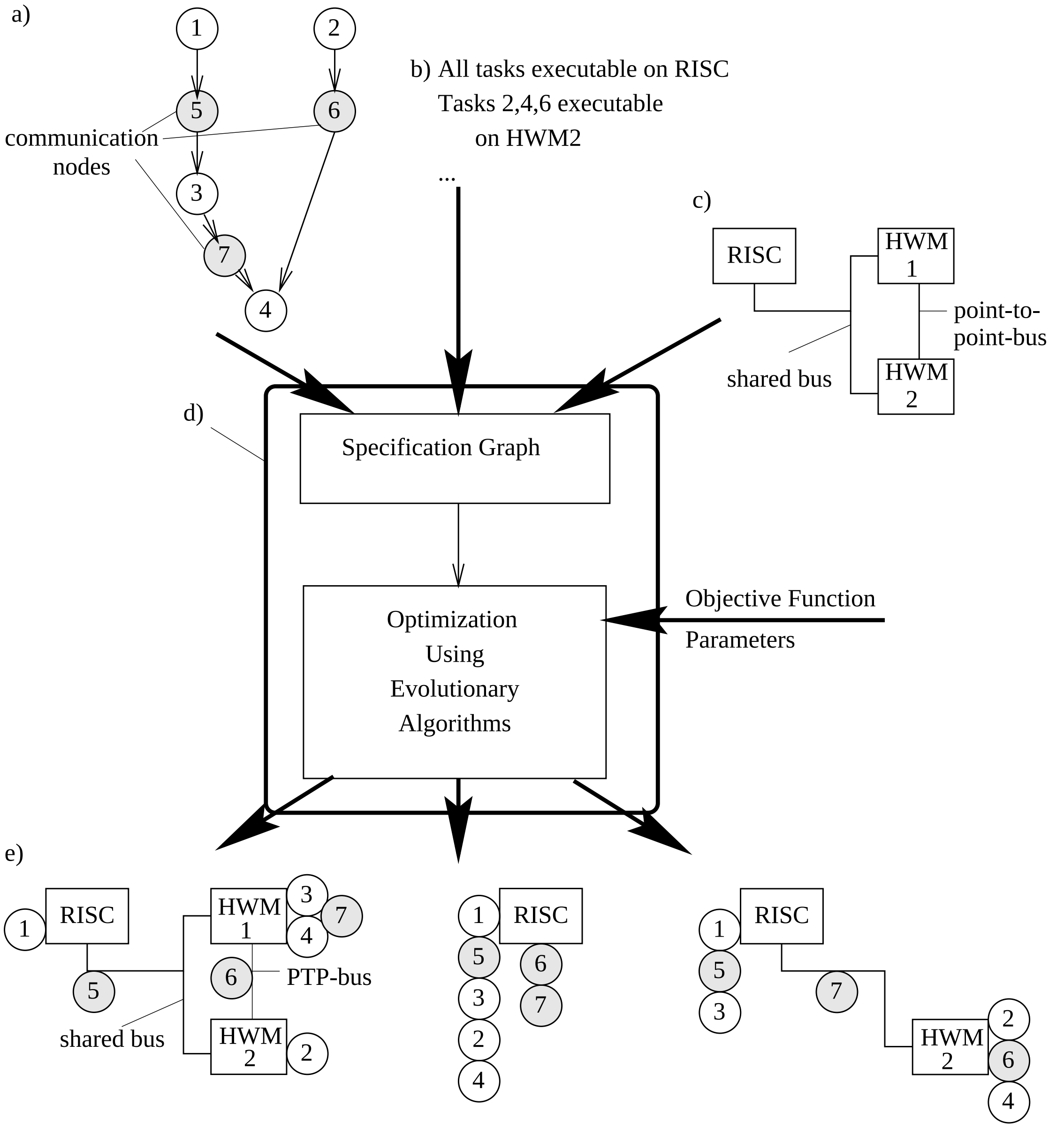


Figure 6.2: *Overview of the proposed optimization methodology: a) dataflow based algorithm specification, b) mapping constraints, c) architecture template, d) optimization methodology, e) implementations*

Chapter 7

Modeling Algorithms and Architectures

This chapter introduces a graph-based model to handle the system-synthesis problem. As the focus is on dataflow-dominant architectures the model is based on dataflow graphs. Several dataflow graphs are combined to from a specification graph that captures all knowledge of the synthesis problem. The three main synthesis tasks (allocation, binding and scheduling) are defined in the context of this model and the task of system synthesis is defined as a constrained global optimization problem.

7.1 Specification

The specification model consists of three main components:

- The algorithm to be mapped on an architecture in addition to the class of possible architectures described with an universal *dependence graph* $G(V, E)$.

- The user-defined mapping constraints between algorithms and architectures are specified in a specification graph $G_S(V_S, E_S)$. Additional parameters used for formulating the objective functions and further functional constraints are associated to the components of G_S.

- Associated to nodes and edges of the specification graph are activations which characterize allocation and binding.

The concept of a dependence graph is used to describe algorithms as well as architectures on different levels of abstraction.

Definition 7.1 (Dependence Graph) *A* dependence graph *is a directed graph $G(V, E)$. V is a finite set of nodes and $E \subseteq (V \times V)$ is a set of edges.*

For example, the dependence graph to model the dataflow dependencies of a given algorithm will be termed *problem graph* $G_P = (V_P, E_P)$. Here, V_P contains nodes which model either functional operations or communication operations. The edges in E_P model dependence relations, i.e. define a partial ordering among the operations.

Example 7.1 *One can think of a problem graph as the graph obtained from a dataflow graph of an algorithm by inserting communication nodes into some edges of the dataflow graph (see Fig. 7.1). These nodes will be drawn shaded in the examples.*

The architecture including functional resources and buses can also be modeled by a dependence graph termed *architecture graph* $G_A = (V_A, E_A)$. V_A may consist of two subsets containing functional resources (hardware units like adder, multiplier, RISC processor, dedicated processor, ASIC) and communication resources (resources that handle the communication like shared busses or point-to-point connections). An edge $e \in E_A$ models a directed link between resources. All the resources are viewed as *potentially allocatable* components.

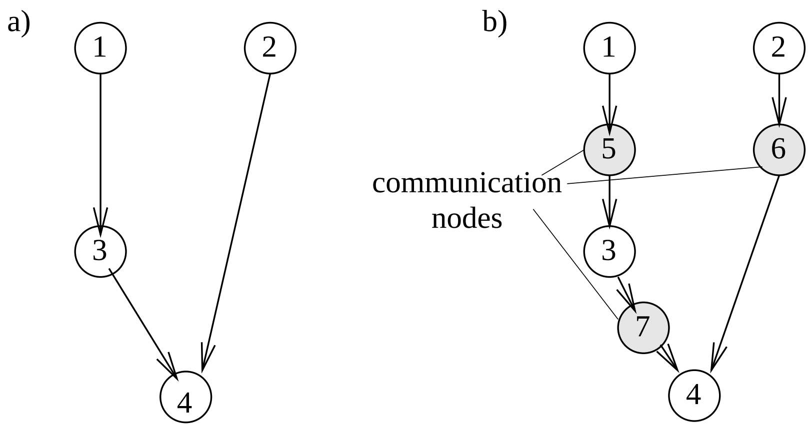


Figure 7.1: *A dataflow graph (a) and the corresponding problem graph (b).*

Example 7.2 *Fig. 7.2a) shows an example of an architecture consisting of three functional resources (RISC, hardware modules HWM1 and HWM2) and two bus resources (one shared bus and one unidirectional point-to-point bus). Fig. 7.2b) shows the corresponding architecture graph G_A.*

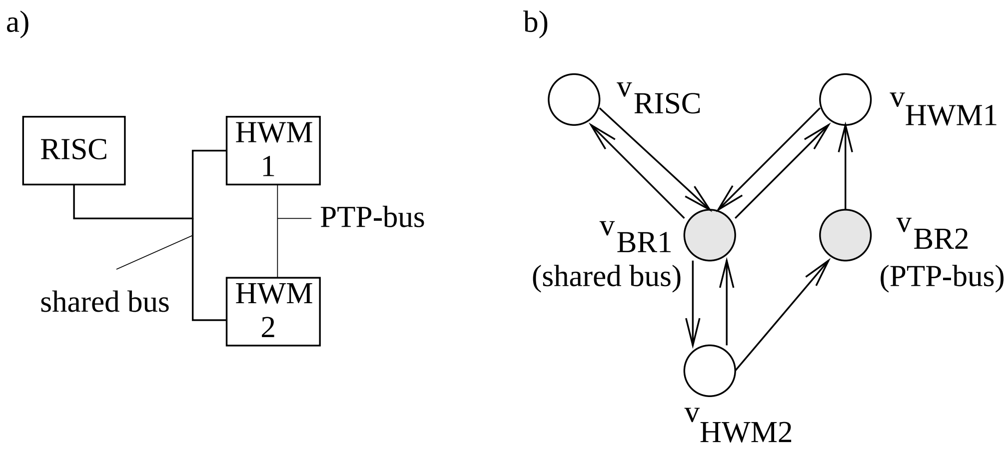


Figure 7.2: *An example of an architecture (a), and the corresponding architecture graph G_A (b).*

In addition, in a next level of abstraction one may define a dependence graph termed *chip graph* $G_C(V_C, E_C)$ whose nodes correspond to integrated circuits and off-chip communication resources.

Example 7.3 *Fig. 7.3a) shows an example of a multi-chip architecture consisting of two integrated circuits* chip1 *and* chip2 *and a bi-directional point-to-point bus resource. Fig. 7.3b) shows the corresponding chip graph* G_C*.*

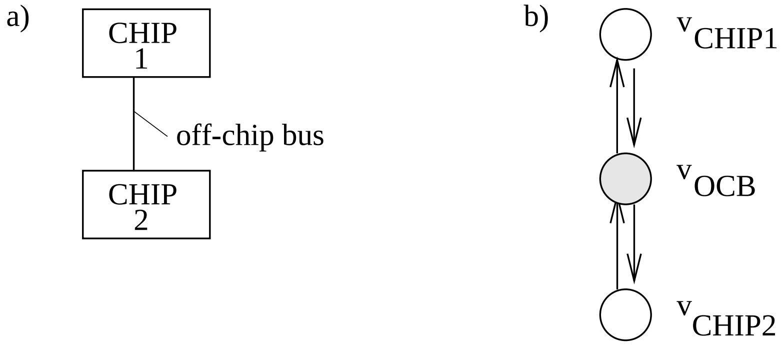


Figure 7.3: *An example of a multi-chip architecture (a) and the corresponding chip graph* G_C *(b).*

Note that the problem, architecture and chip graph are examples only. Next, it is shown how user-defined mapping constraints can be specified in a graph based model. Moreover, the *specification graph* will also be used to define *binding* and *allocation* formally.

Definition 7.2 (Specification Graph) *A* specification graph *is a graph* $G_S = (V_S, E_S)$ *consisting of* d *dependence graphs* $G_i(V_i, E_i)$ *for* $1 \leq i \leq d$ *and a set of* mapping edges E_M*. In particular,* $V_S = \bigcup_{i=1}^{d} V_i$*,* $E_S = \bigcup_{i=1}^{d} E_i \cup E_M$ *and* $E_M = \bigcup_{i=1}^{d-1} E_{Mi}$*, where* $E_{Mi} \subseteq V_i \times V_{i+1}$ *for* $1 \leq i < d$*.*

Consequently, the specification graph consists of several layers of dependence graphs and mapping edges which relate the nodes of two neighboring dependence graphs. These layers correspond to levels of abstractions, for example algorithm description (problem graph), architecture description (architecture graph) and system description (chip graph). The edges represent user-defined mapping constraints in the form of a relation: 'can be implemented by'.

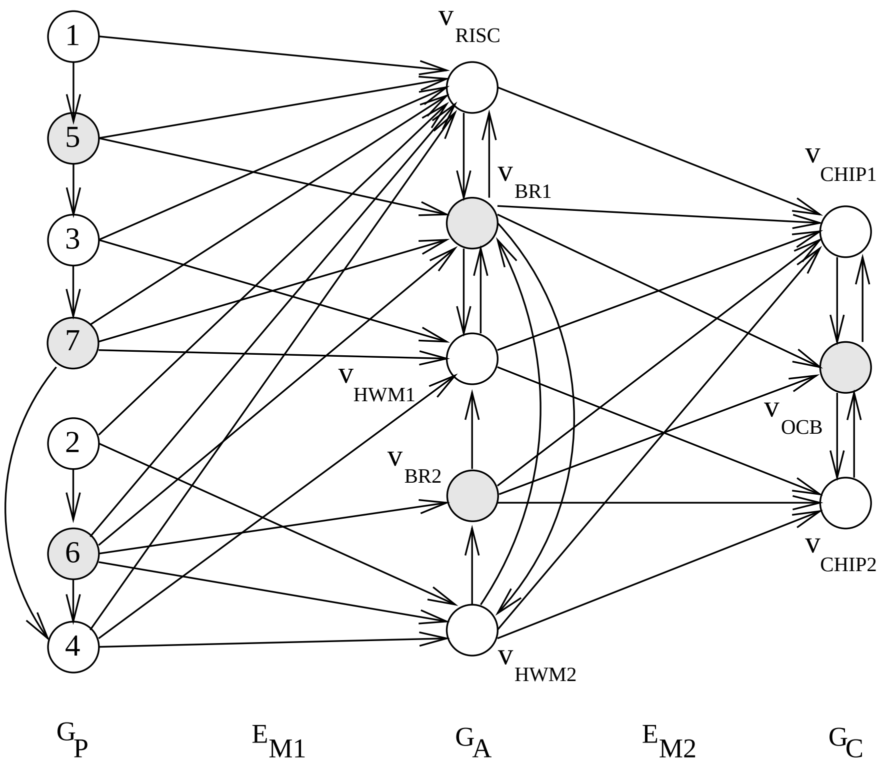


Figure 7.4: *An example of a specification graph G_S.*

Example 7.4 *Fig. 7.4 shows an example of a specification graph using the problem graph of example 7.1 (left), the architecture graph of example 7.2 (middle) and the chip graph of example 7.3 (right). The edges between the two subgraphs are the additional edges E_{M1} and E_{M2} that describe all possible mappings. For example, operation v_1 can be executed only on v_{RISC}. Operation v_2 can be executed on v_{RISC} or v_{HWM2}.*

Note that it may be useful to map communication nodes of the problem graph to functional resources: If both predecessor and successor node of a communication node are mapped to the same functional resource, no communication is necessary and the communication is internal*. In this case, the communication can be viewed as being handled by the functional resource.*

Also, communication v_7 can be executed by v_{BR1} or within v_{RISC} or v_{HWM1}. It can also be seen, that the specification allows the RISC *processor, the hardware modules* HWM1*,* HWM2 *and the communication modules* BR1*,* BR2 *to be implemented in* CHIP1*. The communication* BR1 *can either be handled by* CHIP1 *or by the off-chip bus* OCB*.*

In this model of a specification graph the expert knowledge about useful architectures and mappings is captured in a flexible and intuitive way.

In order to describe a concrete mapping, i.e. an *implementation*, the term *activation* of nodes and edges of a specification graph is defined. Based on this, *allocation*, *binding* and *scheduling* will formally be defined in the next section.

Definition 7.3 (Activation) *The* activation *of a specification graph $G_S(V_S, E_S)$ is a function $a : V_S \cup E_S \mapsto \{0, 1\}$ that assigns to each edge $e \in E_S$ and each node $v \in V_S$ the value* 1 *(*activated*) or* 0 *(*not activated*).*

The activation of a node or edge of a dependence graph describes its use. In the examples used so far, all nodes and edges of the problem graph contained in G_S are necessary, i.e. activated. The determination of a an implementation can be seen as the task of assigning activity values to each node and each edge of the architecture graph and/or chip graph. An activated mapping edge represents the fact that the source node is implemented on the target node.

7.2 The Tasks of System Synthesis

Definition 7.4 (Allocation) *An* allocation α *of a specification graph is the subset of all activated nodes and edges of the dependence graphs, i.e.,*

$$\alpha = \alpha_V \cup \alpha_E$$

$$\alpha_V = \{v \in V_S | a(v) = 1\}$$

$$\alpha_E = \bigcup_{i=1}^{d} \{e \in E_i | a(e) = 1\}$$

Definition 7.5 (Binding) *A* binding β *is the subset of all activated mapping edges, i.e.,*

$$\beta = \{e \in E_M | a(e) = 1\}$$

Definition 7.6 (Feasible Binding) *Given a specification G_S and an allocation α. A* feasible binding β *is a binding that satisfies*

1. *Each activated edge $e \in \beta$ starts and ends at an activated node, i.e.*

   $$\forall e = (v, \tilde{v}) \in \beta \ : \ v, \tilde{v} \in \alpha$$

2. *For each activated node $v \in \alpha_V$ with $v \in V_i$, $1 \le i < d$ exactly one outgoing edge $e \in V_M$ is activated, i.e.*

   $$| \{e \in \beta \ | e = (v, \tilde{v}), \tilde{v} \in V_{i+1}\} \ | = 1$$

3. *For each activated edge $e = (v_i, v_j) \in \alpha_E$ with $e \in E_i$, $1 \le i < d$*

   - *either both operations are mapped onto the same node, i.e.*

     $$\tilde{v}_i = \tilde{v}_j \quad with \quad (v_i, \tilde{v}_i), (v_j, \tilde{v}_j) \in \beta$$

- *or there exists an activated edge* $\tilde{e} = (\tilde{v}_i, \tilde{v}_j) \in \alpha_E$ *with* $\tilde{e} \in E_{i+1}$ *to handle the communication associated with edge* e*, i.e.*

$$(\tilde{v}_i, \tilde{v}_j) \in \alpha_E \quad \textit{with} \quad (v_i, \tilde{v}_i), (v_j, \tilde{v}_j) \in \beta$$

The proposed synthesis procedure simultaneously determines allocation and binding. On the other hand it is useful to determine the set of feasible allocations and feasible bindings in order to restrict the search space.

Definition 7.7 (Feasible Allocation) *A* feasible allocation α *is an allocation that allows at least one feasible binding* β*.*

However, the next theorem shows that the calculation of a feasible binding and therefore the test of an allocation for feasibility is hard. This result will influence the coding of allocation and binding in the Evolutionary Algorithm.

Theorem 7.1 *The determination of a feasible binding is NP-complete.*

Proof: Obviously, the test of a binding for feasibility can be done in polynomial time, see Def. 7.6. Now, the Boolean Satisfiability problem will be polynomially reduced to the determination of a feasible binding. The Boolean Satisfiability problem is known to be NP-complete (see, for example, [Garey and Johnson, 1979]).

At first, suppose that Boolean variables are associated to the mapping edges $e \in E_M$. Moreover, if $e \in \beta$, then the corresponding variable is 1 (true), if $e \notin \beta$, the variable is 0 (false). Now, it is shown how to construct a specification graph from a given set of clauses such that a feasible binding yields values for the Boolean variables which make all clauses true iff there is a solution.

Instead of using $(a \vee b \vee \cdots) = 1$ for the clauses equations of the dual form $(c \wedge d \wedge \cdots) = 0$ are transformed into a specification graph. The specification graph contains two dependence graphs G_1 and G_2 only and all nodes and edges of G_1 and G_2 are allocated. The necessary components for building the specification graph are given in Fig. 7.5. Any set of clauses can now be represented as follows:

1. For any variable in the set of clauses a structure like in Fig. 7.5a is used.

2. For any clause with n literals use $n-1$ times the structure in Fig. 7.5d concatenated and linked to the variable structures constructed in step 1 by structures like Fig. 7.5b.

3. Use structures as in Fig. 7.5c to guarantee that all clauses are true.

It can be seen that there is a one-to-one correspondence between a solution to a given satisfiability problem and a feasible binding in the corresponding specification graph.

□

Finally, it is necessary to define a *schedule*. For this purpose, the execution time $delay(v, \beta)$ of an operation is associated to node v of a problem graph G_P. In order to be as general as possible at this point, it is supposed that the execution time depends on a particular binding β. For example, the time delay of an operation may depend on the resource it is bound to.

Definition 7.8 (Schedule) *Given specification G_S containing a problem graph $G_1 = G_P$, a feasible binding β, and a function which determines the execution time $delay(v, \beta) \in \mathbb{Z}^+$ of a node $v \in V_P$. A* schedule *is a function $\tau : V_P \mapsto \mathbb{Z}^+$ that satisfies for all edges $e = (v_i, v_j) \in E_P$:*

$$\tau(v_j) \geq \tau(v_i) + delay(v_i, \beta)$$

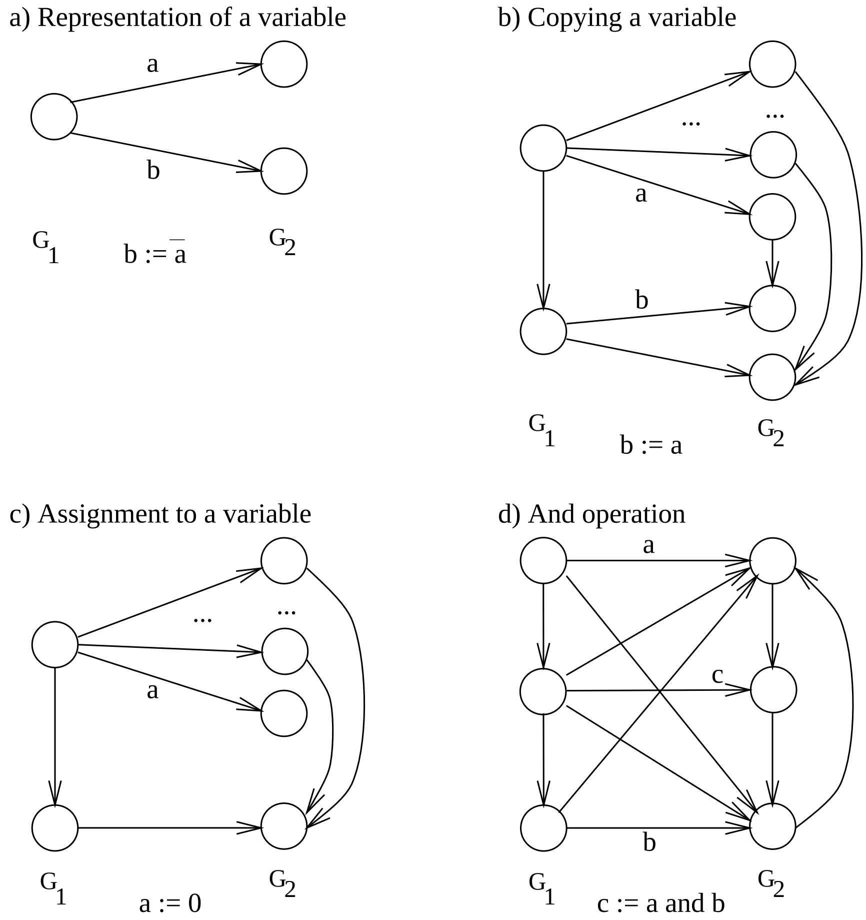


Figure 7.5: *Components for the transformation of Boolean expressions to a specification graph.*

$\tau(v_i)$ may be interpreted as the start time of the operation of node $v_i \in V_P$. For example, the delay values of communication nodes might be considered as the time units necessary to transfer the associated data using the node bound to it. Usually, these values depend not only on the amount of data transferred but also on the capacity of the resource, for example the bus width and the bus transfer rate. Therefore, the delay may depend on the actual binding. The special case of an *internal communication* is described in the next example.

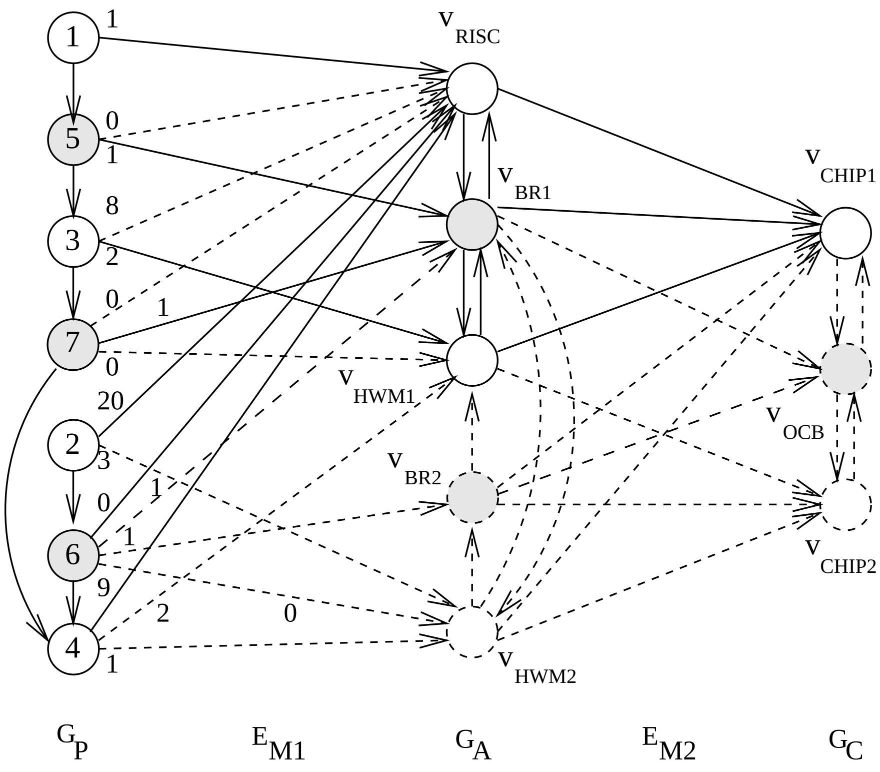


Figure 7.6: *An example of an implementation from the specification given in Fig. 7.4.*

Example 7.5 *Consider the case that the delay values of a node $v_i \in V_P$ only depend on the binding of that particular node. Then the delay values can be associated with the edges E_{M1} (see Fig. 7.6). All execution times of all operations on different resources are shown. For example, operation v_3 takes 8 time units if executed on the RISC (v_{RISC}) but 2 time units if mapped to the hardware module HWM1. Note that internal communications (like the mapping of v_5 to v_{RISC}) are modeled to take zero time. The definition of a feasible binding ensures that internal communication is only possible between nodes bound to the same resource.*

A value of interest in connection with a schedule is the execution time of the whole task graph, called *latency* or *makespan*.

Definition 7.9 (Latency) *The* latency *(*makespan*) L of a schedule τ and a binding β of a specification graph G_S is the time between the earliest starting time and the latest finishing time of any node $v \in V_P$, i.e.,*

$$L(\tau, \beta) = max_{v \in V_P} \{\tau(v) + delay(v, \beta)\} - min_{v \in V_P} \tau(v) \tag{7.1}$$

For convenience it is assumed without loss of generality that the starting time of the first operation is zero, i.e., $min_{v \in V_P} \tau(v) = 0$.

Definition 7.10 (Implementation) *Given a specification graph G_S a* (valid) implementation *is a triple (α, β, τ) where α is an allocation, β is a feasible binding, and τ is a schedule.*

Example 7.6 *Fig. 7.6 shows an implementation of the specification from Fig. 7.4. Unallocated nodes and edges are shown dashed, including edges $e \in E_M$ that are not activated. The allocation of nodes is $\alpha_V = \{v_{RISC}, v_{HWM1}, v_{BR1}, v_{CHIP1}\}$ and the binding is $\beta = \{(v_1, v_{RISC}), (v_2, v_{RISC}), (v_3, v_{RISC}), (v_4, v_{HWM1}), (v_5, v_{RISC}), (v_6, v_{BR1}), (v_7, v_{BR1}), (v_{RISC}, v_{CHIP1}), (v_{BR1}, v_{CHIP1}), (v_{HWM1}, v_{CHIP1})\}$. This means that all architecture components are bound to CHIP1.*

Note that communication modeled by v_6 can be handled by the functional resource v_{RISC} as both predecessor node (v_2) and successor node (v_4) are mapped to resource v_{RISC}. A schedule is $\tau(v_1) = 0$, $\tau(v_2) = 1$, $\tau(v_3) = 2$, $\tau(v_4) = 21$, $\tau(v_5) = 1$, $\tau(v_6) = 21$, $\tau(v_7) = 4$.

7.3 System Synthesis as Optimization Problem

With the model introduced above the task of system synthesis can be formulated an optimization problem.

Definition 7.11 (System Synthesis) *The task of* system synthesis *is the following optimization problem:*

minimize $h(\alpha, \beta, \tau)$, subject to

α *is a feasible allocation,*
β *is a feasible binding,*
τ *is a schedule, and*
$g_i(\alpha, \beta, \tau) \geq 0, \ i \in \{1, \ldots, q\}$.

The constraints on α, β and τ define the set of valid implementations. Additionally, there are functions $g_i, i = 1, \ldots, q$, that together with the objective function h describe the optimization goal.

Example 7.7 *The specification graph G_S may consist only of a problem graph G_P and an architecture graph G_A. Consider the task of latency minimization under resource constraints, i.e., an implementation is searched that is as fast as possible but does not exceed certain cost c_{max}. To this end, a function $V_A \mapsto \mathbb{Z}^+$ is given which describes the basic cost $c_b(\tilde{v})$ that arises if resource $\tilde{v} \in V_A$ is realized, i.e., $\tilde{v} \in \alpha$. The limit in costs is expressed in a constraint $g_1(\alpha, \beta, \tau) = c_{max} - \sum_{\tilde{v} \in \alpha} c_b(\tilde{v})$. The corresponding objective function may be $h(\alpha, \beta, \tau) = L(\tau, \beta)$.*

The objective function may be arbitrary complex and reflect the specific optimization goal. Likewise, the additional constraints g_i can be used to reduce the number of potential implementations.

7.4 Expressiveness of the Model

So far, the basic ideas of modeling the task of system synthesis have been introduced. This model has been shown to be concise, useful, and general as it can handle a broad range of mapping problems. In this section the capabilities of the model are examined. Up to now, only the alternative selection of resources has been introduced. However, the model also allows the concurrent selection of alternative resources for a set of operations in addition to the "complement" selection of resources for operations.

7.4.1 Alternative Resources for an Operation Set

When specifying architectures it is in many cases necessary to be able to express that a group of nodes are to be mapped to the same resource from several possible resources. Consider, for example, an architecture that consists of different RISC processors with different characteristics (execution times, memory demands, etc.). Furthermore, a set of operations is given that can be executed by any of these processors. However, only exactly one of the processors shall be used, i.e., if one operation of the set is bound to a particular processor all other operations have to bound to the same processor.

To express this condition within the model a *selection node* v_S is introduced. All operation nodes depend on this node and the selection node can be bound to any of the resource nodes. Due to the definition of a feasible binding this forces all nodes to be bound to the same resource. An optional arc inscription for the time delay of the selection node has to be set to zero. This guarantees that the new node has no impact on the schedule.

Example 7.8 *Consider two functional nodes* v_1 *and* v_2 *that can be bound either to resource* v_A *or* v_B *(Fig. 7.7). The concurrent selection for the same resource is ensured by in introducing the node* v_S *(shaded) and some additional arcs (dashed in Fig. 7.7). Only two feasible bindings exist:*

- *If node* v_1 *is bound to resource* v_A *the selection node* v_S *has also to be bound to resource* v_A *(according to the edge from node* v_S

to node v_1). The binding of node v_S then forces node v_2 to be bound to resource v_A. Hence all nodes are bound to the same resource v_A.

- *If node v_1 is bound to resource v_B the selection node v_S has also to be bound to resource v_B (according to the edge from node v_S to node v_1). The binding of node v_S then forces node v_2 to be bound to resource v_B. Hence all nodes are bound to the same resource v_B.*

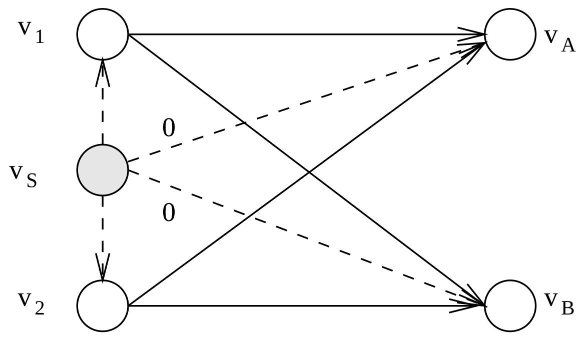


Figure 7.7: *Modeling the condition that two operations must be bound to the same resource.*

This scheme can easily be generalized to an arbitrary number of nodes and resources.

7.4.2 Complement Resources for two Operations

Similar to the preceding paragraph a specification may be of interest that avoids two nodes being bound to the same resource. The basic idea is introduced in the following example.

Example 7.9 *Consider the same graph as in Example 7.8, i.e., two functional nodes v_1 and v_2 and two resource v_A or v_B. To model the alternative mapping of the nodes to resources three additional nodes* *(shaded) and some addition arcs (dashed) are introduced (see Fig. 7.8). Only two feasible bindings exist*

- *If node v_1 is bound to resource v_A the selection node v_S has to be bound to the dummy resource v_{1A2B}, in order to fulfill the dependence constraint expressed by the edge from v_1 to v_S. This forces the node v_2 to be mapped to resource v_B. Dummy resource v_{1B2A} is not used (allocated).*
- *If node v_1 is bound to resource v_B the selection node v_S has to be bound to the dummy resource v_{1B2A}. This forces the node v_2 to be mapped to resource v_A. Dummy resource v_{1A2B} is not used (allocated).*

The arc weights of $-\infty$ ensure that the sequence of nodes v_1 and v_2 due to the insertion of the node v_S has no influence on the schedule.

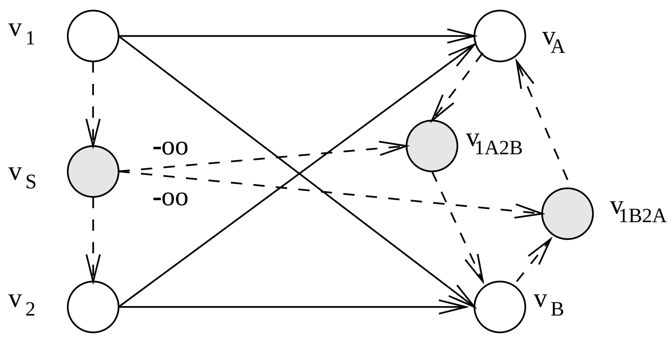


Figure 7.8: *Modeling the condition that two operations have to be bound to different resources.*

7.5 Examples of Model Refinement

In the following examples of model refinements are discussed that may be necessary to model a certain design style or a particular design goal.

These refinements will consider the modeling of program memory, resource sharing and chip boundaries.

7.5.1 Model for Resource Sharing

A simple cost model was given in Example 7.7 that only considers the cost of realizing a certain hardware module. In the following a more detailed model is introduced that takes program size or chip area into account. For simplicity it is assumed that a specification only consists of a problem graph G_P and an architecture graph G_A.

Definition 7.12 (Cost) *The function $c_b : V_A \mapsto \mathbb{Z}^+$ describes the basic cost $c_b(\tilde{v})$ that arises if resource $\tilde{v} \in V_A$ is realized.*

The following definition reflects additional costs that occurs if more than one functionality is mapped to the same resource. These costs can for example model program memory or chip area specific to each different task.

Definition 7.13 (Addcost) *The function $c_a : E_M \mapsto \mathbb{Z}^+$ describes for each edge $e = (v, \tilde{v}) \in E_M$ the additional cost of implementing node $v \in V_P$ on resource $\tilde{v} \in V_A$.*

However, there are often different tasks that can share the same program memory or share the same hardware area. In this case summing up additional costs is not appropriate. As a remedy, so-called *types* are defined.

Definition 7.14 (Types) *The node set V_P is partitioned into* types *$\Theta \in \mathcal{T}$, i.e., each node has exactly one type from the set of types.*

The additional costs of all nodes of the same type Θ mapped onto the same resource $\tilde{v} \in V_A$ only cause additional costs

$$c_t(\Theta, \tilde{v}) = \max\{c_a(e) | v \in \Theta : e = (v, \tilde{v}) \in \beta\}$$

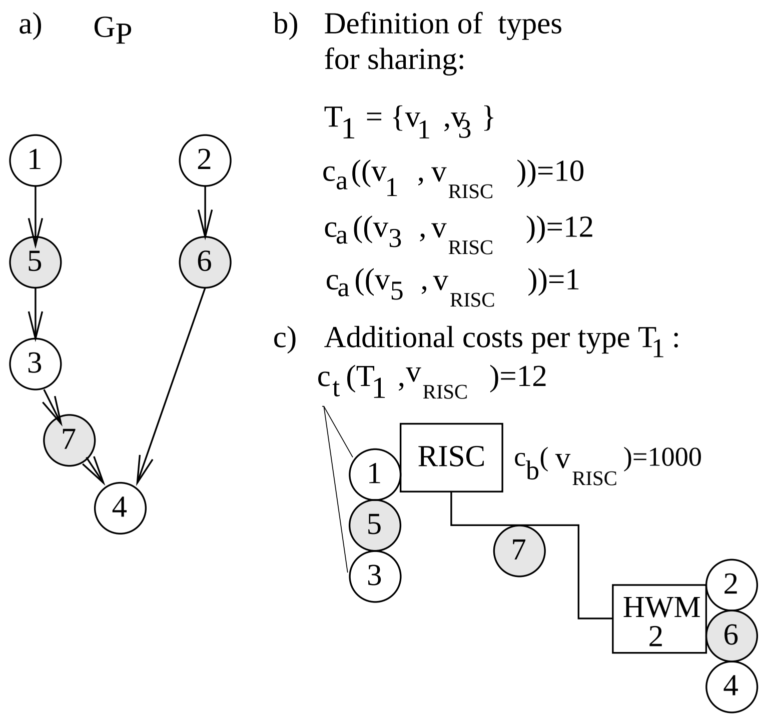


Figure 7.9: *Modeling of resource sharing (program memory or hardware modules) by definition of types for sharing.*

With this definitions a more realistic modeling of the cost of an implementation is possible. In particular, the cost $c(\alpha, \beta)$ of an implementation is specified by

$$c(\alpha, \beta) = \sum_{\tilde{v} \in \alpha \cap V_A} c_h(\tilde{v}, \beta)$$

where $c_h(\tilde{v}, \beta)$ specifies the cost for each allocated hardware component, i.e.,

$$c_h(\tilde{v}, \beta) = c_b(\tilde{v}) + \sum_{\Theta \in \mathcal{T} : (v, \tilde{v}) \in \beta \wedge v \in \Theta} c_t(\Theta, \tilde{v})$$

This way, a constraint on a resource $v \in \alpha \cap V_A$ can be specified that limits the maximum cost for this resource to $c_{max,v}$, and thereby limits the maximum program size or chip area:

$$g_v = c_{max,v} - c_h(\tilde{v}, \beta)$$

Similar, a constraint on the total cost of the implementation can be expressed by

$$g = c_{max} - c(\alpha, \beta)$$

Example 7.10 *Fig. 7.9 demonstrates the use of types to model costs of shared resources. One type is defined in Fig. 7.9b). $\Theta_1 = \{v_1, v_3\}$, all other nodes belong to a dedicated type, i.e., $\mathcal{T} = \{\Theta, \{v_2\}, \{v_4\}, \{v_5\}, \{v_6\}, \{v_7\}\}$. As an example the calculation of the cost of the resource RISC ($c_1(v_{RISC}, \beta)$) is carried out for the particular binding β given by the implementation in Fig. 7.9c).*

Let the additional costs for implementing node v_1 on the RISC resource be $c_a((v_1, v_{RISC})) = 10$ and that of v_3 on the RISC resource be $c_a((v_3, v_{RISC})) = 12$. As both nodes are implemented by the same resource the additional costs are reflected by the type costs, i.e.,

$$\begin{aligned} c_t(\Theta_1, v_{RISC}) &= \max\{c_a((v_1, v_{RISC})), c_a((v_3, v_{RISC}))\} \\ &= \max\{10, 12\} = 12 \end{aligned}$$

The additional cost of implementing operation v_5 is $c_a(v_5) = 1$. As there is no other node that belongs to the same type as v_5 the type cost calculate to

$$c_t(\{v_5\}, v_{RISC}) = \max\{c_a((v_5, v_{RISC}))\} = \max\{1\} = 1$$

Furthermore, assume the basic cost for realizing RISC is $c_b(v_{RISC}) = 1000$ then the total cost of resource RISC is

$$\begin{aligned} c_1(v_{RISC}) &= c_b(v_{RISC}) + \sum_{\Theta \in \{\Theta_1, \{v_5\}\}} c_t(\Theta, v_{RISC}) \\ &= c_b(v_{RISC}) + c_t(\Theta_1, v_{RISC}) + c_t(\{v_5\}, v_{RISC}) \\ &= 1000 + 12 + 1 = 1013 \end{aligned}$$

7.5.2 Model for Chip Boundaries

Module and bus costs strongly depend on whether a system is implemented on a single chip or on a board using multiple chips. Costs and delay values depend on the selection of a layout macro cell in case of single-chip design or on the package and die specification in case of a multiple-chip design.

The selection of a chip-set can easily obtained by introducing a dedicated level of abstraction, the chip graph G_C (as in example 7.3). It is thereby assumed that a chip is only a container for all resources that are mapped to it and that all these resources operate concurrently. Similarly an off-chip bus (as in example 7.3) is only a bundle of wires. The possible sequentialization of operations or communication is modeled by mapping operations in G_P to resources in G_A. Therefore, the specification graph G_S consists of a problem graph G_P, an architecture graph G_A and a chip graph G_C.

As an example modeling the number of pins to specify an upper bound on the number of pins for a chip is made.

The necessary number of pins can be split into a fixed part pin_c (e.g. for power supply) and a part that depends on the size of off-chip communication. Let $V_B(\bar{v}, \beta)$ be the set of resources $\tilde{v} \in V_A$ that handle a communication of chip $\bar{v} \in V_C$ to the outside, i.e.,

$$\begin{aligned} V_B(\bar{v}, \beta) = \{\tilde{u} \in V_A | \quad & \exists \tilde{v} \in V_A, \bar{u} \in V_C \text{ with} \\ & (\tilde{u}, \bar{u}), (\tilde{v}, \bar{v}) \in \beta \wedge \\ & ((\bar{u}, \bar{v}), (\tilde{u}, \tilde{v}) \in \alpha \vee (\bar{v}, \bar{u}), (\tilde{v}, \tilde{u}) \in \alpha)\} \end{aligned}$$

Assume furthermore a function $width : V_A \mapsto \mathbb{Z}^+$ that assigns to each node of the architecture graph the width of a bus that is modeled by this node. Then the number of pins required for a chip $\bar{v}$ is given by

$$pin(\bar{v}) = pin_c(\bar{v}) + \sum_{\tilde{v} \in V_B(\bar{v})} width(\tilde{v})$$

The limitation of pins to $pin_{max}(\bar{v})$ of a chip $\bar{v}$ can now be expressed by the following constraint:

$$g = pin_{max}(\bar{v}) - pin(\bar{v})$$

Example 7.11 *Consider the specification of example 7.4. The width for the bus resources may be* $width(v_{BR1}) = width(v_{BR2}) = 16$*, the number of the fixed pins of CHIP1* $pin_c = 2$*. Furthermore, assume a constraint that limits the number of pins of CHIP1 to 20, i.e.,* $g = 20 - pin(v_{CHIP1})$*.*

Consider now the implementation shown in Fig. 7.10a) with binding β_a*. For sake of clarity, only the architecture graph* G_A *and the chip graph* G_C *are shown and the problem graph* G_P *is omitted. The set of bus resources that handle communication to the outside of CHIP1 is* $V_B(v_{CHIP1}, \beta_a) = \{v_{BR1}, v_{BR2}\}$*. The pins of CHIP1 then calculate to*

$$\begin{aligned} pin(v_{CHIP1}) &= pin_c(v_{CHIP1}) + \sum_{\tilde{v} \in V_B(v_{CHIP1}, \beta_a)} width(\tilde{v}) \\ &= 2 + width(v_{BR1}) + width(v_{BR2}) = 2 + 16 + 16 = 34 \end{aligned}$$

Hence, the implementation in Fig. 7.10a) violates the constraint as $g = 20 - pin(v_{CHIP1}) = -14 < 0$*.*

On the other hand, the implementation in Fig. 7.10b) with binding β_b *does satisfy this constraint. As* $V_B(v_{CHIP1}, \beta_b) = \{v_{BR2}\}$ *the pins of CHIP1 calculate to*

$$\begin{aligned} pin(v_{CHIP1}) &= pin_c(v_{CHIP1}) + \sum_{\tilde{v} \in V_B(v_{CHIP1}, \beta_b)} width(\tilde{v}) \\ &= 2 + width(v_{BR2}) = 2 + 16 = 18, \end{aligned}$$

yielding $g = 20 - 18 = 2 \geq 0$*.*

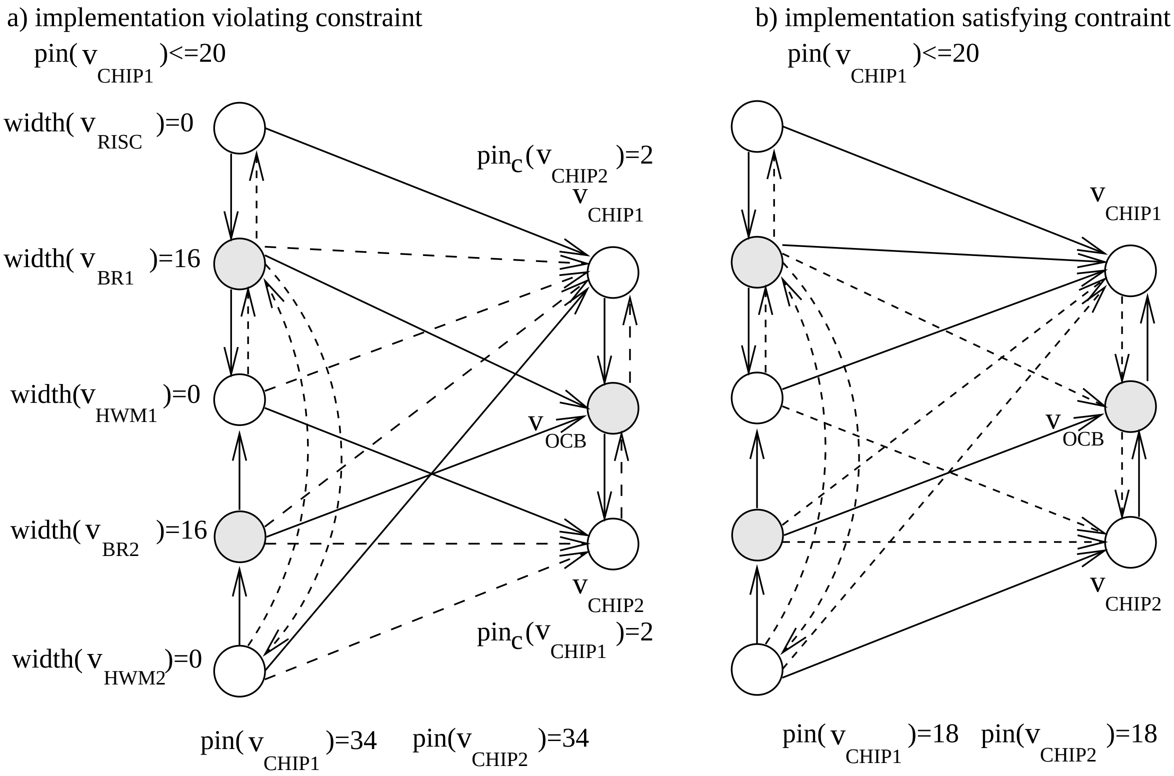


Figure 7.10: *Two implementations from the specification of Fig. 7.4: a) violating the pinlimit constraint* $pin(v_{CHIP1}) \leq 20$ *and b) satisfying the pinlimit constraint* $pin(v_{CHIP1}) \leq 20$.

Chapter 8

Solving the System Synthesis Problem

In this chapter the application of an Evolutionary Algorithm is described to solve the problem of system synthesis as specified by Definition 7.11.

Evolutionary Algorithms are especially suited for the system synthesis problem, because:

- they work well on problems with large and non-convex search spaces,
- they have moderate run-time due to their heuristic character and,
- they work on a population of implementations, enabling design space exploration.

In the proposed approach, the Evolutionary Algorithm is used to determine the allocation and the binding. As the search space for these tasks is large and discrete and the determination of a feasible binding is NP-complete (see Theorem 7.1). Hence, for reasonable problem sizes exact methods are intractable and EAs are known to work well on such problems. Furthermore, the inherent parallelism of EAs allows design space exploration during a single optimization run. As there are good heuristics available for the scheduling problem it is not necessary to encumber the EA with this task. The schedule for the fixed architecture obtained by the EA is computed by a heuristic scheduler. This division of work is depicted in Fig. 8.1.

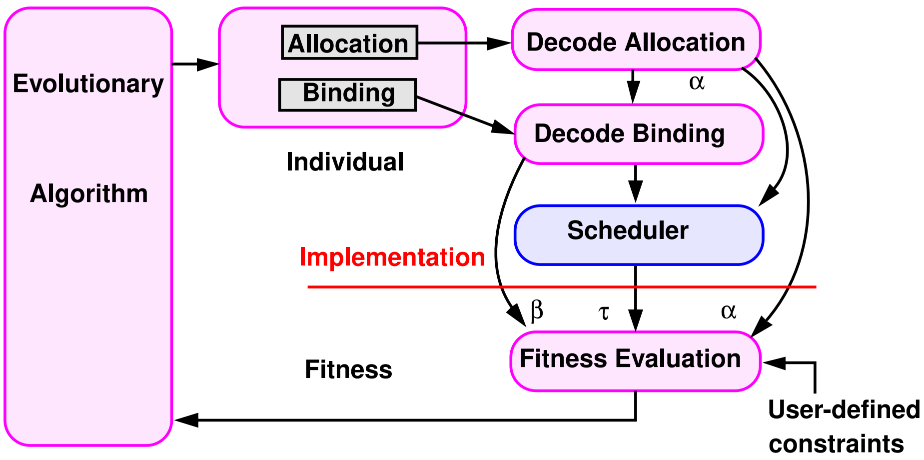


Figure 8.1: *The decoding of an individual to an implementation.*

In the remainder of this chapter the coding of implementations as individuals is described, followed by the introduction of the fitness function f. The last section introduces a new scheduling heuristic for iterative scheduling used in collaboration with the EA.

8.1 Coding of Implementations

To obtain a meaningful coding the question of how to handle infeasible allocations and infeasible bindings suggested by the EA are handled must be addressed. Obviously, if allocations and bindings are randomly chosen, many will not be infeasible. In general, all methods of constrained optimization discussed in Chapter 3.3 can be applied, such as constraint-handling based on penalty functions or constraint-handling by repair. The exclusive use of the penalty function approach is not suited for this problem as depending on the specification, many possible allocations and bindings might be infeasible. Furthermore, the infeasibility of allocation or binding is a Boolean event, i.e. a solution is not "more" or "less" feasible. Hence, the optimization would result in a needle-in-the-haystack search and the EA will perform badly. On the other hand, one could "repair" invalid allocations and bindings with some mechanism and incorporate domain knowledge in these repair mechanisms. But as the determination of a feasible allocation or binding is NP-complete, this would result in solving a NP-complete task for every individual to be repaired. These considerations lead to the following compromise: The randomly generated allocations of the EA are partially repaired using a heuristic. Possible complications detected later on during the calculation of the binding will be penalized.

A specification graph G_S may consist of d subgraphs $G_i, 1 \leq i \leq d$ corresponding to $d-1$ mapping tasks. In order to handle these mapping tasks the allocation and binding of levels is carried out sequentially from level 1 to level $d-1$ (see Algorithm 13). In each level three steps are executed:

- First, the allocation of nodes V_{i+1} is decoded from the individual and repaired with a simple heuristic (the function *node_allocation()*),
- next the binding of the edges $e \in E_{Mi}$ is performed (the function *binding()*), and
- finally the allocation is updated in order to eliminate unnecessary nodes $v \in V_{i+1}$ from the allocation and to add all necessary edges $e \in E_{i+1}$ to the allocation (the function *update_allocation()*).

One iteration of the loop results in a feasible allocation and binding of nodes and edges of G_i to nodes and edges of G_{i+1}. If no feasible binding is found the decoding of the individual is aborted.

Algorithm 13: (Decoding)

Input: Individual J consisting of allocations $alloc_i$,
repair allocation priority lists $\mathtt{L}_{Ri}$,
binding order lists $\mathtt{L}_{Oi}$ and binding
priority lists $\mathtt{L}_{Bi}(v)$, for all $1 \leq i < d$.
Output: The allocation α and the binding β
if both are feasible, $(\{\}, \{\})$ if no feasible
binding is represented by the individual J

decode(J):
  $\alpha \leftarrow V_1 \cup E_1$
  $\beta \leftarrow \{\}$
  for $i \leftarrow 1$ **to** d-1 **do**
    $\bar{\alpha} \leftarrow$ node_allocation($alloc_i(J), \mathtt{L}_{Ri}(J)$)
    $\bar{\beta} \leftarrow$ binding($\mathtt{L}_{Bi}(J), \mathtt{L}_{Oi}(J), \bar{\alpha}$)
    if $\bar{\beta} = \{\}$
      return $(\{\}, \{\})$
    endif
    $\beta \leftarrow \beta \cup \bar{\beta}$
    $\alpha \leftarrow \alpha \cup$ update_allocation($\bar{\alpha}, \bar{\beta}$)
  od
  return (α, β)

In the following the three functions node_allocation(), binding(), and update_allocation() are explained in detail.

8.1.1 The Function node_allocation()

The allocation of nodes is directly encoded in the individual, i.e., for each node $v \in V_{i+1}$ there exists a vector $alloc_i$ that codes the activation of v, i.e., $a(v) = alloc_i[v]$. This simple coding might result in a many infeasible allocations, e.g. 77% of randomly generated allocations of the specification in Example 7.4 (for the first mapping level from G_P to G_A) lead to infeasible allocations. As a result a simple repair heuristic is applied. This heuristic only adds new nodes $v \in V_{i+1}$ to the allocation and reflects a simple case of infeasibility: Consider the set $V_{Bi} \subseteq V_i$ that contains all nodes that can not be executed because not a single corresponding resource node is allocated,i.e., $V_{Bi} = \{v \in V_i | (v, \tilde{v}) \in E_{Mi} : a(\tilde{v}) = 0\}$. To make the allocation feasible (in this sense) for each $v \in V_{Bi}$, at most one $\tilde{v} \in V_{i+1}$ is added to the allocation, until a feasible solution is achieved. This way, the number of infeasible allocations for the first mapping level of Example 7.4 could be reduced to 6.5%.

The order in which the new resources are added greatly influences the resulting allocation. For example, one could be interested in an allocation with minimal costs or with maximum performance. As this depends on the optimization goal expressed in the objective function h the order new resources are added to the allocation by the heuristic should automatically be adapted. This is achieved by an *repair allocation priority list* $\mathtt{L}_{Ri}$ coded for each mapping level in the individual. In this list, all nodes $v \in V_{i+1}$ are present and the order in the list determines the order the nodes are added to the allocation. This list also undergoes genetic operators like crossover and mutation and can therefore be adapted by the Evolutionary Algorithm.

In summary, Algorithm 14 is obtained to decode the allocation of nodes.

```
Algorithm 14: (Allocation)

Input: The allocation alloc_i and repair allocation
   priority list L_Ri of individual J
Output: The allocation α

node_allocation(alloc_i, L_Ri):
   α ← {}
   forall ṽ ∈ V_{i+1} do
      if alloc_i[ṽ] = 1
         α ← α ∪ {ṽ}
      endif
   od
   V_B ← not_bindable_nodes(α)
   ṽ_r ← first(L_Ri)
   while (V_B ≠ {}) do
      if V_B ≠ not_bindable_nodes(α ∪ {ṽ_r})
         α ← α ∪ {ṽ_r}
         V_B ← not_bindable_nodes(α)
      endif
      ṽ_r ← next(L_Ri)
   od
   return α
```

Example 8.1 *Consider the specification of Example 7.4. In Fig. 8.2 the allocation information for the first mapping level as stored in the individual is shown on the left. The direct decoding of the allocation string yields the allocation* $\alpha = \{v_{BR1}, v_{HWM1}\}$*. The allocation is not valid as no allocated resource for executing* $v_1, v_2 \in G_P$ *exists. This allocation is then repaired using the repair allocation priority list.* v_{HWM1} *and* v_{BR1} *are already in the allocation. The allocation of* v_{RISC} *resolves the conflict for both* v_1 *or* v_2*, hence it is allocated. The rest of the list is then ignored, as no node with a conflict remains.*

Note that a different repair allocation priority list may result in a different allocation. If the entries of v_{RISC} and v_{HWM2} are swapped in the list, both v_{HWM2} and v_{RISC} would be allocated as the allocation of v_{HWM2} only solves the conflict for v_2 but not for v_1. This would code a more "lavish" allocation.

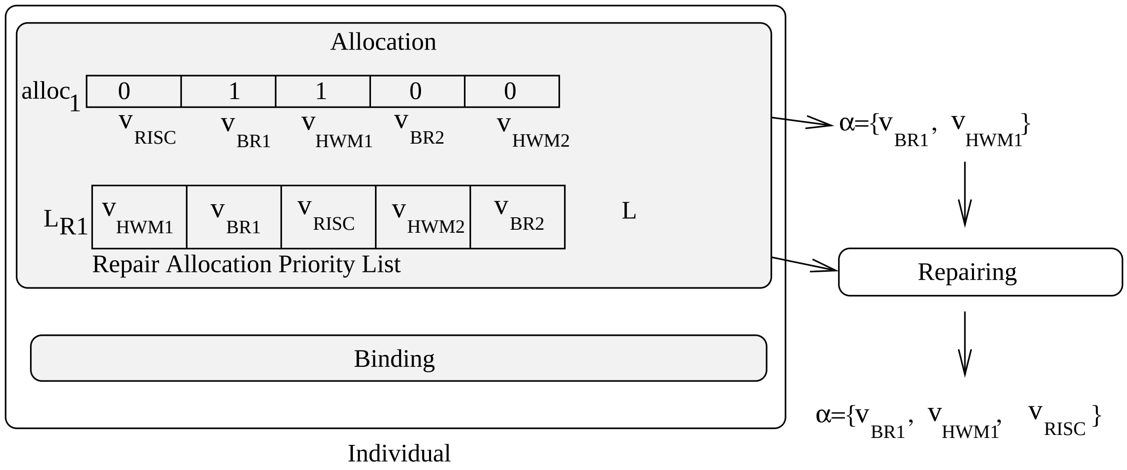


Figure 8.2: *The decoding of an allocation.*

8.1.2 The Function binding()

A binding is obtained by activating exactly one edge $e \in E_{Mi}$ adjacent to an allocated node v for each $v \in V_i$. The problem of coding the binding lies in the strong inter-dependence of the binding and the current allocation. As crossover or mutation might change the allocation, a directly encoded binding could be meaningless for a different allocation. Hence, a coding of the binding is of interest that can be interpreted *independently* of the allocation. This is achieved in the following way:

```
Algorithm 15: (Binding)

Input: Binding priority lists L_Bi(v) ∀v ∈ V_i,
    binding order list L_Oi, allocation α of individual J.
Output: The binding β, or {} if no
    feasible binding was decoded.

binding(L_Bi, L_Oi, α_i):
    β ← {}
    forall_listelements u ∈ L_Oi ∩ α do
        e' ←nil
        forall_listelements e ∈ L_B(u) ∩ α do
            if is_valid_binding(e, β, α )
                e' ← e
                break
            endif
        od
        if e' = nil
            return {}
        else
            β ← β ∪ {e'}
        endif
    od
    return β
```

For each node $v \in V_i$ a list is coded that contains all adjacent edges $e \in E_{Mi}$ of v. This list is seen as a priority list and the first edge e_k with $e_k = (v, \tilde{v})$ that gives a feasible binding is included in the binding, i.e., $a(e_k) := 1$. The feasibility test is directly related to the definition of a feasible binding (Definition 7.6). Details are given in the two algorithms (Algorithm 15 and 16). Note that it is possible, that no feasible binding is specified by the individual. In this case, β is the empty set, no schedule can be computed, and the individual will be given a penalty value as fitness value.

```
Algorithm 16: (Check for Valid Binding)

Input: The edge e = (v, ṽ) ∈ V_Mi,
    the already computed binding β and allocation α
Output: true if e can be included to the binding false else.

is_valid_binding(e = (v, ṽ), β, α):
    forall ê = (w, v) ∈ E_i ∩ α do
        if (w, w̃) ∈ β and ṽ ≠ w̃ and (w̃, ṽ) ∉ E_{i+1}
            return false
        endif
    od
    forall ê = (v, w) ∈ E_i ∩ α do
        if (w, w̃) ∈ β and ṽ ≠ w̃ and (ṽ, w̃) ∉ E_{i+1}
            return false
        endif
    od
    return true
```

Example 8.2 *Fig. 8.3 shows an example of a binding as it is coded in the individual for the first mapping level of the specification of Example 7.4 (Fig. 7.4). The binding order specified by the list* $\mathtt{L}_{O1}$ *is* v_1*,* v_4*,* v_2*,* v_3*,* v_6*,* v_5*,* v_7*. The binding priority lists for all nodes are also shown. For example, the priority list for node* v_6 *implies that* v_6 *is bound to the resource BR2 (point-to-point bus). If this is not possible, it should be bound to HWM2 and if this is also not possible it should be bound to BR1 (shared bus). As the allocation from Example 8.1 does not contain HWM2 and BR2, this node is finally bound to BR1.*

8.1.3 The Function update_allocation()

In this function nodes of the allocation that are not used will be removed from the allocation. Furthermore, all edges $e \in E_{i+1}$ are

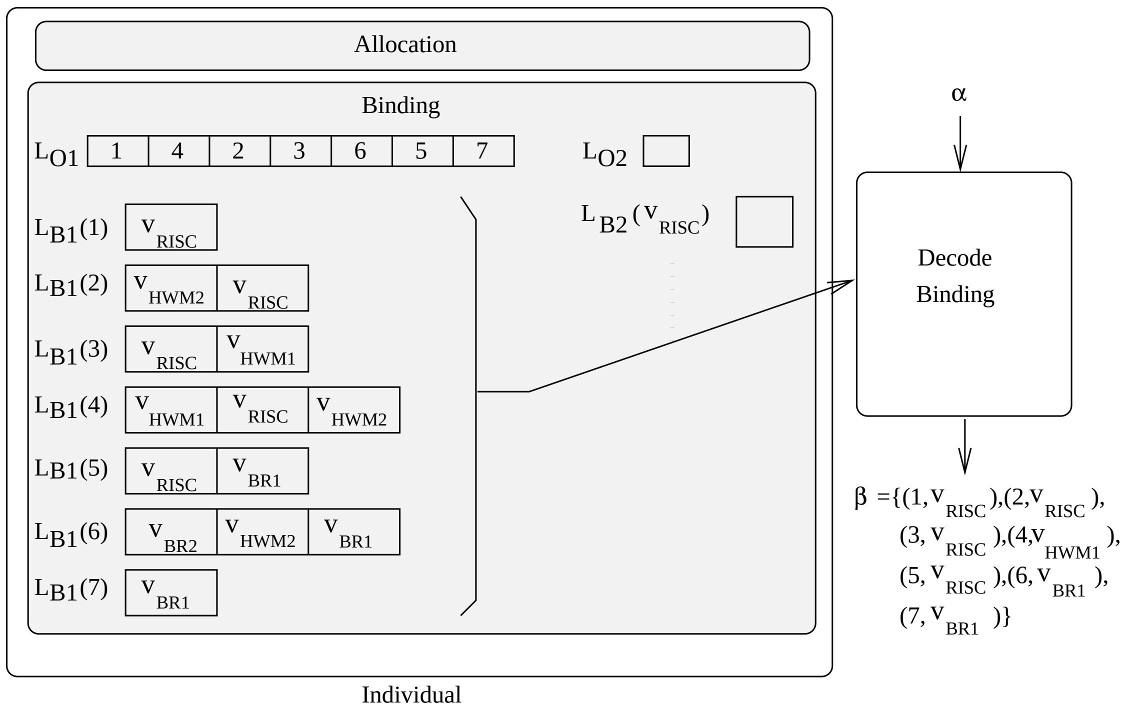


Figure 8.3: *An example of the coding of a binding and the resulting binding using the allocation of Fig. 8.2. The obtained architecture is the one depicted in Fig. 7.6 of Example 7.6.*

added to the allocation that are necessary to obtain a feasible binding (see Algorithm 17).

Example 8.3 *Consider the allocation and binding of Example 8.1 and 8.2, respectively. No node $\tilde{v} \in V_A$ is removed from the allocation, as no unnecessary nodes have been allocated. The computing of the edges $\tilde{v} \in E_A$ leads to the following set that is added to the allocation: $\alpha_E = \{(v_{RISC}, v_{BR1}), (v_{BR1}, v_{RISC}), (v_{HWM1}, v_{BR1}), (v_{BR1}, v_{HWM1})\}$. For example the edge $e = (v_3, v_7) \in E_P$ results in the allocation of edge $(v_{BR1}, v_{RISC}) \in E_A$ as node v_3 is bound to resource v_{RISC} and node v_7 is bound to resource v_{BR1}.*

These algorithms describe in detail the transformation of an individual into an allocation α and a binding β. This specific architecture is used as input to the scheduler to obtain the complete implementation.

Algorithm 17: (Update Allocation)

Input: Current allocation α and binding β
of the mapping level i
Output: Updated allocation, i.e., the allocation where unnecessary nodes are removed from and necessary edges are added to the allocation.

update_allocation(α, β):
  forall $\tilde{v} \in V_{i+1} \cap \alpha$ **do**
    if $\nexists v \in V_i : (v, \tilde{v}) \in \beta$
      $\alpha \leftarrow \alpha \setminus \{\tilde{v}\}$
    endif
  od
  forall $e = (v, u) \in E_i \cap \alpha$ **do**
    $\tilde{e} \leftarrow (\tilde{u}, \tilde{v})$ with $(v, \tilde{v}), (u, \tilde{u}) \in \beta$
    $\alpha \leftarrow \alpha \cup \{\tilde{e}\}$
  od
  return α

8.2 Scheduling

In this section the tasks of a scheduler are briefly described and a new simple heuristic to obtain iterative schedules for acyclic graphs is presented. As the allocation and binding is fixed when the scheduler is invoked, only the task of latency minimization under resource constraints is discussed. Furthermore, only non-cyclic problem graphs are considered.

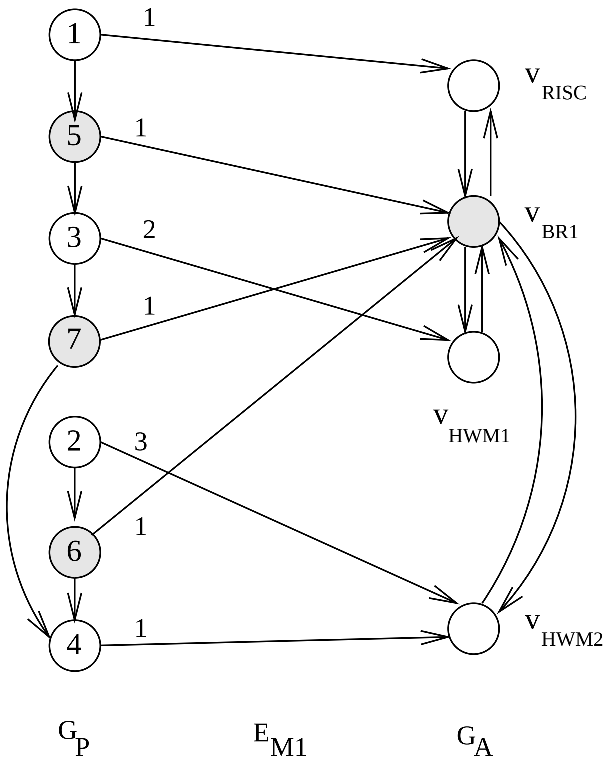


Figure 8.4: *An allocation and binding of the specification graph of Example 7.4.*

A scheduler assigns a starting time $\tau(v)$ to each node $v \in V_P$ of the problem graph G_P such that the overall execution time (latency) is minimized. Additionally, all precedence constraints and resource constraints must be fulfilled. The scheduler introduced here is based on a list scheduler, see, e.g., [DeMicheli, 1994]. List scheduling is a constructive scheduling heuristics that successively plans those tasks that have no unplanned predecessors and thereby considers resource constraints. List scheduling is able to schedule a non-cyclic task graph into non-iterative schedules. However, in dataflow-dominant applications iterative schedules are of interest. This is due to the fact that usually the operations specified by the problem graph are repeatedly executed. Iterative schedules are characterized by the fact that the schedule repeats after a certain time (called period P).

Example 8.4 *Consider the specification graph of Example 7.4 with the allocation and binding displayed in Fig. 8.4. An iterative schedule is given in Fig. 8.5 by means of a Ganttchart. A Ganttchart shows the execution times of the operations and the assignment of operations to resources. The period of the schedule is $P = 4$ as it repeats every four time units. However, the latency is $L = 8$: Calculation of one data set (drawn as unfilled rectangles) starts at time 0 with operation v_1 and v_2 and finishes with operation v_4 at time step 8. Note, that the operations v_4 starting at time step 3 and v_7 starting at time step 0 executed in the first period belong to the previous data set.*

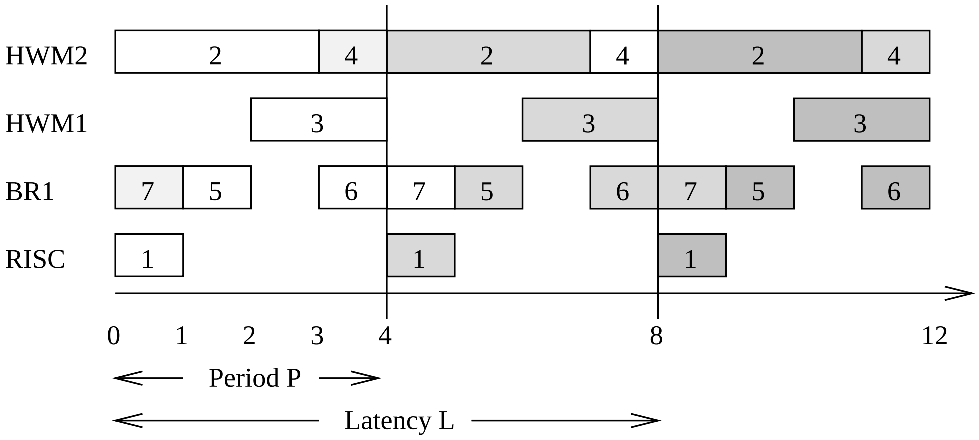


Figure 8.5: *An iterative schedule for the allocation and binding of Fig. 8.4*

The basic idea of the iterative scheduling heuristic is the removing of edges (dependencies) from the problem graph, such that the non-iterative list scheduler can be used.

The detailed algorithm is given by Algorithm 18. First, a non-iterative schedule is obtained using a list scheduler. In this initial try all operations are in the same iteration interval $\varphi(v) = 1, \forall v \in V_P$.

Then all operations that end later than the minimum period P_{min} are moved into the next iteration interval. "Moving" an operation v corresponds to the deletion of all precedence constraints on that operation, i.e., to remove all edges $e \in \{e = (u,v) \in G_E | \tau(v) + delay(v,\beta) > P_{min} \wedge \tau(u) + delay(u,\beta) \leq P_{min}\}$. The new problem graph (with some edges removed) is scheduled again using a list scheduler. The latency of this schedule is then equivalent to the period P of the iterative schedule. This procedure repeats until the current period is greater than the previous one. A parameter of the algorithm limits the maximum number of iteration intervals φ_{max} to be used. P_{min} is obtained as the maximum occupation of any resource. As allocation and binding is already fixed, this value computes to

$$P_{min} = \max_{\tilde{v} \in \alpha} \sum_{v \in V_P \wedge (v,\tilde{v}) \in \beta} delay(v,\beta) \tag{8.1}$$

Example 8.5 *Consider again the allocation and binding of Fig. 8.4. The minimum period calculates to $P_{min} = max\ \{1,\ 1+1+1,\ 2,\ 3+1\} = 4$. A non-iterative schedule is given in Fig. 8.6. According to the heuristic of Algorithm 18 the operations v_4 and v_7 are moved into the next iteration, i.e., the edges (v_3, v_7) and (v_6, v_4) are removed from the problem graph G_P in Fig. 8.4. Note, that the edge (v_7, v_4) is not removed, as both operations are in the same iteration interval. If the modified problem graph is scheduled again, the schedule of Fig. 8.5 is obtained. As the period of this schedule is optimal, the calculation is finished.*

Algorithm 18: (Iterative List Scheduling)

Input: Specification graph G_S,
current allocation α, current binding β,
delay function $delay$, maximum number of iterations φmax
Output: The iterative schedule $\vec{\tau}$,
the iteration indices $\vec{\varphi}$.

iterative_list$(G_S, delay, \alpha, \beta)$:
  $\vec{\varphi}' \leftarrow \vec{1}$
  $\vec{\tau}' \leftarrow$ list_schedule$(G_S, delay, \alpha, \beta)$
  $P_{min} \leftarrow \max_{\tilde{v}\in\alpha} \sum_{v\in V_P \wedge (v,\tilde{v})\in\beta} delay(v,\beta)$
  do
    $\vec{\varphi} \leftarrow \vec{\varphi}'$
    $\vec{\tau} \leftarrow \vec{\tau}'$
    forall $v \in G_P$ **do**
      if $\tau(v) + delay(v,\beta) > P_{min}$ **and** $\varphi(v) < \varphi max$
        $\varphi'(v) \leftarrow \varphi(v) + 1$
      endif
    od
    forall $e \in E_P$ **do**
      if $\varphi'(source(e)) \neq \varphi'(target(e))$
        hide_edge(G_S, e)
      endif
    od
    $\vec{\tau}' \leftarrow$ list_schedule$(G_S, delay, \alpha, \beta)$
    restore_edges(G_S)
  while $P(\vec{\tau}') < P(\vec{\tau})$ **and** $P(\vec{\tau}') > P_{min}$
  if $P(\vec{\tau}') = P_{min}$
    return $\vec{\tau}', \vec{\varphi}'$
  else
    return $\vec{\tau}, \vec{\varphi}$
  endif

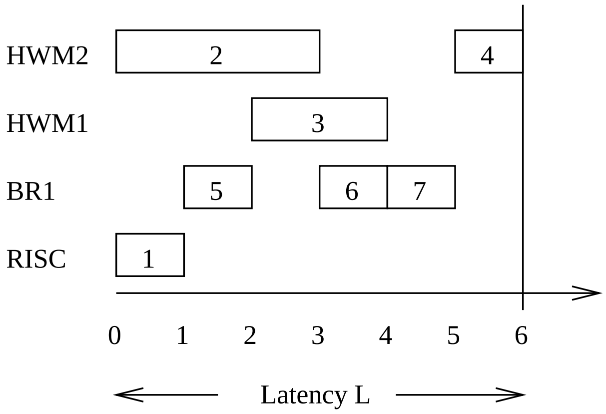


Figure 8.6: *Non-iterative schedule of the allocation and binding of Fig. 8.4 obtained with list scheduling.*

8.3 Fitness Function and Constraint-Handling

In the preceding paragraph the coding of an individual was presented. The quality of an implementation and the optimization goal for the Evolutionary Algorithm will be discussed by describing the calculation of the fitness function.

According to Definition 7.11 the system-synthesis task was described as an optimization problem. As the particular optimization goal depends on the specific design problem, the objective function h is designed individually for each problem. However, the restriction of a feasible allocation and binding still needs to be considered. The repairing heuristics introduced in the previous section already capture a large part of infeasible allocation and bindings. The remaining infeasible implementations are handled by penalty terms. Hence, the fitness function f to be minimized can be given as

$$f(J) = \begin{cases} x_a(J)p_a + x_b(J)p_b & : \quad x_a(J) = 1 \vee x_b(J) = 1 \\ f'(J) & : \quad else \end{cases} \tag{8.2}$$

The p values are the penalty terms, i.e., p_a is the penalty term for an infeasible allocation and p_b for a infeasible binding. The boolean variables x denote whether the corresponding constraint is violated or not, e.g., $x_a(J) = 0$ if the allocation is feasible and $x_a(J) = 1$ if the allocation is infeasible. The values for the penalty terms p_a and p_b should be chosen such that any infeasible allocation or binding has a larger fitness value than any feasible implementation. The modified fitness function $f'(J)$ has to reflect the additional constraints g_i. For this purpose numerous methods for constraint-handling can be applied (see Chapter 3.3). Here three different methods will be considered and empirically recommended:

- Constraint-handling using penalty terms:
  Penalty terms p_i are added to the objective function h if the corresponding constraint is violated.

$$f'(J) = h(\alpha(J), \beta(J), \tau(J)) + \sum_{i=1}^{q} x_i(J)p_i \tag{8.3}$$

- Constraint-handling by individual objective switching (IOS):
  The optimization goal is changed sequentially until all constraints are considered.

$$f'(J) = \begin{cases} q - \frac{2}{\pi}\arctan g_1(\alpha,\beta,\tau) & : \quad g_1 < 0 \\ q - 1 - \frac{2}{\pi}\arctan g_2(\alpha,\beta,\tau) & : \quad g_1 \geq 0 \wedge g_2 < 0 \\ q - 2 - \frac{2}{\pi}\arctan g_3(\alpha,\beta,\tau) & : \quad g_1, g_2 \geq 0 \\ & : \quad \wedge g_3 < 0 \\ \ldots & : \quad \ldots \\ 1 - \frac{2}{\pi}\arctan g_q(\alpha,\beta,\tau) & : \quad g_1, \ldots, g_{q-1} \geq 0 \\ & : \quad \wedge g_q < 0 \\ \frac{2}{\pi}\arctan h(\alpha,\beta,\tau) & : \quad g_1, \ldots, g_q \geq 0 \end{cases} \tag{8.4}$$

  This fitness function sequentially takes more and more constraints into account. At each stage, the objective fulfills a particular constraint. Note that the arctan function is used for normalization purposes only. Other normalization functions are possible. For details of this constraint-handling method refer to Chapter 3.3.

- Pareto-optimization:
  In [Goldberg, 1989] a Pareto ranking scheme is proposed for multi-modal optimization. Thereby the fitness of an individual J is determined by the number of individuals of the population that are better in at least one criterion than the individual J (Definition 6.1), i.e.,

$$f'(J) = \sum_{i=1,\ldots N \wedge J_i \neq J} \begin{cases} 1 & : \quad J_i \prec J \\ 0 & : \quad else \end{cases} \tag{8.5}$$

  All Pareto points in the population have an (optimal) fitness of zero. Note that the fitness of an individual depends on the population.

For each method an example is given.

Example 8.6 *As an example of the use of the penalty approach consider the adaption of the trade-off between speed and cost of an implementation. A useful definition of the function h is*

$$h(\alpha, \beta, \tau) = c_c h_c + c_t h_t \tag{8.6}$$

where h can be split into two parts: First, there is a fitness contribution due to the period of the given implementation (h_t) and second, the fitness due to the costs of the implementation (h_c). h_t and h_c are normalized by the user-specified constraints for period (P_{max}) and cost (c_{max}), i.e., $h_t = \frac{P(\tau)}{P_{max}}$, $h_c = \frac{c(\alpha,\beta)}{c_{max}}$. The sum is normalized such that $c_c h_c + c_t h_t \leq 1$ for all valid implementations by choosing the factors c_t and c_c such that $c_t + c_c = 1$. The value h_t is easily obtained as the period of a resource- constrained scheduling problem for a given allocation α and binding β. The value c is given by the amount of allocated hardware (see Section 7.5.1). The limitations on cost and time are furthermore transformed into two constrains $g_1 = P_{max} - P(\tau)$ and $g_2 = c_{max} - c(\alpha, \beta)$. It is convenient to assure that all implementations violating at least one constraint have a worse fitness (higher fitness value) than any valid implementation. As the maximal value of the function h is 1 for all valid implementations, all penalty terms *should be greater than 1. This construction of the fitness function ensures that all legal implementations have a fitness value less or equal to 1. By adjusting the coefficients c_c and c_t the optimization goal can be adjusted towards fast implementations or cheap implementations.*

Example 8.7 *As example using the IOS approach consider the task of latency minimization under resource constraints as expressed in Example 7.7. A appropriate objective function is*

$$f'(J) = \begin{cases} 1 + \frac{2}{\pi}\arctan\left\{\sum_{\tilde{v}\in\alpha(J)} c_b(\tilde{v})\right\} & : \sum_{\tilde{v}\in\alpha(J)} c_b(\tilde{v}) > c_{max} \\ \frac{2}{\pi}\arctan P(\tau(J)) & : \textit{else} \end{cases} \tag{8.7}$$

Example 8.8 *As an example for Pareto-optimization consider the case of two-dimensional optimization with the criteria cost $c(\alpha, \beta)$ and period $P(\tau)$. One obtains as fitness function:*

$$f'(J) = \sum_{i=1,\ldots,N} \begin{cases} 1 & : & (c(J) > c(J_i) \wedge P(J) \geq P(J_i)) \\ 1 & : & (c(J) \geq c(J_i) \wedge P(J) > P(J_i)) \\ 0 & : & \textit{else} \end{cases} \tag{8.8}$$

Here, $c(J)$ is used as an abbreviation of $c(\alpha(J), \beta(J))$ and the dependence of α and β on the individual J is explicitly denoted by $\alpha(J)$, $\beta(J)$. Similar $P(J)$ stands for $P(\tau(J))$.

8.4 Parameters of the Evolutionary Algorithm

The specific encoding of the individuals makes special crossover and mutation schemes necessary. In particular, for the allocations α_i uniform crossover was used [Syswerda, 1989], whereas for the several lists (repair allocation priority lists $\mathtt{L}_{Ri}$, binding order lists $\mathtt{L}_{Oi}$ and the binding priority lists $\mathtt{L}_{Bi}(v)$) order based crossover (also named position-based crossover) is applied (see, e.g. [Syswerda and Palmucci, 1991; Fox and McMahon, 1991]). Order based crossover ensures that only permutations of the elements in the lists are created. Due to the construction of the individuals no special repairing methods are necessary.

The selection operator is tournament selection with tournament size two. However, in the case of design space exploration a multi-modal selection scheme is used to keep the diversity of the population high and avoid the convergence towards a single Pareto point. The selection method used in this case is restricted tournament selection (RTS) [Harik, 1995]. RTS is a special binary tournament selection for steady state Evolutionary Algorithms: two individuals are drawn at random from the population, participate crossover and mutation and hold a binary tournament with the most similar of w randomly picked individuals of the population. w is called window size and Harik's experiments suggest a window size of four times the number niches (Pareto points) to be found.

Chapter 9

Case Study: Video Codec

In this chapter the methodology as described in Chapter 7 and 8 is used to derive implementations of a video codec algorithm. The synthesis problem is restricted to single level of hierarchy, i.e. only the mapping of the problem graph G_P to the architecture graph G_A is examined.

Fig. 9.1 shows a a block diagram of the behavior of a hybrid coder used in digital signal applications. The coder compresses video images using the H.261 standard. The basic principle is a combination of intraframe and interframe coding schemes. The algorithm operates on so called *macro blocks* containing four 8×8 pixels of luminance information and two 8×8 pixel blocks of chrominance information (color difference signals Y-R (red) and Y-B (blue)). The macro blocks are coded with variable length and a typical block size of 384 bytes.

Every prediction error frame `b` (8×8 block) is transformed into DCT coefficients (DCT operation) and quantized (Q operation) (see Fig. 9.1). Before being transmitted, the data is further reduced using a variable word-length entropy coding (RLC operation). The uncoded frame data `c` is also recovered (using inverse quantizer IQ, the inverse transformation IDCT, and adding the predicted frame `k`) and stored in the picture memory (called frame memory). The interframe coding is based on a motion estimation by comparing every macro block of the current frame `a` to all neighboring macro blocks of the previous frame `f` read from the frame memory. The result is a motion estimation vector `g` that is transmitted to the receiver and used to reconstruct the frame (motion compensation). The reconstructed frame `k` is subtracted from the input frame, such that only the prediction error is transmitted to the receiver. Additionally, a loop filter can be switched on and off to improve the picture quality by reducing high-frequency noise.

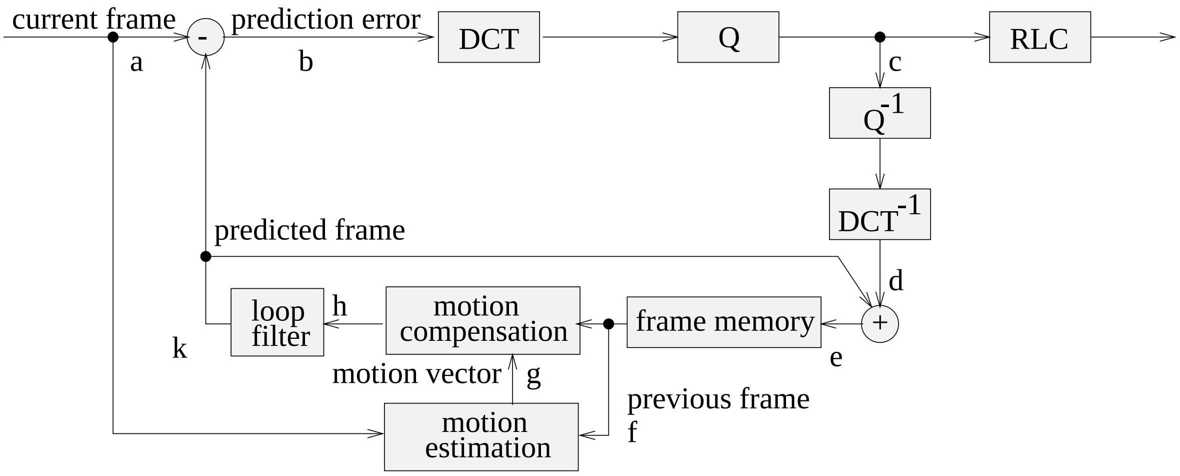


Figure 9.1: *Behavioral specification of a video codec for video compression.*

9.1 Problem Specification

The behavioral specification is refined to the problem graph of the coder (Fig. 9.2). Motion estimation is done by the block matching operation (BM), the block subtraction is named DIFF (difference), the block addition REC (recover), and the memory operations RF (read frame) and SF (store frame). Additionally, the quantization is split up into threshold calculation (TH) and quantization (Q). The operations are performed on macro blocks and each communication node (circle) represents a transmission of one macro block. However, the block matching operation needs in average three macro blocks of the previous frame for its operation. This is symbolized by a “3” in the corresponding communication node (node #17).

The coder is usually combined with a decoder to obtain a codec implementation. Fig. 9.3 shows the problem graph of the decoding algorithm. All decoding operations end with the suffix “D”. Basically, the decoding consist of the lower part of the loop of the coding algorithm. However, the motion compensation vector has to be extracted from the input data after the run-length decoding operation (RLD). As the amount of transmitted data is small compared to the size of a macro block the transmission is assumed to take zero time (symbolized by a ”0” in communication node #48).

Both graphs are mapped onto a target architecture shown in Fig. 9.4. The architecture consists of three shared bus with different data rates, two memory modules (a single and a dual ported memory) two programmable RISC processors, a signal processor (DSP), several predefined hardware modules (namely a block matching module (BMM), a module for performing DCT/IDCT operations (DCTM), an subtract/adder module (SAM) and a Huffman coder (HC)), and I/O devices (INM and OUTM).

This architecture set allows a wide range of possible implementations. The three busses are modeled to be alternatively used, as each module is intended to have only a single bus connector. This is achieved by introducing a *selection node* (node #30, not shown in the figures) as specified in Chapter 7.4.1. The only exception is the BM-module that may use two ports of which one is dedicated to the (potentially allocated) dual ported frame memory (DPFM). This special point-to-point bus allows a fast architecture to be synthesized. The two frame memories are modeled for alternative use only using a selection node (node #49).

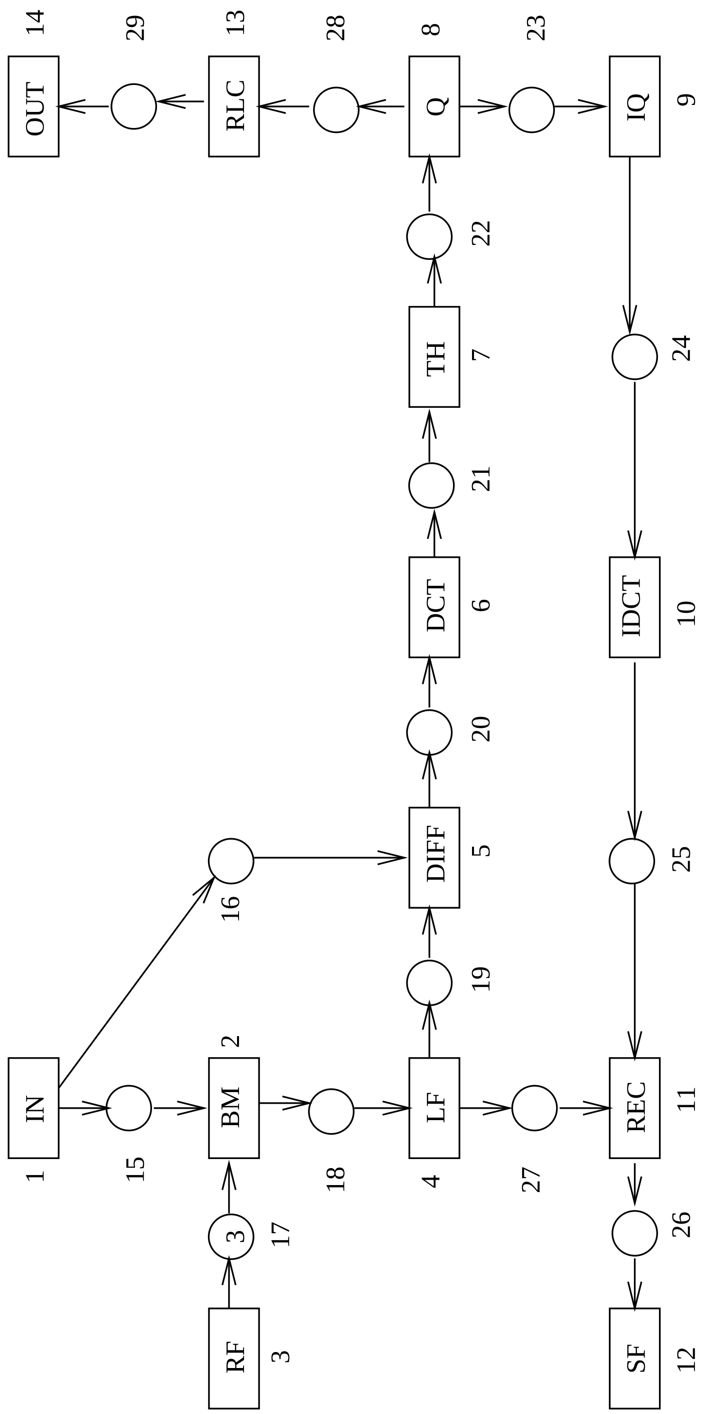


Figure 9.2: *Problem graph of the video coder in Fig. 9.1. Functional nodes are symbolized by squares, communication nodes by circles.*

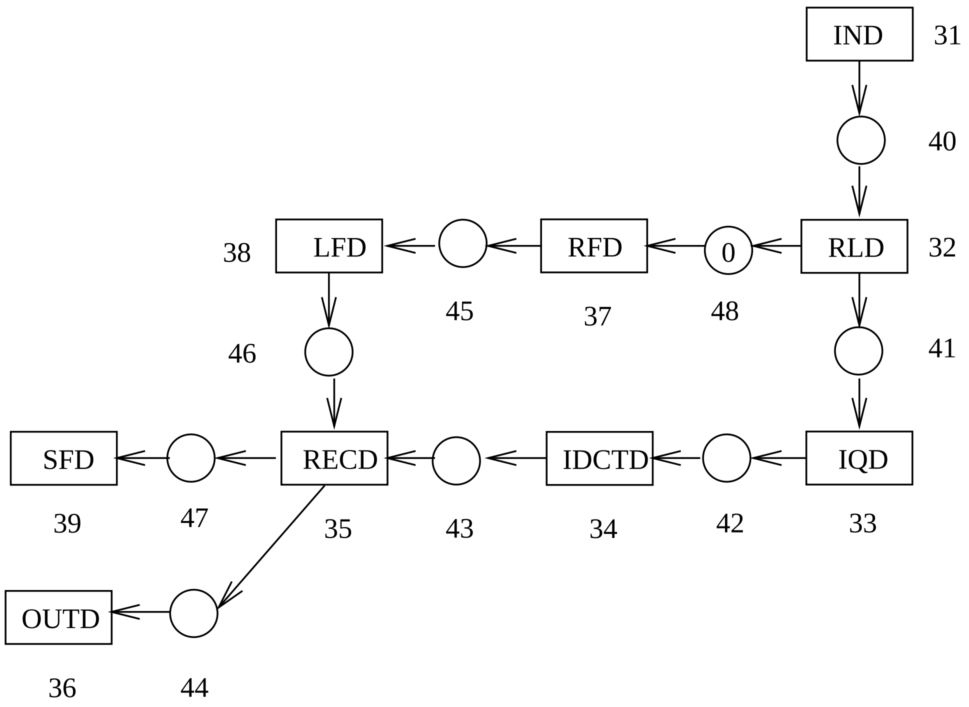


Figure 9.3: *Problem graph of the video decoder.*

The cost of an implementation is simply reflected by the sum of the costs of the allocated hardware (see Example 7.7). The cost of each module is given in brackets in Fig. 9.4.

It is assumed that the delay value of a node depends only on the binding of this particular node. Hence, the delay values can be associated to the mapping edges. The risc processors (RISC1 and RISC2) as well as the signal processor (DSP) are capable of performing any functional operation. However, the DSP executes the operations faster and is more expensive in acquisition. The other hardware modules are dedicated to special groups of operations. For example, the DCTM can only perform the three operations DCT, IDCT, and IDCTD. Possible mappings and delays of all modules are shown in Table 9.1.

Node #	Operation	Resource/ Delay	Resource/ Delay	Resource/ Delay
1	IN	INM/0		
31	IND	INM/0		
14	OUT	OUTM/0		
36	OUTD	OUTM/0		
2	BM	BMM/22	DSP/60	RISC/88
3	RF	FM/0	DPFM/0	
37	RFD	FM/0	DPFM/0	
12	SF	FM/0	DPFM/0	
39	SFD	FM/0	DPFM/0	
4	LF	HC/2	DSP/3	RISC/9
38	LFD	HC/2	DSP/3	RISC/9
5	DIFF	SAM/1	DSP/2	RISC/2
6	DCT	DCTM/2	DSP/4	RISC/8
10	IDCT	DCTM/2	DSP/4	RISC/8
34	IDCTD	DCTM/2	DSP/4	RISC/8
7	TH	HC/2	DSP/8	RISC/8
8	Q	HC/1	DSP/2	RISC/2
9	IQ	HC/1	DSP/2	RISC/2
33	IQD	HC/1	DSP/2	RISC/2
11	REC	SAM/1	DSP/2	RISC/2
35	RECD	SAM/1	DSP/2	RISC/2
13	RLC	HC/2	DSP/8	RISC/8
32	RLD	HC/2	DSP/8	RISC/8

Table 9.1: *The mapping of the functional nodes to architectural nodes.*

Comm. Node #	Res./ Delay	Res./ Delay	Res./ Delay	Res./ Delay	Res./ Delay	Res./ Delay
15	SBF/1	SBM/2	SBS/3			
16	SBF/1	SBM/2	SBS/3			
17	SBF/3	SBM/6	SBS/9	PTP/1		
18	SBF/1	SBM/2	SBS/3	RISC/0	DSP/0	
19	SBF/1	SBM/2	SBS/3	RISC/0	DSP/0	
20	SBF/1	SBM/2	SBS/3	RISC/0	DSP/0	
21	SBF/1	SBM/2	SBS/3	RISC/0	DSP/0	
22	SBF/1	SBM/2	SBS/3	RISC/0	DSP/0	HC/0
23	SBF/1	SBM/2	SBS/3	RISC/0	DSP/0	HC/0
24	SBF/1	SBM/2	SBS/3	RISC/0	DSP/0	
25	SBF/1	SBM/2	SBS/3	RISC/0	DSP/0	
26	SBF/1	SBM/2	SBS/3			
27	SBF/1	SBM/2	SBS/3	RISC/0	DSP/0	
28	SBF/1	SBM/2	SBS/3	RISC/0	DSP/0	HC/0
29	SBF/1	SBM/2	SBS/3			
30	SBF/0	SBM/0	SBS/0			
40	SBF/1	SBM/2	SBS/3			
41	SBF/1	SBM/2	SBS/3	RISC/0	DSP/0	HC/0
42	SBF/1	SBM/2	SBS/3	RISC/0	DSP/0	
43	SBF/1	SBM/2	SBS/3	RISC/0	DSP/0	
44	SBF/1	SBM/2	SBS/3			
45	SBF/1	SBM/2	SBS/3			
46	SBF/1	SBM/2	SBS/3	RISC/0	DSP/0	
47	SBF/1	SBM/2	SBS/3			

Table 9.2: *The mapping of the communication nodes to architectural nodes.*

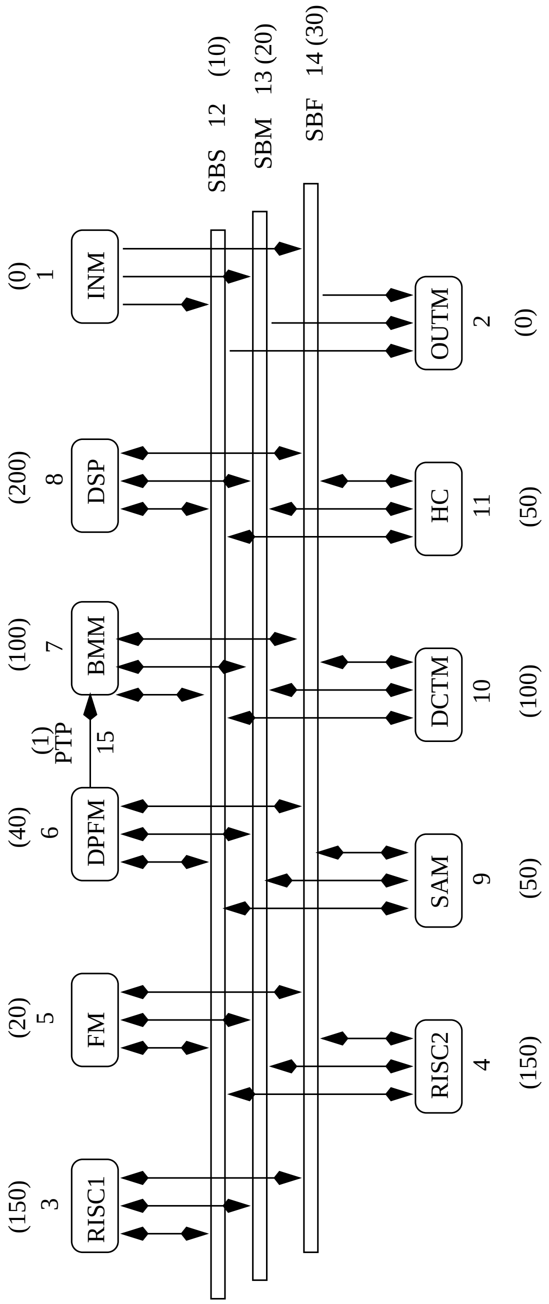


Figure 9.4: *Architecture template of architectures to implement the problem graph of the video codec. The values in brackets give the basic cost c_b of each resource.*

The mappings of communication nodes to resources are specified in Table 9.2. Note that several communication operations may be internal, i.e., handled by functional resources. Only those mappings are specified that are a priori known to be possible. By this the search space is reduced in advance using domain knowledge. Nevertheless, if such knowledge is not present a more extensive specification can be given. For example, all communication nodes could be mapped to all functional resources. However, this enlarges the search space and makes the optimization process more difficult.

This specification models are realistic synthesis task. The number of possible bindings are $1.9 \cdot 10^{27}$. This demonstrates the intractability of enumerative or exact methods to explore the search space.

9.2 Synthesizing Architectures

This section describes the implementations found using the novel approach. Three different groups of experiments are performed. First, optimization towards a single implementation is carried out. Next, a design-space exploration is considered by exploring the Pareto set of implementations, followed by an experiment optimizing the architectures for maximum load, i.e., the average usage of the resources is maximized.

9.2.1 Optimizing for a Single Architecture

Frequently, time constraints or cost constraints are given in advance, and only implementations fulfilling these constraints are of interest. The objective of these problems are usually stated as period minimization under cost constraints or cost minimization under period constraints, respectively.

For this optimization task the IOS constraint-handling approach is used. The fitness function for period minimization under cost constraints is:

$$f(J) = \begin{cases} x_a(J)p_a + x_b(J)p_b & : \quad x_a(J) = 1 \vee x_b(J) = 1 \\ f'(J) & : \quad else \end{cases}$$

with

$$f'(J) = \begin{cases} 1 + \frac{2}{\pi} \arctan \left\{ \sum_{\tilde{v} \in \alpha(J)} c_b(\tilde{v}) \right\} & : \quad \sum_{\tilde{v} \in \alpha(J)} c_b(\tilde{v}) > c_{max} \\ \frac{2}{\pi} \arctan P(\tau(J)) & : \quad else \end{cases}$$

For different values of maximum cost an optimization is performed. In all experiments the population size was $N = 30$, the crossover probability $p_c = 0.5$, the mutation probability $p_m = 0.2$, and the maximum number of generation $k_{max} = 100$. The best results of ten runs each are summarized in Table 9.3(right).

Similar, several EAs have been executed for cost minimization under period constraints (Table 9.3(left)) using the fitness function:

$$f'(J) = \begin{cases} 1 + \frac{2}{\pi} \arctan P(\tau(J)) & : \quad P(\tau(J)) > P_{max} \\ \frac{2}{\pi} \arctan \left\{ \sum_{\tilde{v} \in \alpha(J)} c_b(\tilde{v}) \right\} & : \quad else \end{cases}$$

The entries shown in bold in Table 9.3 mark the Pareto points of all implementations in this table.

9.2.2 Design Space Exploration

The experiments in this section aim to perform a design space exploration. For this purpose, Pareto optimization is used with the fitness function of Example 8.8, i.e.

$$f(J) = \begin{cases} x_a(J)p_a + x_b(J)p_b & : \quad x_a(J) = 1 \vee x_b(J) = 1 \\ f'(J) & : \quad else \end{cases}$$

P_{max}	P	c	c_{max}	c	P
22	**22**	**350**	345	**340**	**42**
30	24	350	340	340	42
42	23	350	330	**330**	**54**
50	48	340	325	290	78
75	66	330	300	300	78
80	80	280	280	**280**	**78**
100	**78**	**280**	270	250	114
110	103	280	250	230	114
114	**114**	**230**	229	200	166
130	**114**	**230**	200	190	166
165	116	230	180	**180**	**166**

Table 9.3: *Implementations obtained for cost minimization under period constraints (left) and period minimization under cost constraints (right). The entries in bold mark Pareto points.*

and f' given as

$$f'(J) = \sum_{i=1,\ldots,N} \begin{cases} 1 & : \quad (c(J) > c(J_i) \wedge P(J) \geq P(J_i)) \\ 1 & : \quad (c(J) \geq c(J_i) \wedge P(J) > P(J_i)) \\ 0 & : \quad else \end{cases} \qquad (8.8)$$

The fitness value directly gives the number of implementations that dominate an individual J and the Pareto implementations have a fitness value of zero. Note that the Pareto points determined the way above are only Pareto points according to the current population. In order to obtain many Pareto-optimal implementations the population size is increased to $N = 100$ and RTS selection is used with window size $w = 20$ (see Section 8.4). The Pareto set found in a single optimization run after 200 generations is shown in Table 9.4 and depicted in Fig. 9.5.

There is no evidence that the Pareto points obtained above are the Pareto points of the problem but for the fastest and cheapest

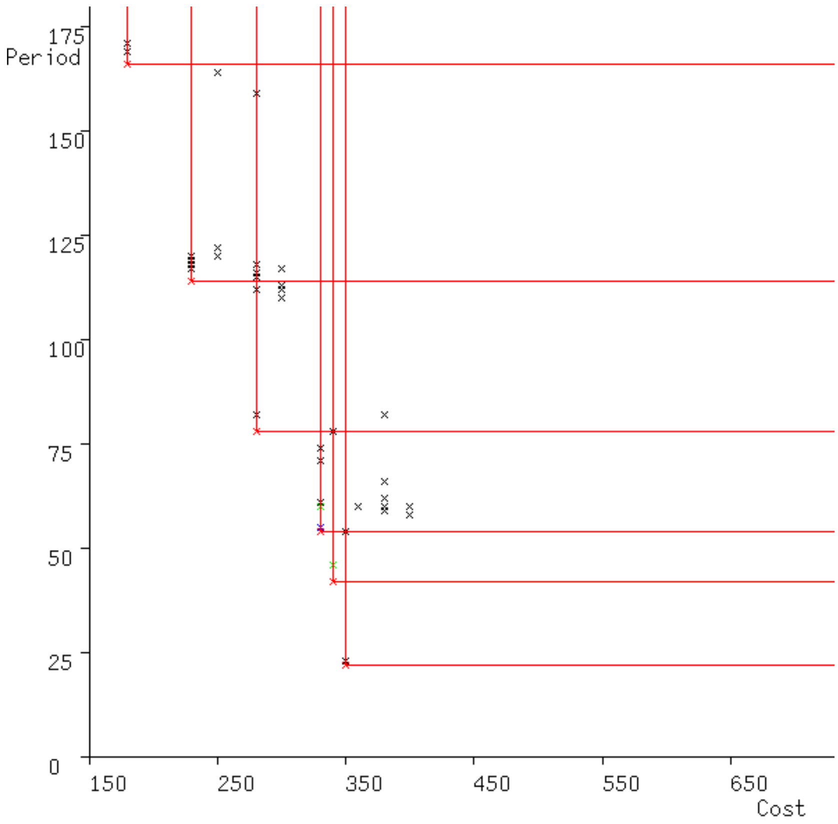


Figure 9.5: *Design space with Pareto points of Table 9.4. The crosses mark implementations represented in the final population. Pareto points are located at the origins of the horizontal and vertical lines.*

Period P	22	42	54	78	114	166
Cost c	350	340	330	280	230	180
Fig.	9.6			9.8	9.7	

Table 9.4: *Implementations obtained as Pareto points in a single run of the Evolutionary Algorithm (see also Fig. 9.5).*

implementation it can be seen that they are optimal. Furthermore, the Pareto points are identical to the Pareto points in Table 9.3.

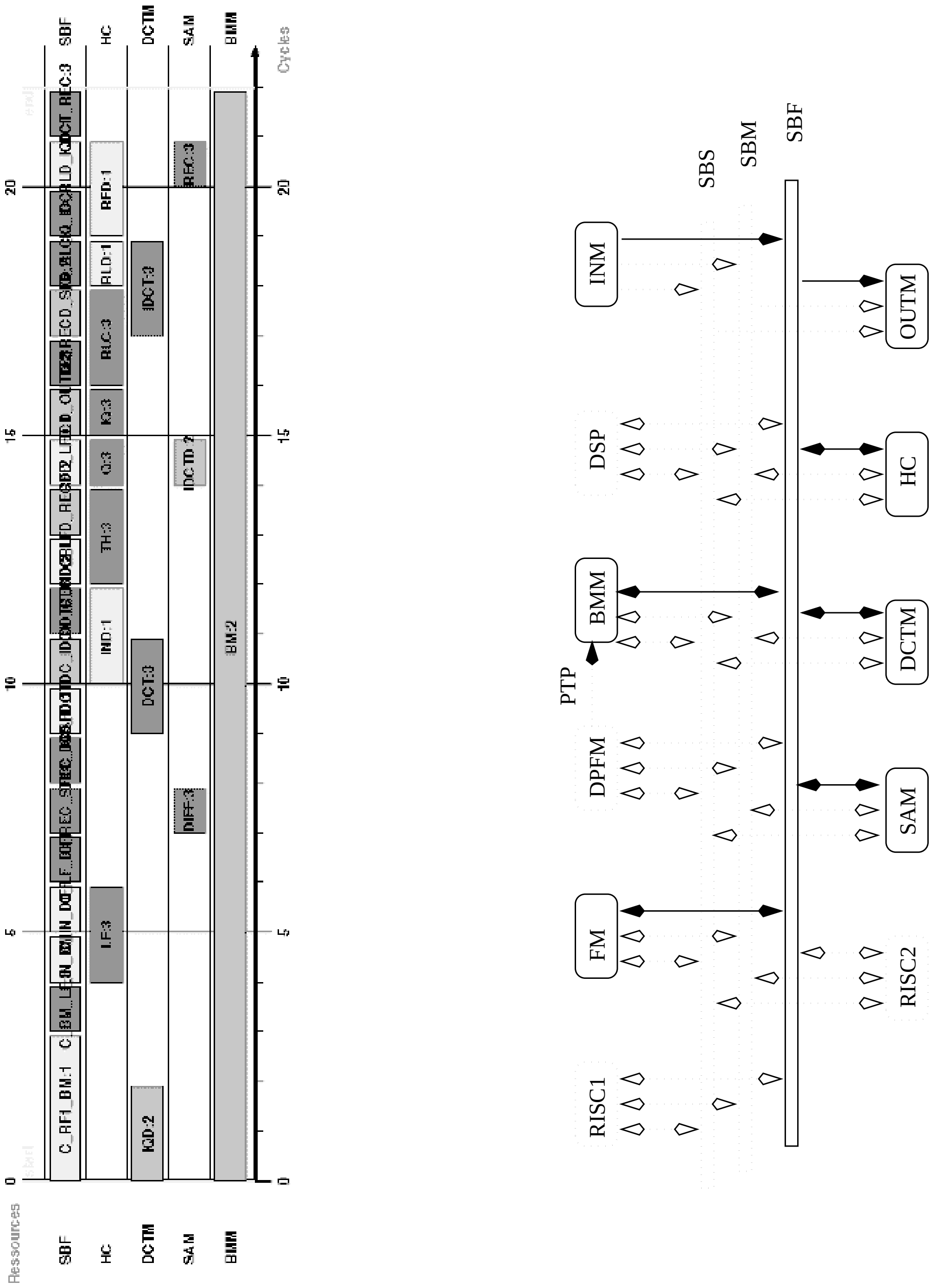


Figure 9.6: *Ganttchart and architecture of the fastest implementation (cost $c = 350$, period $P = 22$).*

Fig. 9.6 shows the Ganttchart and the corresponding architecture of the fastest implementation found. The minimal period of the implementation is obviously determined by the execution time of the block matching module (BMM). The architecture seems to indicate that all another modules are too fast or the BMM is too slow, respectively. This shows that the resources are not well balanced in their performance.

The cheapest implementation is shown in Fig 9.7. There are no surprises here as the cheapest way to implement the codec algorithm is to use the cheapest module capable of handling all functionalities, the cheapest bus and the cheapest memory module.

As a last example consider the Pareto point with cost $c = 280$ and period $P = 78$. The Ganttchart and architecture are shown in Fig. 9.8.

9.2.3 Optimizing the Efficiency of Resources

An useful measure to evaluate an implementation is the efficiency.

Definition 9.1 (Efficiency) *The* efficiency ρ *of a resource* $\tilde{v} \in G_A$ *of an implementation* (α, β, τ) *is the fraction of its usage time during a period, i.e.,*

$$\rho(\tilde{v}) = \frac{1}{P(\tau)} \sum_{v \in G_P : (v, \tilde{v}) \in \beta} delay(v, \beta) \tag{9.1}$$

The optimization for maximum average efficiency of all resources of an implementation yields an architecture where all resources have a good utilization. The objective can be stated as

$$\text{minimize } h = -\bar{\rho}(\alpha, \beta, \tau) = -\sum_{\tilde{v} \in \alpha} \rho(\tilde{v})$$

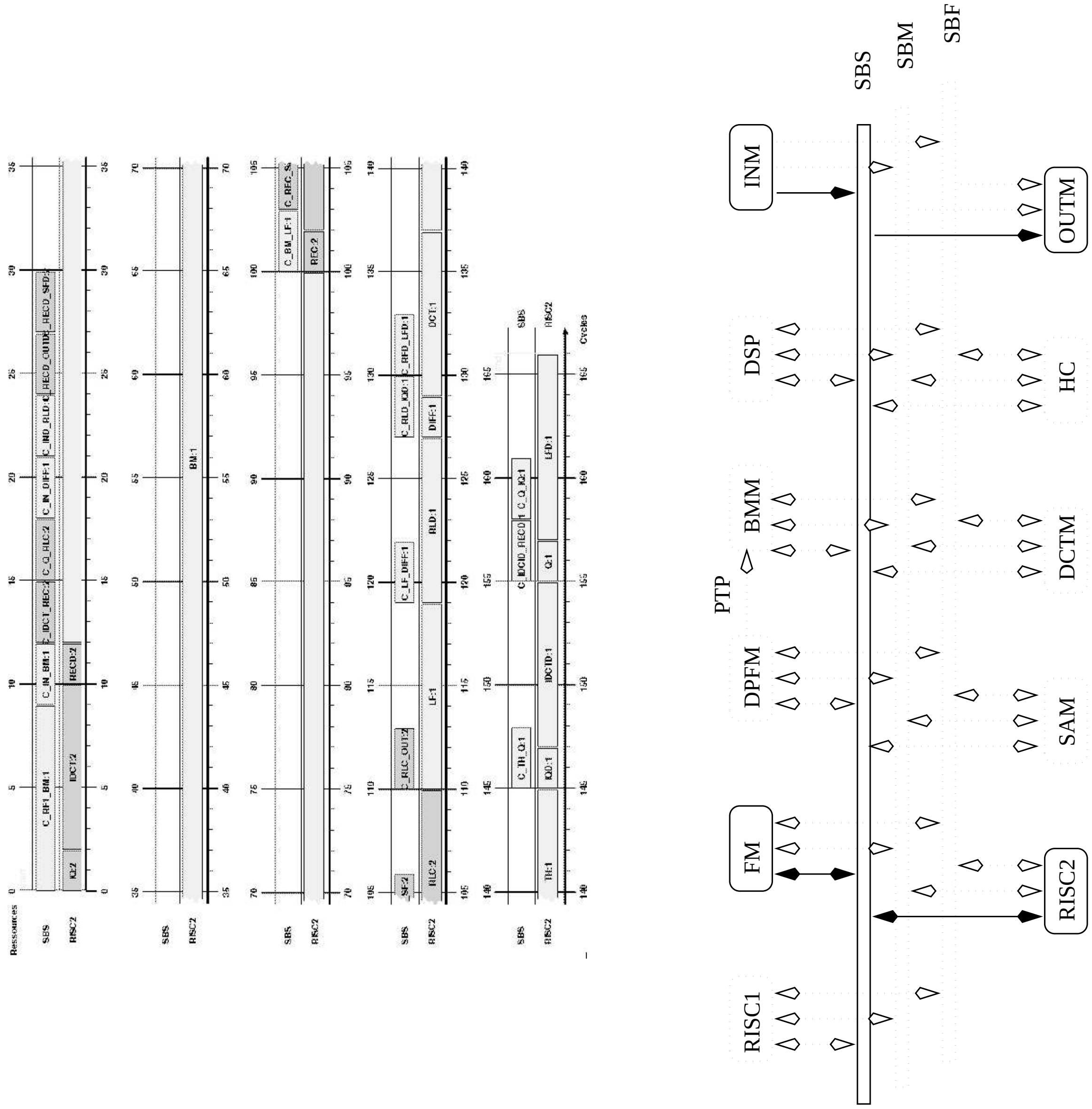


Figure 9.7: *Ganttchart and architecture of the cheapest implementation (cost $c = 180$, period $P = 166$).*

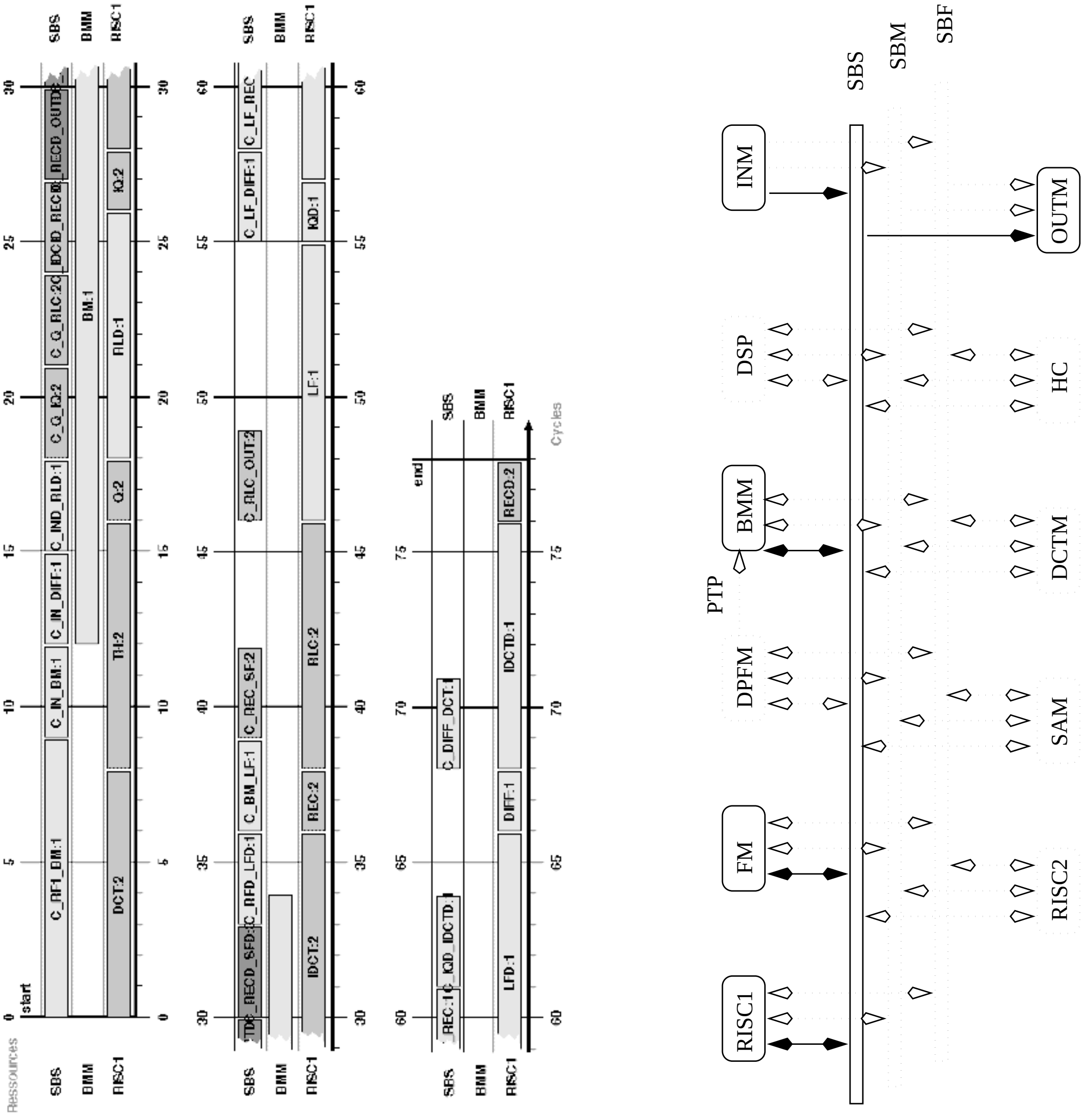


Figure 9.8: *Ganttchart and architecture of the implementation with cost* $c = 280$ *and period* $P = 78$.

It is interesting to note that the cost of the implementations are not considered at all. This way, a cheap implementation is not preferred to expensive one. However, any implementation with a high average efficiency is good in the sense that the allocated hardware is well utilized.

Table 9.5 summarizes the results of the experiments performed. The runs stopped after 6000 fitness functions evaluations and the best results are reported (population size $N = 30$, mutation rate $p_m = 0.2$, crossover rate $p_c = 0.5$). Different maximum numbers of iteration intervals have been used. One expects that higher iteration intervals allow a better efficiency. However, higher iteration indices correspond to a higher memory demand. The table gives the best solution found in ten runs. The increase in utilization going from two to three iterations is enormous: The best utilization with $\varphi_{max} = 2$ is only 85.8% whereas $\varphi_{max} = 3$ results in an almost optimal utilization (98.5%). The increase in performance using a higher number of iterations is marginal. However, the probability to obtain a good solutions increases steadily. This can be seen in Fig. 9.9. Here the average of all utilization of the ten runs is shown in dependence of the number of fitness evaluations. This demonstrates the influence of the scheduler on the optimization problem. Obviously, a schedule with a high number of iterations is more dense than one with only few iterations allowed. Consequently, "dense" implementations are found more frequently.

φ_{max}	$\bar{\rho}_{best}$	P	c	Figure
2	0.858	77	400	
3	0.985	67	380	9.10
4	0.986	70	400	
5	0.986	70	380	

Table 9.5: *Results of efficiency maximization.*

Fig. 9.10 shows the Ganttchart and architecture of the implementation with a maximum iteration interval of $\varphi_{max} = 3$ and an average resource utilization of 98.5%. Note, that the bus load is also maxi-

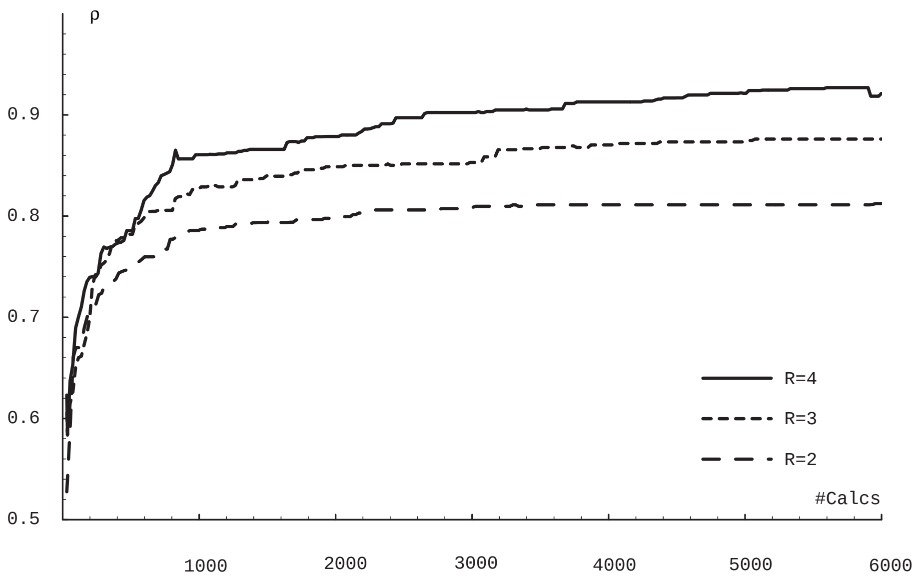
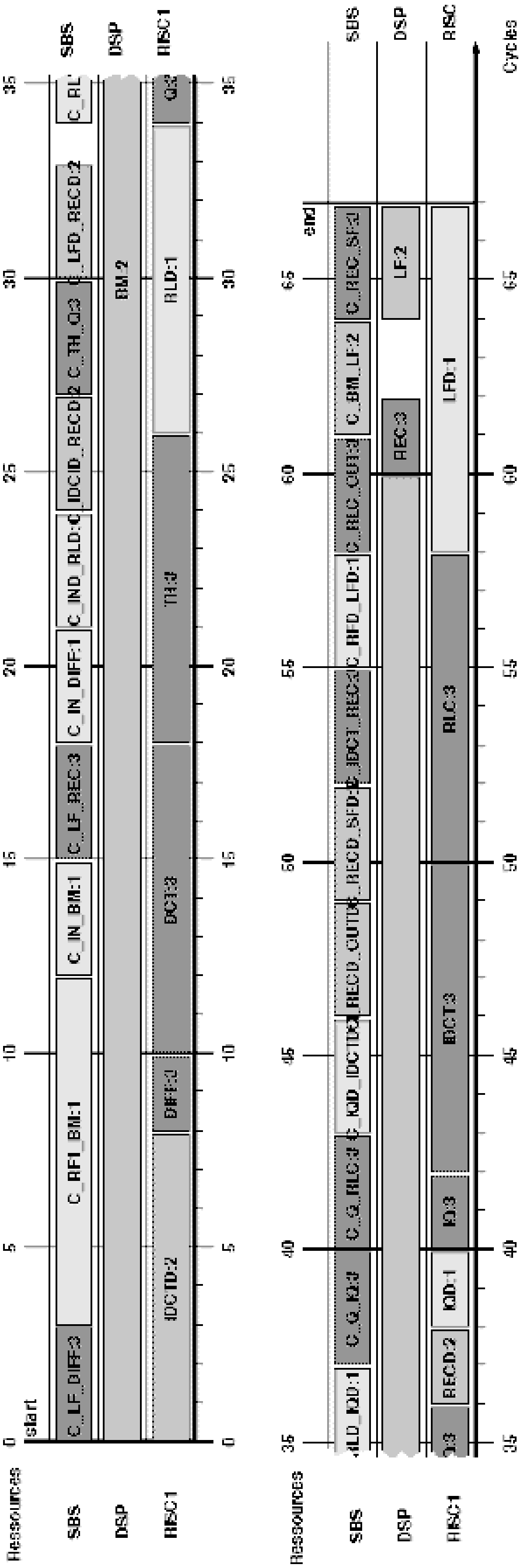


Figure 9.9: *Influence of the maximum iteration interval on the optimization*

mized as the model makes no differences between functional resources and bus resources. Hence, many "unnecessary" communications are performed as internal communications are partly handled by the external bus.

9.3 Performance of the Optimization Methodology

In this section some performance issues of the optimization methodology are discussed. As example the optimization of the average load $\bar{\rho}$ is used with maximum number of iterations $\varphi_{max} = 3$. This task seems to be more difficult than cost or period minimization problems. All results given are averages over ten runs.



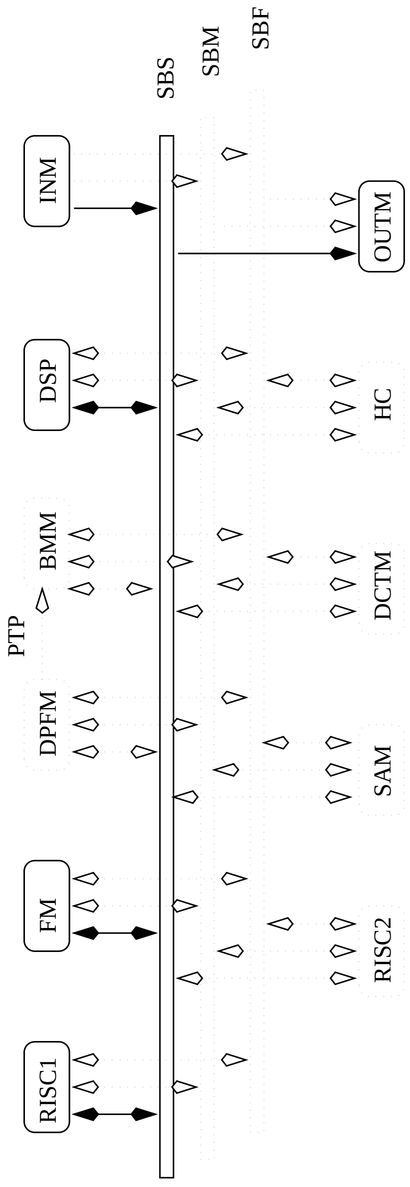


Figure 9.10: *Ganttchart and architecture of the implementation with highest average load (cost $c = 380$, period $P = 67$, efficiency $\bar{\rho} = 98.5\%$).*

9.3.1 Monte Carlo and Hill Climbing

This experiment optimizes the video-codec using the Monte Carlo heuristic (see also Algorithm 1 on page 18) and Hill Climbing heuristic (Algorithm 2 on page 19). Table 9.6 summarizes the results: The mean best average load $\bar{\rho}$ over ten runs is almost identical (0.807 for Monte Carlo and 0.804 for Hill Climbing) and the best implementation found in the ten runs is only slightly better than the average results.

	Mean $\bar{\rho}$	$\bar{\rho}_{best}$
Monte Carlo	0.807	0.873
Hill Climbing	0.804	0.867

Table 9.6: *Performance of Hill Climbing and Monte Carlo (5000 fitness evaluations).*

In the next section several parameter setting of the Evolutionary Algorithm are tested on the same problem.

9.3.2 Influence of mutation probability p_m and crossover probability p_c

The population size is $N = 30$ and the runs aborted after 5000 fitness function evaluations. Table 9.7 shows the average fitness value of ten runs for various crossover and mutation rates and Table 9.8 the best objective values. In almost all cases the performance of the EA is better than the performance of Monte Carlo or Hill Climbing. Only the numbers shown in italic indicate worse performance. This happened when no mutation was used. The best results obtained are printed in bold.

The results seem to indicate that mutation is more important than crossover for this problem. With no mutation a mean value of the average load above 0.8 is only achieved for a crossover rate of 50%.

	$p_c = 0$	$p_c = 0.25$	$p_c = 0.5$	$p_c = 0.75$
$p_m = 0$		*0.737*	0.843	*0.775*
$p_m = 0.1$	0.866	0.827	0.871	0.860
$p_m = 0.15$	0.883	0.875	**0.912**	0.838
$p_m = 0.2$	0.876	0.879	0.894	0.894
$p_m = 0.4$	0.862	0.848	0.855	0.852

Table 9.7: *Influence of mutation probability p_m and crossover probability p_c on the average fitness (5000 fitness evaluations).*

	$p_c = 0$	$p_c = 0.25$	$p_c = 0.5$	$p_c = 0.75$
$p_m = 0$		*0.802*	0.955	*0.868*
$p_m = 0.1$	0.950	0.882	0.966	0.966
$p_m = 0.15$	0.952	0.970	**0.971**	0.957
$p_m = 0.2$	0.970	**0.971**	0.955	0.957
$p_m = 0.4$	0.952	0.900	0.923	0.955

Table 9.8: *Best results obtained in ten runs for different crossover and mutation rates (5000 fitness evaluations).*

Using no crossover the performance is better, especially if the best average load values are considered (Table 9.8). For both parameters a high probability leads to a reduction of performance. A crossover rate of 75% shows in almost all case a decrease of performance compared to 50% crossover, independently of the mutation rate. Similar, a high mutation rate of 40% leads to a "noisy" optimization behaviour and a decreased performance. Using moderate mutation rates and moderate crossover rates yields a synergetic effect: The best behaviour is obtained for a mutation rate of $p_m = 0.15$ and a crossover rate of $p_c = 0.5$.

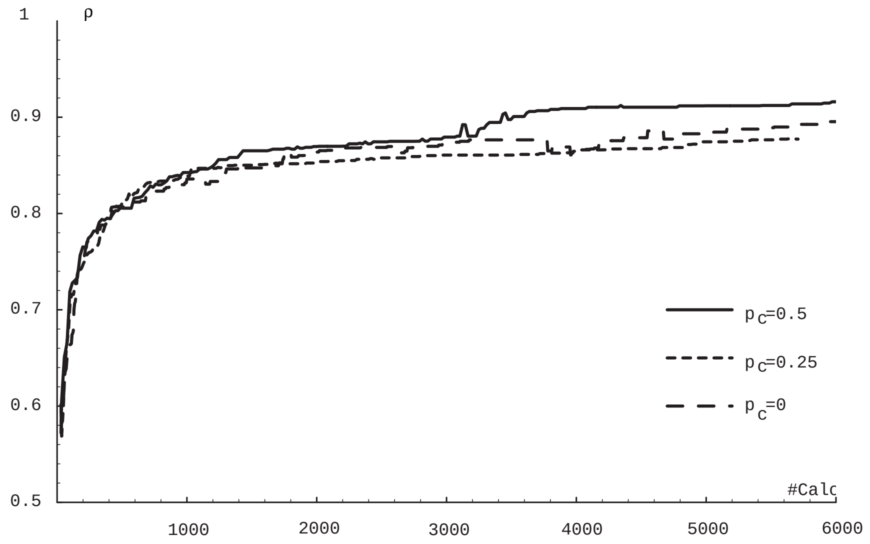


Figure 9.11: *Performance of the EA for different crossover probabilities and constant mutation probability $p_m = 0.15$.*

For a mutation rate of $p_m = 0.2$ Fig. 9.11 shows the performance for three different crossover probabilities. Likewise, Fig. 9.12 shows the performance for constant crossover probability of 50% and four different mutation rates.

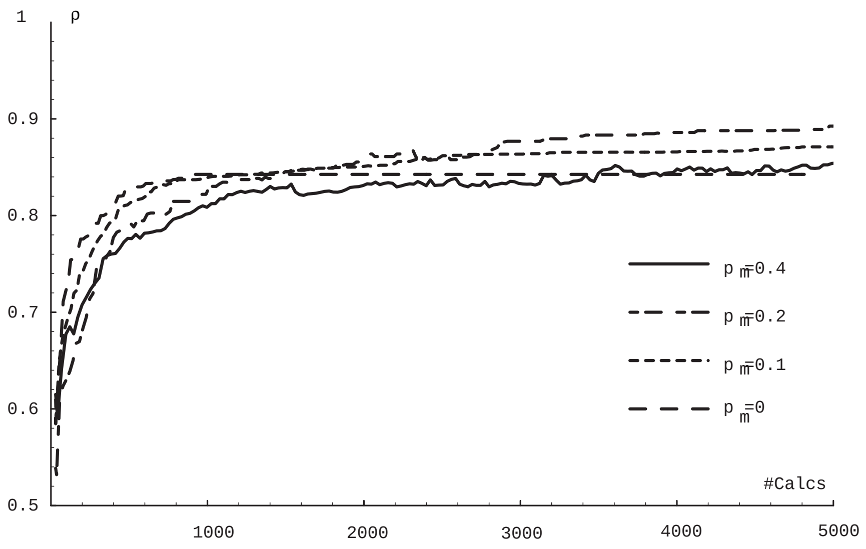


Figure 9.12: *Performance of the EA for different mutation probabilities and constant crossover probability* $p_c = 0.5$.

9.3.3 Influence of the population size N

This experiment investigated the influence of the population size of the performance. The mutation probability is $p_m = 0.2$ and the crossover probability is $p_c = 0.5$. The runs are aborted after 6000 fitness function evaluations. The results are shown in Fig 9.13. A larger population size ($N = 100$) seems not to be appropriate for this problem. Better results are achieved with smaller populations. A population size of $N = 30$ seems to be appropriate.

9.3.4 Influence of the search space size

The binding search space of the video codec as specified above calculates to $1.9 \cdot 10^{27}$ combinations. As mention above this reflects already a "reduced" search space, as the possible bindings for the communication nodes are a priori minimized using domain knowledge (see page 201). Without this knowledge the specification must be changed such

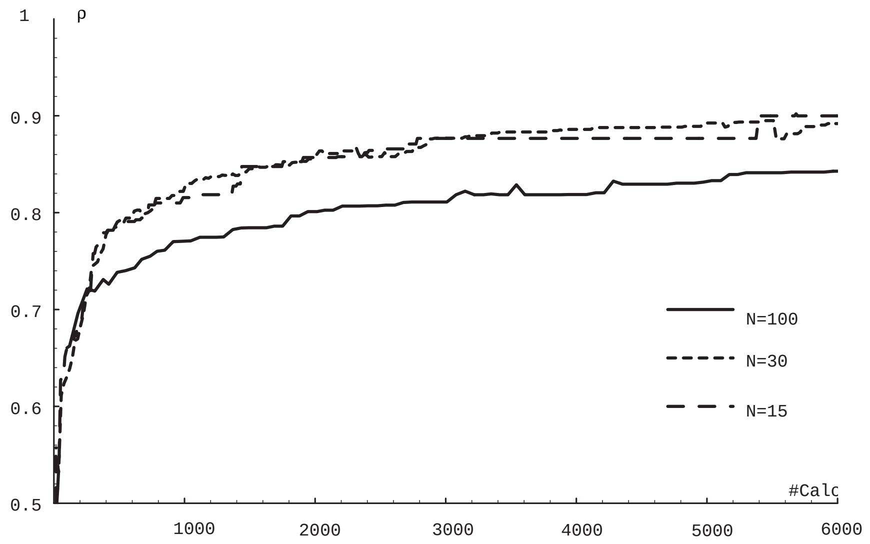


Figure 9.13: *Comparison of the influence of the population size on the quality of the optimization.*

that all communication operations can be mapped to any functional resource. This enlarges the search spaces to $4.6 \cdot 10^{29}$ binding combinations, a factor of about 200. Fig. 9.14 shows the performance of the EA for the two different problem sizes. The performance is only slightly worse for the more complex problem indicating the robustness of the EA in searching complex spaces.

9.4 Conclusions

In this chapter a real-world synthesis problem has been considered, the synthesis of an architecture for a h261 video codec circuit. The complexity of the problem is realistic and typical for system-level synthesis. The methodology is able to simultaneously select the architecture (allocation), perform the assignment of operations to resources (binding) and annotate starting times to each operation (scheduling). The implementation of the model is based on Evolutionary Algorithms.

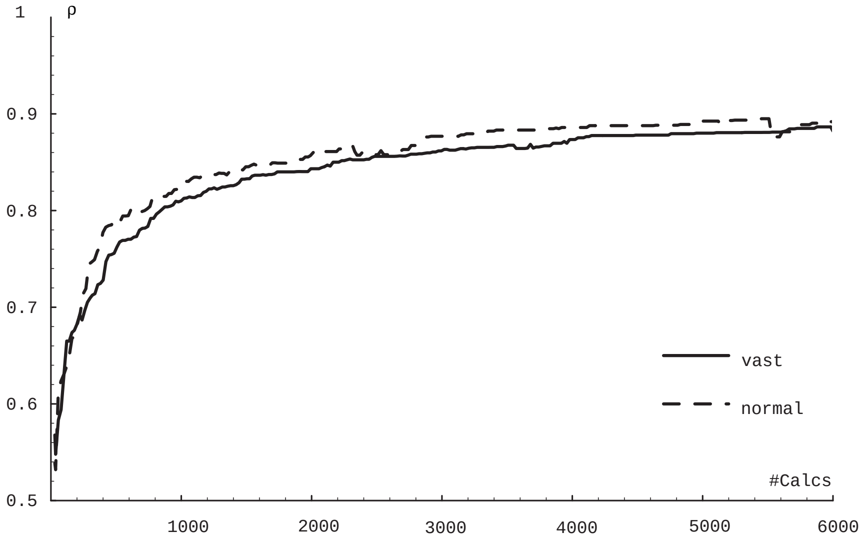


Figure 9.14: *Comparison of the performance on two different problem sizes.*

Remarkable is the fact that the EA is capable of exploring the design space and identifying the Pareto points of a synthesis problem in a single optimization run.

Chapter 10

Summary and Outlook

10.1 Fundamental Results

Part I of this thesis was devoted to an analysis of Evolutionary Algorithms with regard to solving global optimization problems as stated in Chapter 2. In Chapter 3 EAs were introduced and various constraint handling methods discussed including a new method for constraint handling called individual objective switching (IOS). In Chapters 4 and 5 a detailed mathematical analysis of Evolutionary Algorithms, in particular Genetic Programming, was made.

In Chapter 4 the most popular generational, unimodal, and non-elitist selection schemes were analyzed. The analysis was based on fitness distributions and examined only the local behaviour of the selection, i.e., the fitness distributions after selection were compared with the fitness distributions before selection. Although this idea is not new, the consequent realization of this idea led to a powerful framework that allowed the following results to be obtained:

- Fitness distributions allow an unified view of the selection schemes and enable several, up to now independently and isolated obtained aspects of selection schemes to be derived with one single methodology.
- Some interesting features of selection schemes could be proven, e.g., the concatenation of several tournament selections (Theorem 4.7) and the equivalence of binary tournament and linear ranking (Theorem 4.21).
- Important characteristics of selection schemes, like selection intensity, selection variance, and loss of diversity were derived.
- The selection intensity was used to obtain a convergence prediction of the simple Genetic Algorithm with uniform crossover optimizing the ONEMAX function.
- The comparison of the loss of diversity and the selection variance based on the selection intensity allowed for the first time "second order" properties of selection schemes to be compared. This gives a well grounded basis for deciding which selection scheme is applicable, assuming that for a particular problem the impact of these properties on the optimization process is known.

In Chapter 5 the crossover operator in Genetic Programming was investigated. A mathematical analysis quantifying the redundancy occurring in the trees produced by crossover was carried out. This enabled a prediction of the generational behaviour of GP to be made. From this analysis it was shown that introns or redundant code segments aggravate the optimization process. To reduce the negative effects a special crossover operator, the *deleting crossover*, was proposed. This operator was shown to be very effective for a single discrete problem. Though this example may be especially suited for the deleting crossover operator, it demonstrates the influence of redundancy on the evolvability of the population.

In addition, the bloating phenomenon was addressed by a comparison of several proposed methods. They were applied on two experiments, taken from a continuous and a discrete domain, to quantify their ability to reduce the tree size. The main results were:

- For discrete problems, a constant parsimony pressure is sufficient to reduce the tree size, but it reduces the probability of success. The *deleting crossover* lead to a higher success rate whereas the introduction of *explicitly defined introns* (EDIs) [Nordin *et al.*, 1995] reduced the performance for the 6-multiplexer problem.
- For discrete problems, an adaptive parsimony pressure [Zhang and Mühlenbein, 1995] can be applied resulting in almost the same probability of success compared with the unconstrained optimization problem and in a reduced tree size.
- For continuous problems a constant parsimony pressure is inadequate and the use of EDIs or deleting crossover superfluous.
- For continuous problems the adaptive parsimony pressure is able to efficiently reduce the tree size.
- The new *individual objective switching* (IOS) approach works well on both discrete and continuous problems.

Part II of this thesis addressed the system-level synthesis of complex digital systems consisting of hardware and software components. The main characteristics of the approach proposed in this thesis are:

- The approach is designed for dataflow-dominant applications that can be scheduled statically during compile-time.
- The approach handles the *selection of the architecture* (allocation), *mapping* of the algorithm onto the selected architecture in space (*binding*) and time (*scheduling*).
- A formulation of the synthesis problem as a constrained global optimization problem was given, based on a formal graph-based specification. Using this formulation costs, delays, bus- and pin-constraints were modeled.

Furthermore, it was shown, that the determination of a feasible binding (and hence the whole system synthesis problem) is NP-complete (Theorem 7.1). Consequently, exact methods are impracticable. A solution to the synthesis problem was proposed based on Evolutionary Algorithms and tested on a real-world problem. The EA approach showed:

- In order to handle the complexity and the specific problems of the synthesis problem, the EA must be extended by problem-dependent heuristics for allocation and binding. This is not unique to EAs but necessary to solve all difficult optimization problems.
- The EA is capable of handling the complex search spaces arising in system synthesis and seems to be a robust optimization tool.
- Due to the population-based concept of the EA the design space is explored and all Pareto points of a problem can be obtained in a single optimization run.
- The EA with meaningful parameter settings outperformed Monte Carlo and Hill Climbing approaches on the investigated problem.

10.2 Outlook

10.2.1 Theory of Evolutionary Algorithms

A prediction of the behaviour of EAs is very desirable. However, the accuracy of a prediction is directly related to the assumptions made. An example is the prediction of the convergence speed of a Genetic Algorithm optimizing the ONEMAX function (see Chapter 4.7). The restriction of a single specific objective function (the ONEMAX function) allowed a very accurate prediction of the convergence time. However, the results are specific to this problem and difficult to generalize for other optimization problems.

Similarly, some results obtained in this thesis might be too general to help for a specific problem. The prediction of the generational behaviour of GP was concentrated on a single aspect of GP, the redundancy or bloating. Although the results gave new insights in the influence of introns on the behaviour of GP, they are not capable to predict the performance of GP for a particular problem. A possible direction of future research could be the refinement of the redundancy analysis.

10.2.2 System Synthesis

Despite the fundamental results achieved with the new methodology several extensions remain to be addressed:

- The model is not able to handle memory demand. Currently, the model assumes that each functional resource has its own (sufficiently large) local memory to store the incoming data until they are processed and the outgoing data until they are transmitted. Given a schedule, the memory demand can easily be calculated, however, it is not optimized. The data needs to be stored from the production of results until the last receiver of the data has read the result. This time is usually termed *life time* of variables [Bachmann, 1995]. This time should be as short as possible to minimize the memory demand of the implementation. However, this is a problem of the scheduler and not a problem of allocation and binding and hence does not affect the optimization tool based on Evolutionary Algorithms.

- Although the model is able to handle communication there are still some details left that might be of interest: Usually communication needs not only a bus-resource but also blocks sender and receiver. Consequently, the simultaneous execution of two or more nodes needs to be modeled which is not possible within the current model. However again, this mainly concerns the scheduler.

The proposed model was designed for the synthesis of hardware/software systems. Nevertheless, the application domain could be widened to include planning and scheduling problems in industry (such as job-shop scheduling) which have a similar nature.

Appendix A

Glossary

A	Accuracy (approximation error)
α	Schedule
β	Binding
c	Basis for exponential ranking
C	Complexity measure (usually number of nodes in a tree)
δ	Parsimony pressure (Occam factor)
d	Number of subgraphs in a specification graph G_S
D	Loss of diversity
ϵ	Accuracy of approximation
e	Edge of a graph
E	Set of edges in a graph
E_M	Mapping edges ($E_M \subseteq E_S$)
η^-	Selection probability of worst fit individual in ranking selection
f	Fitness value
$f(J)$	Fitness value of individual J
$\vec{\phi}$	Vector of fitness values of a population ($\phi_1 = f(J_1), \vec{\phi} \in \boldsymbol{R}^N$)
$\vec{f}$	Vector of unique fitness values in a fitness distribution ($\vec{f} \in \boldsymbol{R}^n$)

$\mathcal{G}(\mu, \sigma)$	Gaussian distribution with mean μ and variance σ^2
$G(V, E)$	Graph with node set V and arc set E
$G_S(V_S, E_S)$	Specification graph
$G_P(V_P, E_P)$	Problem graph ($V_P \subseteq V_S, V_E \subseteq V_P$)
$G_A(V_A, E_A)$	Architecture graph ($V_A \subseteq V_S, V_A \subseteq V_P$)
$G_C(V_C, E_C)$	Chip graph ($V_C \subseteq V_S, V_C \subseteq V_P$)
g_i	Constraints (global optimization problem)
h	Objective function (global optimization problem)
k	Generation counter
k_{max}	Maximum number of generations
k_c	Convergence time (in generations) for the ONEMAX example
I	Selection intensity
J	Individual, point in search space
$\vec{J}$	Population
$\mathbf{J}$	Space of all possible individuals, search space
$\tilde{\mathbf{J}}$	Feasible region
J^*	Redundancy class
κ	Parameter of exponential ranking ($\kappa = c^N$)
l	Size of training data set
L	Latency (makespan) of a schedule
M	Average Population Fitness
n	Number of unique fitness values in a population
N	Population size
$\mathcal{N}$	Neighborhood function
Ω	Selection method
Ω_E	Exponential ranking selection
Ω_T	Tournament selection
Ω_Γ	Truncation selection
Ω_P	Proportional selection
Ω_R	Ranking selection
p_c	Crossover probability
p_m	Mutation probability
p_r	Proportion of redundant edges
p_s	Probability of an individual to survive crossover
$\varphi(v)$	Iteration index of operation v

P	Period of a iterative schedule
q	Number of constraints in global optimization
R	Reproduction rate
$\vec{R}$	Vector of reproduction rates ($\vec{R} \in [0,1]^n$)
$\boldsymbol{R}$	Set of real numbers
ρ	Efficiency
s	(Discrete) Fitness Distribution
$\bar{s}$	(Continuous) Fitness Distribution
S	Cumulative (Discrete) Fitness Distribution
$\bar{S}$	Cumulative (Continuous) Fitness Distribution
$\bar{\sigma}$	Mean Variance of the Population Fitness
t	Tournament size
T	Truncation threshold
τ	Schedule
Θ	Type attribute of the nodes of a problem graph
$v(J)$	Number of nodes is individual J (if J is represented as tree)
v_{max}	maximum number of nodes in a tree
Φ	Selection variance
V	Set of vertices in a graph
$\mathbb{Z}$	Set of integers

Appendix B

Deriving Approximation Formulas Using Genetic Programming

In this appendix the derivation of the approximation formulas for the selection variance of tournament selection (4.37), the selection intensity of exponential ranking selection (4.62), and the selection variance of exponential ranking (4.63) are described.

In general the approach is the same as described in Chapter 5.4: the Genetic Programming optimization method is used to obtain a symbolic approximation of numerical data. A certain number of data points (x_i, y_i) are given and an analytic expression is searched that approximates the functional dependence $y = J(x)$.

The basic setup is here shortly summarized:

- The fitness function (that has to be minimized) is given by:

$$f(J) = A_c(J) + \delta\, C_v(J) + j\text{PENALTY} \qquad \text{(B.1)}$$

where $j \in \{0, \ldots, l\}$ gives the number of occurrences of mathematical exceptions during evaluation of the fitness function.

- The maximum relative error over all data points is used as accuracy measure:

$$A_c(J) = \max_{i=1,\ldots,l} \left|\frac{J(\vec{x}_i) - y_i}{y_i}\right| \tag{B.2}$$

- The complexity measure is the relative number of nodes in the tree J, i.e.:

$$C_v(J) = \frac{v(J)}{v_{max}}$$

- The penalty factor δ is adjusted according to the adaptive method suggested in [Zhang and Mühlenbein, 1995] (see Chapter 5.2.2 on page 113).

- All individuals with a fitness value better than a certain threshold f_{th} are selected for constant optimization. The optimization consists of sequentially changing all constants in the tree by a fixed amount $\xi = 0.1$ and accepting the new value as long as the fitness is improved or a maximum number of optimization steps s_{max} have been performed.

- Other parameter settings are summarized in Table B.1.

B.1 Approximating the Selection Variance of Tournament Selection

The operators and terminal provided to the optimization method for this problem are

$$\begin{aligned} F &= \{Plus, Subtract, Times, Divide, Log, Sqrt\} \\ T &= \{t, \pi, RFPC\} \end{aligned}$$

Number of runs	20
Population size	$N = 3000$
Selection method	Tournament Selection 10
Crossover method	crossover limiting tree size
Crossover probability	$p_c = 0.9$
Max. generations	$k_{max} = 100$
Max. tree size	$v_{max} = 45$
Constant-optimization threshold	$f_{th} = 1.0$
Maximum number of constant-optimization steps	$s_{max} = 10$
Penalty value	$PENALTY = 100.000$

Table B.1: *The global parameters of GP for the symbolic regression problem.*

where $RFPC$ is a random floating point number in the range from [-10,10] once determined at creation time of the population. These sets were chosen with some knowledge in mind about the possible dependency.

The GP found the following approximation with an relative error of less than 0.56%:

$$\Phi_T(t) \approx \frac{0.918}{\ln(1.186 + 1.328t)} \qquad ; t \in \{1, \ldots, 30\} \qquad (4.37)$$

Table B.1 displays the numerical calculated values for the selection variance, the approximation by (4.37) and the relative error of the approximation for the tournament sizes $t = 1, \ldots, 30$.

B.2 Approximating the Selection Intensity of Exponential Ranking Selection

The operators and terminal provided to the optimization method for this problem were

$$
\begin{aligned}
F &= \{Plus, Subtract, Times, Divide, Log, Sqrt, Exp, Power\} \\
T &= \{\kappa, \pi RFPC\}
\end{aligned}
$$

were $RFPC$ is a random floating point number in the range from [-10,10] once determined at creation time of the population.

The GP found the following approximation with an relative error of 1.8 %:

$$
I_E(\kappa) \approx \frac{\ln \kappa}{-2.548 - 1.086\sqrt{\kappa} + 0.4028 \ln \kappa} \qquad (4.62)
$$

Table B.2 displays again the numerical calculated values for the selection intensity, the approximation by (4.62) and the relative error of the approximation.

B.3 Approximating the Selection Variance of Exponential Ranking Selection

The operators and terminal provided to the optimization method for this problem were

$$
\begin{aligned}
F &= \{Plus, Subtract, Times, Divide, Log, Sqrt, Exp, Power\} \\
T &= \{\kappa, \pi, RFPC\}
\end{aligned}
$$

were $RFPC$ is a random floating point number in the range from [-10,10] once determined at creation time of the population.

One solution with an accuracy of 2.8% found by GP was

$$\Phi_E(\kappa) \approx 0.1 + \kappa^{\frac{1}{4}}(e^{\frac{-\kappa}{2}} + 0.2428\kappa^{-\frac{15}{64}}) \quad (4.63)$$

Table B.3 displays again the numerical calculated values for the selection variance, the approximation by (4.63) and the relative error of the approximation.

Tournament size t	$\Phi_T(t)$	Approximation Equation (4.37)	rel. error (E_c [in %])
1	1	0.995796	0.420351
2	0.6816901138160949	0.682024	0.0490256
3	0.5594672037973512	0.558777	0.123311
4	0.4917152368747342	0.490517	0.243667
5	0.4475340690206629	0.446183	0.30188
6	0.4159271089832759	0.414598	0.31946
7	0.3919177761267493	0.390695	0.312022
8	0.3728971432867331	0.371818	0.289264
9	0.357353326357783	0.356434	0.25714
10	0.3443438232607686	0.343588	0.219365
11	0.3332474427030835	0.332653	0.178296
12	0.3236363870477149	0.323198	0.135459
13	0.3152053842122778	0.314916	0.0918529
14	0.3077301024704087	0.307582	0.0481354
15	0.3010415703137873	0.301027	0.00474231
16	0.2950098090102839	0.295122	0.0380403
17	0.2895330036877659	0.289765	0.0800272
18	0.2845301297414324	0.284875	0.121102
19	0.2799358049283933	0.280387	0.161197
20	0.2756966156185853	0.276249	0.200277
21	0.2717684436810235	0.272416	0.238329
22	0.2681144875238161	0.268853	0.275356
23	0.2647037741277227	0.265528	0.311374
24	0.2615098815029825	0.262416	0.346458
25	0.2585107005876581	0.259494	0.3805
26	0.2556866644747772	0.256744	0.413621
27	0.2530210522851858	0.254149	0.445846
28	0.2504992994478195	0.251695	0.477206
29	0.2481086538596352	0.249368	0.50773
30	0.245837896441101	0.247159	0.537449

Table B.1: Approximation of the selection variance of tournament selection.

κ	$I_E(\kappa)$	Approximation Equation (4.62)	rel. error (E_c [in %])
1.10^{-20}	2.211866099272095	2.18288	1.31037
1.10^{-19}	2.191269514581677	2.16909	1.01198
1.10^{-18}	2.169380995296958	2.15398	0.710068
1.10^{-17}	2.146037077802608	2.13733	0.405801
1.10^{-16}	2.1210419443645	2.1189	0.100803
1.10^{-15}	2.094158402171831	2.0984	0.202687
1.10^{-14}	2.065095511766934	2.07545	0.50159
1.10^{-13}	2.033491261026872	2.04959	0.791679
1.10^{-12}	1.998887711729481	2.02022	1.06713
1.10^{-11}	1.960694345596714	1.98657	1.31991
1.10^{-10}	1.91813222188039	1.94765	1.53885
1.10^{-9}	1.870145548458747	1.9021	1.70842
1.10^{-8}	1.815254927827707	1.84806	1.80692
1.10^{-7}	1.751298381117838	1.7829	1.80424
1.10^{-6}	1.674944877679115	1.70274	1.6593
0.00001	1.580680636857533	1.60155	1.32024
0.0001	1.458497508852043	1.46928	0.739363
0.001	1.28826023480579	1.28765	0.0476957
0.01	1.027561822431405	1.02078	0.660161
0.015848932	0.958452065521477	0.951907	0.682847
0.025118864	0.882110337691857	0.876351	0.65289
0.039810717	0.7979436139674149	0.793406	0.568601
0.063095734	0.7055294262673961	0.702435	0.438621
0.1	0.60471903	0.602972	0.288818
0.125892541	0.5512204082282337	0.55	0.221411
0.158489319	0.4957454162357966	0.494897	0.171069
0.199526231	0.4383980076382858	0.437741	0.149945
0.251188643	0.3793153742211207	0.378659	0.172994
0.316227766	0.318668032179538	0.317845	0.258142
0.398107171	0.2566586452109129	0.255564	0.426334
0.501187234	0.1935194902774893	0.192162	0.701405
0.630957344	0.1295085639502477	0.128071	1.10972
0.794328235	0.06490442854651867	0.0638143	1.67952

Table B.2: Approximation of the selection intensity of exponential ranking selection.

κ	$\Phi(\kappa)$	Approximation Equation (4.63)	rel. error (E_c [in %])
1.10^{-20}	0.224504	0.218246	2.78781
1.10^{-19}	0.227642	0.222585	2.22139
1.10^{-18}	0.231048	0.227089	1.71378
1.10^{-17}	0.234767	0.231768	1.27747
1.10^{-16}	0.238849	0.236636	0.926457
1.10^{-15}	0.243361	0.241716	0.676054
1.10^{-14}	0.248386	0.247039	0.542158
1.10^{-13}	0.254032	0.25266	0.539747
1.10^{-12}	0.260441	0.25867	0.679935
1.10^{-11}	0.267807	0.265224	0.964636
1.10^{-10}	0.276403	0.272596	1.3774
1.10^{-9}	0.286619	0.281264	1.86857
1.10^{-8}	0.299052	0.292074	2.3333
1.10^{-7}	0.314661	0.306527	2.58512
1.10^{-6}	0.335109	0.327281	2.3359
0.00001	0.363607	0.35906	1.25048
0.0001	0.407156	0.410251	0.760135
0.001	0.482419	0.495697	2.75254
0.01	0.624515	0.640593	2.57458
0.0158489	0.664523	0.679587	2.2669
0.0251189	0.70839	0.722356	1.97162
0.0398107	0.755421	0.768753	1.7649
0.0630957	0.804351	0.818163	1.71712
0.1	0.853263	0.869135	1.86015
0.125893	0.876937	0.894387	1.98994
0.158489	0.899606	0.918798	2.1334
0.199526	0.920882	0.941647	2.25493
0.251189	0.940366	0.962004	2.30103
0.316228	0.957663	0.978695	2.19622
0.398107	0.972403	0.990288	1.83925
0.501187	0.98425	0.995084	1.10067
0.630957	0.992927	0.991173	0.17664
0.794328	0.998221	0.976548	2.17111

Table B.3: Approximation of the selection variance of exponential ranking selection.

Appendix C

YAGPLIC Implementation

The experiments carried out in this thesis are based on YAGPLIC [Blickle, 1995], a library for genetic programming. The main features of this library are:

- Written in C for maximum performance.
- Object-oriented user-interface.
- Up to 32 data-types possible in a tree enabling type-consistent crossover (strongly typed genetic programming).
- Several selection and crossover schemes implemented.
- Advanced operators to mutate constants in a tree.
- Two parallelization schemes implemented using a pool of workstations and PVM [Geist *et al.*, 1994]:
  - Distributed calculation of the fitness function.

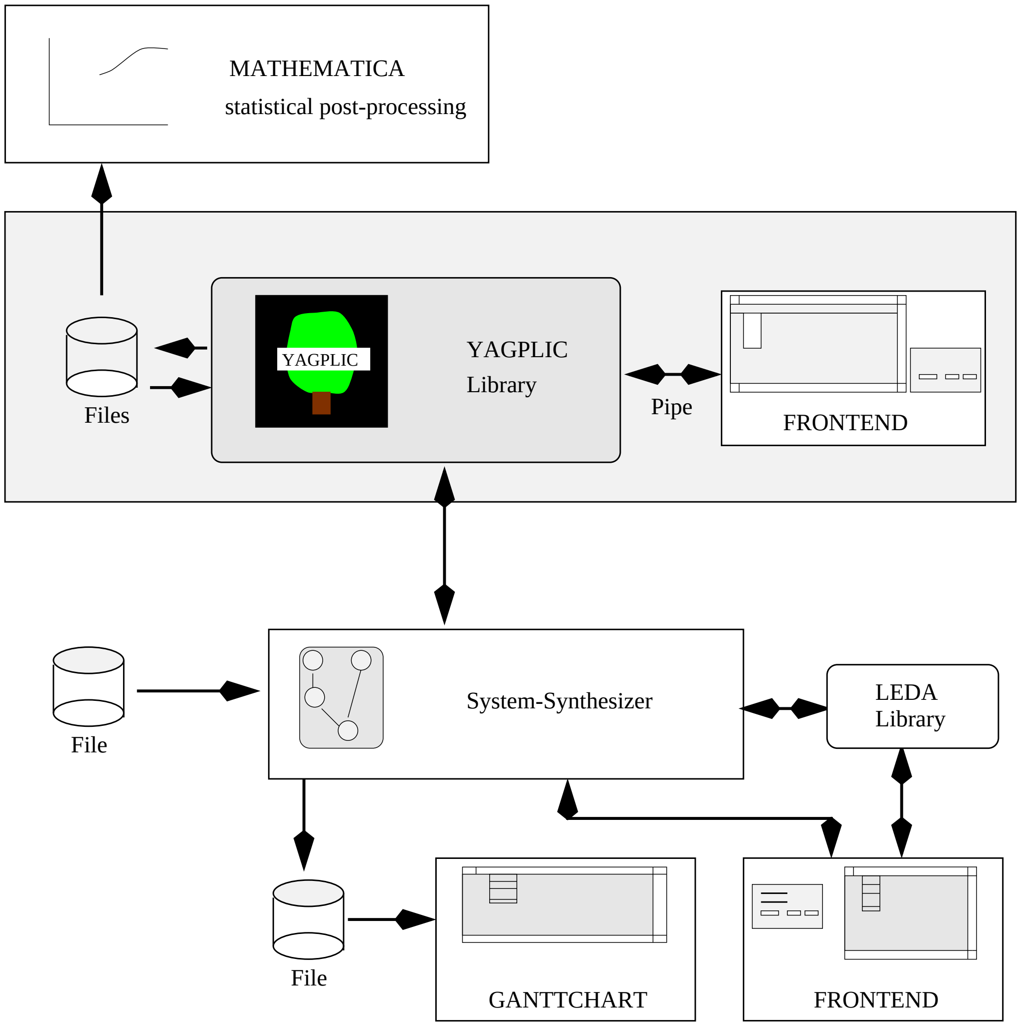


Figure C.1: *Overview of the components of the SSEA system synthesizer.*

  - Distributing the population across several workstations in various topologies (ring, torus, etc).

  More detail on the parallelization can be found in [Frey and Vaterlaus, 1996].

- Extensive output of statistical data for post processing with MATHEMATICA.

- C++ compatibility.

A graphical user-interface (front-end) for YAGPLIC is currently under development. Originally, this library was designed for the Genetic Programming optimization method only, i.e., designed to handle trees as individuals. However, due to the object-oriented design it is possible to extend the library to an arbitrary representation of an individual. This fact has been used to adapt the system-synthesis problem.

The system-synthesis Evolutionary Algorithm (SSEA) is implemented in C++ using a "library of efficient data types and algorithms" (LEDA), developed at the Max-Planck-Institut für Informatik [Näher and Uhrig, 1996]. This library provides classes for several data types, for example, graphs, lists and maps.

The core of SSEA is the class `implementation` that provides the complete handling of an individual's genotype and phenotype. It considers only a single mapping task from a graph G_P to a graph G_A. More levels can easily be handled by combining several instances of this class. The class is combined with YAGPLIC and a graphical user interface written in LEDA. Fig. C.1 gives an overview of the main components of the SSEA realization. Fig. C.2 shows a screen dump of the system.

In the config window the main parameter settings are adjusted:

- the specification file containing the specification graph G_S and delay and cost specifications.
- the optimization goal, namely Pareto optimization, period minimization under cost constraint, cost minimization under period constraints, average load maximation, and weighted sum optimization of cost and period.
- the maximum number of iteration for the periodic scheduler.
- various constraints.

The main window displays the current population as points in the design space. A particular implementation can be chosen from

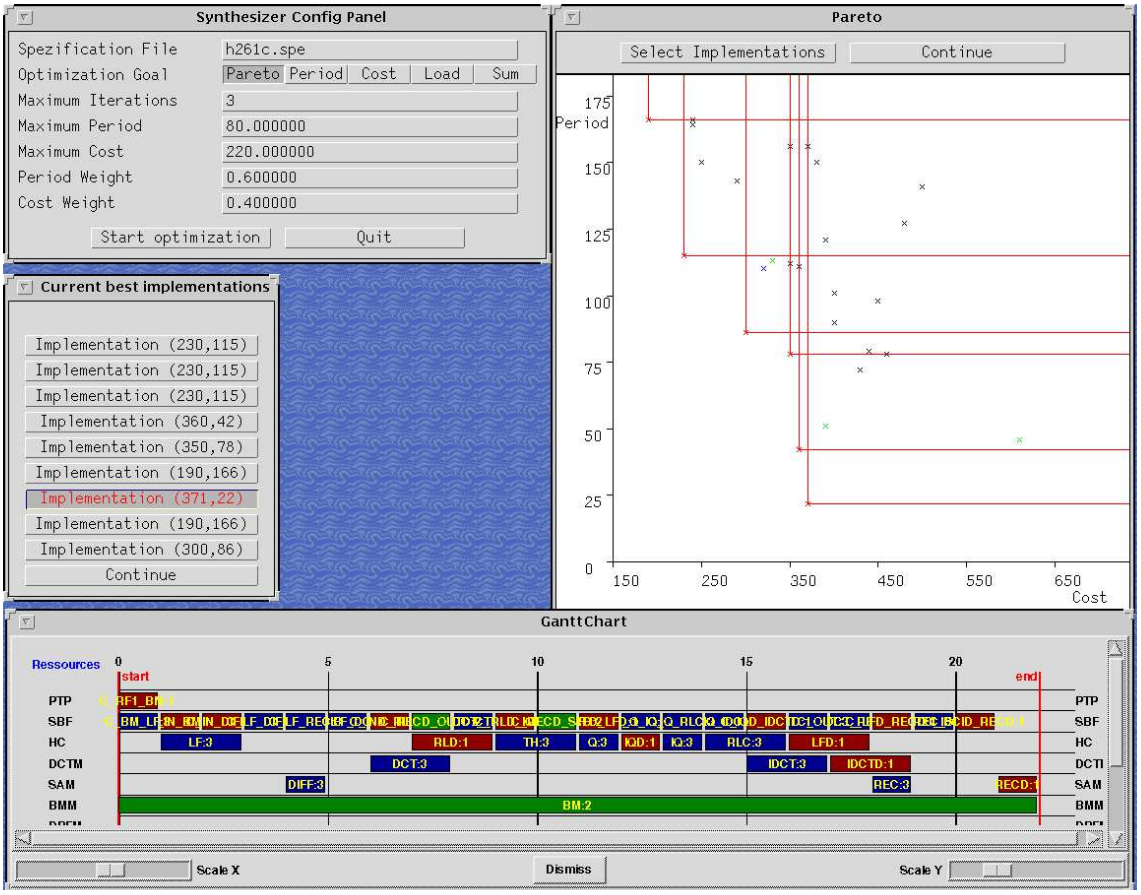


Figure C.2: *Screen dump of the SSEA system synthesizer.*

the panel displaying the current best individuals. The Ganttchart of the corresponding implementation is then drawn in the Ganttchart window.

Appendix D

Used Integrals

$$\int_{-\infty}^{\infty} e^{-\frac{x^2}{2}} \int_{-\infty}^{x} e^{-\frac{y^2}{2}} dy\, dx = \pi \tag{D.1}$$

$$\int_{-\infty}^{\infty} xe^{-\frac{x^2}{2}} = 0 \tag{D.2}$$

$$\int_{-\infty}^{\infty} xe^{-\frac{x^2}{2}} \int_{-\infty}^{x} e^{-\frac{y^2}{2}} dy\, dx = \sqrt{\pi} \tag{D.3}$$

$$\int_{-\infty}^{\infty} xe^{-\frac{x^2}{2}} \left(\int_{-\infty}^{x} e^{-\frac{y^2}{2}} dy\right)^2 dx = \sqrt{2}\pi \tag{D.4}$$

$$\int_{-\infty}^{\infty} x^2 e^{-\frac{x^2}{2}}\, dx = \sqrt{2\pi} \tag{D.5}$$

$$\int_{-\infty}^{\infty} x^2 e^{-\frac{x^2}{2}} \int_{-\infty}^{x} e^{-\frac{y^2}{2}} dy\, dx = \pi \tag{D.6}$$

$$\int_{-\infty}^{\infty} t\, e^{-\frac{x^2}{2}} \left(\int_{-\infty}^{x} e^{-\frac{y^2}{2}} dy\right)^{t-1} dx = (2\pi)^{\frac{t}{2}} \tag{D.7}$$

Bibliography

[Alander, 1995a] Jarmo T. Alander. Indexed bibliographies of genetic algorithms. Technical Report 94-1-*, Department of Information Technology and Industrial Management, University of Vaasa, P.O. Box 700, FIN-65101 Vaasa, Finland, 1995.

[Alander, 1995b] Jarmo T. Alander. An indexed bibliography of genetic algorithms - theory and analysis. Technical report, Department of Information Technology and Industrial Management, University of Vaasa, P.O. Box 700, FIN-65101 Vaasa, Finland, 1995.

[Altenberg, 1995] Lee Altenberg. The schema theorem and price's theorem. In L. Darrell Whitley and Michael D. Vose, editors, *Foundantions of Genetic Algorithms 3*. Morgan Kaufmann, 1995.

[Andre and Teller, 1996] David Andre and Astro Teller. A study in program response an the nagive effects of introns in genetic programming. In John Koza, David Goldberg, David Fogel, and Rick Riolo, editors, *Genetic Programming 1996: Proceedings of the First Annual Conference*. The MIT-Press, 1996.

[Arnold *et al.*, 1992] B.C. Arnold, N. Balakrishnan, and H. N. Nagaraja. *A First Course in Order Statistics.* Wiley Series in Probability and Mathematical Statistics, Wiley, New York, 1992.

[Bachmann, 1995] A. Bachmann. *Syntheseverfahren für heterogene Multiprozessorsysteme.* PhD thesis, Department of Electrical Engineering, Univ. des Saarlandes, Saarbrücken, July 1995.

[Bäck and Schwefel, 1993] Thomas Bäck and Hans-Paul Schwefel. An overview of evolutionary algorithms for parameter optimization. *Evolutionary Computation*, 1(1):1–23, 1993.

[Bäck *et al.*, 1993] Thomas Bäck, Frank Hoffmeister, and Hans-Paul Schwefel. Applications of evolutionary algorithms. Technical Report SYS -2/92, System Analysis Research Group, University of Dortmund, Department of Computer Science, D-44221 Dortmund, Germany, 1993.

[Bäck, 1994a] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice*. PhD thesis, Fachbereich Informatik, Universität Dortmund, 1994.

[Bäck, 1994b] Thomas Bäck. Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. In *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence (ICEC94)*, pages 57–62, 1994.

[Bäck, 1995] Thomas Bäck. Generalized convergence models for tournament- and (μ,λ)-selection. In L. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA95)*, San Francisco, CA, 1995. Morgan Kaufmann Publishers.

[Baker, 1987] J. E. Baker. Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 14–21, Cambridge, MA, 1987. Lawrence Erlbaum Associates.

[Baker, 1989] J. E. Baker. *An Analysis of the Effects of Selection in Genetic Algorithms*. PhD thesis, Graduate School of Vanderbilt University, Nashville, Tennessee, 1989.

[Barros and Rosenstiel, 1992] E. Barros and W. Rosenstiel. A method for hardware software partitioning. In *Proc. 1992 COMPEURO: Computer Systems and Software Engineering*, pages 580–585, The Hague, Netherlands, May 1992.

[Beyer, 1995a] Hans-Georg Beyer. How GAs do NOT work understanding GAs without schemata and building blocks. Technical Report SYS -2/95, System Analysis Research Group, Universität Dortmund, D-44221 Dortmund, 1995.

[Beyer, 1995b] Hans-Georg Beyer. Toward a theory of evolution strategies: On the benefits of sex - the $(\mu/\mu, \lambda)$ theory. *Evolutionary Computation*, 3(1):81–111, 1995.

[Blickle and Thiele, 1994] 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)*, pages 33–38. Max-Planck-Institut für Informatik (MPI-I-94-241), 1994.

[Blickle and Thiele, 1995] Tobias Blickle and Lothar Thiele. A mathematical analysis of tournament selection. In L. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA95)*, pages 9–16, San Francisco, CA, 1995. Morgan Kaufmann Publishers.

[Blickle, 1995] Tobias Blickle. *YAGPLIC - User Manual.* Computer Engineering and Communication Networks Lab (TIK), Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, CH-8092 Zurich, 1995.

[Blickle, 1996] Tobias Blickle. Evolving compact solutions in genetic programming: A case study. In Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel, editors, *Parallel Problem Solving From Nature IV. Proceedings of the International Conference on Evolutionary Computation.*, pages 564–573, Heidelberg, Germany, 1996. Springer-Verlag, LNCS 1141.

[Brill *et al.*, 1992] F. Z. Brill, D. E. Brown, and W. N. Martin. Fast genetic selection of features for neural network classifiers. *IEEE Transactions on Neural Networks*, 2(3):324–328, March 1992.

[Buchenrieder *et al.*, 1995] K. Buchenrieder, A. Sedlmeier, and C. Veith. Codes a framework for modeling heterogeneous systems.

In J. Rozenbilt and K. Buchenrieder, editors, *Codesign: Computer-Aided Software/Hardware Engineering*, pages 378–392. IEEE Press, Piscataway, NJ, U.S.A., 1995.

[Bulmer, 1980] M.G. Bulmer. *The Mathematical Theory of Quantitative Genetics.* Clarendon Press, Oxford, 1980.

[Camposano and Brayton, 1987] R. Camposano and R. K. Brayton. Partitioning before logic synthesis. In *Proc. ICCAD*, 1987.

[Cramer, 1985] N. L. Cramer. A representation for the adaptive generation of simple sequential programs. In J.J. Grefenstette, editor, *Proceedings of an International Conference on Genetic Algorithms and Their Applications*, Hillsdale NJ, 1985. Lawrence Erlbaum Associates.

[Crow and Kimura, 1970] J.F. Crow and M. Kimura. *An Introduction to Population Genetics Theory.* Harper and Row, New York, 1970.

[D'Ambrosio and Hu, 1994] J. G. D'Ambrosio and X. Hu. Configuration-level hardware/software partition for real-time embedded systems. In *Proc. of CODES/CASHE'94, Third Intl. Workshop on Hardware/Software Codesign*, pages 34–41, Grenoble, France, September 1994.

[D'Ambrosio *et al.*, 1995] J. G. D'Ambrosio, X. Hu, and G. W. Greenwood. An evolutionary approach to configuration-level hardware/software partitioning. Technical Report R & D-8465, General Motors Corporation, R & D Center, 30500 Mound Road, Box 9055, Warren, Michigan 40090-9055, December 1995.

[Davis, 1991] Lawrence Davis. *Handbook of Genetic Algorithms.* Van Nostrand Reinhold, 115 Fifth Avenue, New York, 1991.

[de la Maza and Tidor, 1993] Michael de la Maza and Bruce Tidor. An analysis of selection procedures with particular attention paid to proportional and bolzmann selection. In Stefanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 124–131, San Mateo, CA, 1993. Morgan Kaufmann Publishers.

[DeMicheli, 1994] G. DeMicheli. *Synthesis and Optimization of Digital Circuits.* Mc Graw Hill, New York, 1994.

[Ernst *et al.*, 1994] R. Ernst, J. Henkel, and T. Benner. Hardware-software cosynthesis for microcontrollers. *IEEE Design & Test of Computers*, pages 64–75, December 1994.

[Fogel, 1991] David B. Fogel. *System Identification through Simulated Evolution: A Machine Learning Approach to Modeling.* Ginn Press, Needham Heights, 1991.

[Fogel, 1992] David B. Fogel. *Evolving Artificial Intelligence.* PhD thesis, University of California, San Diego, CA, 1992.

[Fox and McMahon, 1991] B. R. Fox and M. B. McMahon. Genetic operators for sequencing problems. In Gregory J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 284–300. Morgan Kaufmann Publishers, 1991.

[Frey and Vaterlaus, 1996] Lukas Frey and Oliver Vaterlaus. Verteiltes Genetisches Programmieren. Technical report, Semesterarbeit am Institut TIK, ETH Zürich, Gloriastrasse 35, CH-8092 Zurich, 1996.

[Garey and Johnson, 1979] Michael R. Garey and David S. Johnson. *Computers and Intractability - A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, New York, 1979.

[Geist *et al.*, 1994] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam. Pvm 3 user's guide and reference manual. Technical Report ORNL/TM-12187, Oak Ridge National Laboratory, Oak ridge, Tennessee 37831, 1994.

[Goldberg and Deb, 1991] David E. Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. In G. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 69–93, San Mateo, 1991. Morgan Kaufmann.

[Goldberg, 1989] David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, 1989.

[Gordon *et al.*, 1992] V. Scott Gordon, Darrell Whitley, and A. P. Wim Böhm. Dataflow parallelism in genetic algorithms. In Reinhard Männer and B. Manderick, editors, *Parallel Problem Solving from Nature - PPSN II*, pages 533–542, Amsterdam, 1992. North-Holland.

[Gorges-Schleuter, 1989] M. Gorges-Schleuter. ASPARAGOS: An asynchronous parallel genetic optimization strategy. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, San Mateo, CA, 1989. Morgan Kaufmann Publishers.

[Grefenstette and Baker, 1989] John J. Grefenstette and James E. Baker. How genetic algorithms work: A critical look at implicit parallelism. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 20 – 27, San Mateo, CA, 1989. Morgan Kaufmann Publishers.

[Gupta and Micheli, 1992] R. Gupta and G. De Micheli. System-level synthesis using re-programmable components. In *Proc. of the European Conference on Design Automation (EDAC)*, pages 2–7, 1992.

[Hardt and Camposano, 1995] W. Hardt and R. Camposano. Specification analysis for hw/sw-partitioning. In *Proc. GI/ITG Workshop Application of formal Methods during the Design of Hardware Systems*, pages 1–10, Passau, Germany, March 1995.

[Harik, 1994] Georges R. Harik. Finding multiple solutions in problems of bounded difficulty. Technical Report IlliGal report 94002, The Illinois Genetic Algorithms Laboratory, University of Illinois, Urbana-Champaign, 1994.

[Harik, 1995] Georges R. Harik. Finding multimodal solutions using restricted tournament selection. In Larry J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA6).* Morgan Kaufmann, 1995.

[Henrici, 1977] P. Henrici. *Applied and Computational Complex Analysis*, volume 2. A Wiley-Interscience Series of Texts, Monographs, and Tracts, 1977.

[Holland, 1975] John H. Holland. *Adaption in Natural and Artificial Systems.* The University of Michigan Press, Ann Arbor, MI, 1975.

[Horst and Tuy, 1992] Reiner Horst and Hoang Tuy. *Global Optimization, Deterministic Approaches.* Springer, Berlin, 1992.

[Ismail *et al.*, 1994] T. B. Ismail, K. O'Brien, and A. A. Jerraya. Interactive system-level partitioning with PARTIF. In *Proc. of the European Conference on Design Automation (EDAC)*, pages 464–473, 1994.

[Janikow, 1995] Cezary Z. Janikow. A genetic algorithm for optimizing fuzzy decision trees. In L. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA95)*, pages 421–428, San Francisco, CA, 1995. Morgan Kaufmann Publishers.

[Joines and Houck, 1994] Jeffrey A. Joines and Christopher R. Houck. On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with ga's. In *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence (ICEC94)*, pages 579–584, 1994.

[Jong, 1975] K. A. De Jong. *An Analysis of the behaviour of a class of genetic adaptive systems.* PhD thesis, 1975.

[Jong, 1992] Kenneth A. De Jong. Are genetic algorithms function optimizers? In Reinhard Männer and B. Manderick, editors, *Parallel Problem Solving from Nature - PPSN II*, pages 3–13, Amsterdam, 1992. North-Holland.

[Jong, 1993] Kenneth A. De Jong. Genetic algorithms are not function optimizers. In L. Darrell Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 5–17, San Mateo, 1993. Morgan Kaufmann.

[Kalavade and Lee, 1994] A. Kalavade and E. A. Lee. A global criticality/local phase driven algorithm for the constrained hardware/software partitioning problem. In *Proc. of*

CODES/CASHE'94, Third Intl. Workshop on Hardware/Software Codesign, Grenoble, France, September 1994.

[Koza, 1992] John R. Koza. *Genetic programming: on the programming of computers by means of natural selection.* The MIT Press, Cambridge, Massachusetts, 1992.

[Koza, 1994a] John Koza. Architecture-altering operations for evolving the architecture of a multi-part program in genetic programming. Technical Report STAN-CS-TR-94-1528, Stanford Universitary Computer Science Department, 1994.

[Koza, 1994b] John R. Koza. *Genetic Programming II.* The MIT Press, Cambridge, Massachusetts, 1994.

[Küçükçakar and Parker, 1995] K. Küçükçakar and A. C. Parker. A methodology and design tools to support system-level VLSI design. *IEEE Trans. on VLSI systems*, 3(3):355–369, September 1995.

[Lagnese and Thomas, 1991] E. D. Lagnese and D. E. Thomas. Architectural partitioning for system level synthesis of integrated circuits. *IEEE Trans. on CAD*, 10(7):847–860, July 1991.

[Lengauer, 1990] Thomas Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout.* John Wiley and Sons, New York, 1990.

[Mahfoud, 1995] Samir W. Mahfoud. A comparison of parallel and sequential niching methods. In Larry J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA6)*, pages 136 – 143. Morgan Kaufmann, 1995.

[McFarland, 1986] M. C. McFarland. Using bottom-up design techniques in the synthesis of hardware from abstract behavioral descriptions. In *Proc. 23rd Design Automation Conference*, pages 474–480, June 1986.

[Michalewicz and Attia, 1994] Zbigniew Michalewicz and Naguib F. Attia. Evolutionary optimization of constrained problems. In Anthony V. Sebald and Lawrence J. Fogel, editors, *Proceedings of the Third Annual Conference on Evolutionary Programming*, pages 98–108. Word Scientific, 1994.

[Michalewicz and Janikow, 1991] Zbigniew Michalewicz and Cezary Z. Janikow. Handling constraints in genetic algorithms. In Richard K. Belew and Lashon B. Booker, editors, *Proceedings of the Forth International Conference on Genetic Algorithms*, pages 151 – 157, San Mateo, CA, 1991. Morgan Kaufmann Publishers.

[Michalewicz, 1992] Zbigniew Michalewicz. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer, Berlin, 1992.

[Michalewicz, 1995a] Z. Michalewicz. Genetic algorithms, numerical optimization, and constraints. In Larry J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA6)*, pages 151–158. Morgan Kaufmann, 1995.

[Michalewicz, 1995b] Z. Michalewicz. A survey of contraint handling techniques in evolutionary computation methods. In *Proceedings of the 4th annual Conference on Evolutionary Programming*, Cambridge, MA, 1995. MIT Press.

[Miller and Goldberg, 1995] Brad L. Miller and David E. Goldberg. Genetic algorithms, tournament selection, and the effects of noise. Technical Report 95006, The Illinois Genetic Algorithm Laboratory, University of Urbana-Champaign, Urbana, Illinois 61801, 1995.

[Moran *et al.*, 1995] F. Moran, A. Moreno, J.J. Merelo, and P. Chacon, editors. *Orgy in the Computer: Multi-Parent Reproduction in Genetic Algorithms.* LNAI 929, Springer-Verlag, 1995.

[Mühlenbein and Schlierkamp-Voosen, 1993] Heinz Mühlenbein and Dirk Schlierkamp-Voosen. Predictive models for the breeder genetic algorithm. *Evolutionary Computation*, 1(1), 1993.

[Mühlenbein and Voigt, 1995] Heinz Mühlenbein and Hans-Michael Voigt. Gene pool recombination in genetic algorithms. In I. H. Osman and J. P. Kelly, editors, *Proceedings of the Metaheuristics Inter. Conf.*, Norwell, 1995. Kluwer Academic Publishers.

[Mühlenbein, 1991] Heinz Mühlenbein. Evolution in time and space – the parallel genetic algorithm. In Gregory J. E. Rawlins, editor, *Foundations of Genetic Algorithms*, pages 316–337. Morgan Kaufmann Publishers, 1991.

[Nagaraja, 1982] H. Nagaraja. Selection differentials. *Encyclopedia of Statistical Science*, pages 334–336, 1982.

[Näher and Uhrig, 1996] Stefan Näher and Christian Uhrig. The leda user manual version r 3.3.c. Technical report, Max-Planck-Institut für Informatik, 66123 Saarbrücken, Germany, 1996.

[Nissen, 1994] Volker Nissen. *Evolutionäre Algorithmen.* Deutscher Universitäts-Verlag, 1994.

[Nordin and Banzhaf, 1995] Peter Nordin and Wolfgang Banzhaf. Complexity compression and evolution. In L. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA95)*, pages 310–317, San Francisco, CA, 1995. Morgan Kaufmann Publishers.

[Nordin *et al.*, 1995] Peter Nordin, Frank Francone, and Wolfgang Banzhaf. Explicitly defined introns and destructive crossover in genetic programming. In Justinian P. Rosca, editor, *Proceedings of the Workshop on Genetic Programming (at Twelfth Int. Conf. on Machine Learning)*, pages 6–22. Technical Report 95.2, University of Rochester, 1995.

[O'Reilly, 1995] Una-May O'Reilly. *An Analysis of Genetic Programming.* PhD thesis, School of Computer Science, Carleton University, Ottawa, Ontario, 1995.

[Powell and Skolnick, 1993] David Powell and Michael M. Skolnick. Using genetic algorithms in engineering design optimization with non-linear constraints. In Stefanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 424–431, San Mateo, CA, 1993. Morgan Kaufmann Publishers.

[Rechenberg, 1973] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann-Holzboog, Stuttgart, Germany, 1973.

[Richardson *et al.*, 1989] Jon T. Richardson, Mark R. Palmer, Gunar Liepins, and Mike Hilliard. Some guidelines for genetic algorithms with penalty functions. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 191–197, San Mateo, CA, 1989. Morgan Kaufmann Publishers.

[Rogenmoser *et al.*, 1996] Robert Rogenmoser, Huber Kaeslin, and Tobias Blickle. Stochastic methods for transistor size optimization of cmos vlsi circuits. In Hans-Michael Voigt, Werner Ebeling, Ingo Rechenberg, and Hans-Paul Schwefel, editors, *Parallel Problem Solving From Nature IV*, pages 849–858, Heidelberg, Germany, 1996. Springer-Verlag, LNCS 1141.

[Rosca, 1995] Justinian P. Rosca. An analysis of hierarchical genetic programming. Technical Report 556, Computer Science Department, University of Rochester, Rochester, New York 14627, March 1995.

[Schoenauer and Xanthakis, 1993] Marc Schoenauer and Spyros Xanthakis. Constrained ga optimization. In Stefanie Forrest, editor, *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 573–580, San Mateo, CA, 1993. Morgan Kaufmann Publishers.

[Schwefel, 1975] H.-P. Schwefel. *Evolutionsstrategie und numerische Optimierung.* PhD thesis, Technische Universität Berlin, 1975.

[Shapiro *et al.*, 1994] Jonathan Shapiro, Adam Prügel-Bennett, and Magnus Rattray. A statistical mechanical formulation of the dynamics of genetic algorithms. In Terence C. Fogarty, editor, *Evolutionary Computing AISB Workshop.* Springer , LNCS 865, 1994.

[Stanley and Mudge, 1995] Timothy J. Stanley and Trevor Mudge. A parallel genetic algorithm for multiobjective microprocessor design. In Larry J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms (ICGA6)*, pages 597–604. Morgan Kaufmann, 1995.

[Stebler, 1996] Reto Stebler. Evolutionäre Algorithmen zur Regelung eines invertierten Pendels. Master's thesis, Institut TIK, ETH Zürich, CH-8092 Zürich, 1996.

[Syswerda and Palmucci, 1991] Gilbert Syswerda and Jeff Palmucci. The application of genetic algorithms to resource scheduling. In R.K. Belew and L.B. Booker, editors, *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 502–508, San Mateo, CA, 1991. Morgan Kaufmann Publishers.

[Syswerda, 1989] Gilbert Syswerda. Uniform crossover in genetic algorithms. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 2–9, San Mateo, CA, 1989. Morgan Kaufmann Publishers.

[Syswerda, 1993] Gilbert Syswerda. Simulated crossover in genetic algorithms. In L. Darrell Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 239–255, San Mateo, 1993. Morgan Kaufmann.

[Tackett, 1994] Walter Alden Tackett. *Recombination, Selection, and the Genetic Construction of Computer Programs*. PhD thesis, Faculty of the Graduate School, University of Southern California, 1994.

[Tackett, 1995] Walter Alden Tackett. Mining the genetic program. *IEEE Expert (Intelligent Systems and their Applications)*, 10(3), June 1995.

[Tamaki and Nishikawa, 1992] Hisashi Tamaki and Yoshikazu Nishikawa. A paralleld genetic algorithm based on a neighborhood model and its application to the jobshop scheduling. In Reinhard Männer and B. Manderick, editors, *Parallel Problem Solving from Nature - PPSN II*, pages 573–582, Amsterdam, 1992. North-Holland.

[Tanese, 1989] Reiko Tanese. Distributed genetic algorithms. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 434–439, San Mateo, CA, 1989. Morgan Kaufmann Publishers.

[Thierens and Goldberg, 1994a] D. Thierens and D. Goldberg. Convergence models of genetic algorithm selection schemes. In Yuval Davidor, Hans-Paul Schwefel, and Reinhard Männer, editors, *Parallel Problem Solving from Nature - PPSN III*, pages 119 – 129, Berlin, 1994. Lecture Notes in Computer Science 866 Springer-Verlag.

[Thierens and Goldberg, 1994b] Dirk Thierens and David Goldberg. Elitist recombination: an integrated selection recombination ga. In *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence (ICEC94)*, pages 508–512, 1994.

[Thomas *et al.*, 1993] D. E. Thomas, J. K. Adams, and H. Schmitt. A model and methodology for hardware-software codesign. *IEEE Design & Test of Computers*, 10(3):6–15, September 1993.

[Törn and Zilinskas, 1987] Aimo Törn and Antanas Zilinskas. *Global Optimization.* Lecture Notes in Computer Science 350, Springer, Berlin, Germany, 1987.

[Vahid and Gajski, 1992] F. Vahid and D. Gajski. Specification partitioning for system design. In *Proc. 29th Design Automation Conference*, pages 219–224, Anaheim, CA, June 1992.

[van Kemenade *et al.*, 1995] C.H.M. van Kemenade, J.N. Kok, and A.E. Eiben. Raising ga performance by simultaneous tuning of selective pressure and recombination disruptiveness. In *Proceedings of the 2nd IEEE World Conference on Evolutionary Computation.* IEEE Press, 1995.

[Vose, 1991] Michael D. Vose. Generalizing the notion of schema in genetic algorithms. *Artificial Intelligence*, 50:385–396, 1991.

[Whitley, 1989] Darrell Whitley. The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best. In J. David Schaffer, editor, *Proceedings of the Third International Conference on Genetic Algorithms*, pages 116 – 121, San Mateo, CA, 1989. Morgan Kaufmann Publishers.

[Wolpert and Macready, 1995] David H. Wolpert and Williaom G. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, The Santa Fe Institute, 1399 Hyde Park Road, Santa Fe, NM, 87801, 1995.

[Zanakis and Evans, 1981] Stelios H. Zanakis and James R. Evans. Heuristic "optimization": why, when, and how to use it. *Interfaces*, 11(5):84–91, 1981.

[Zhang and Mühlenbein, 1995] Byoung-Tak Zhang and Heinz Mühlenbein. Balancing accuracy and parsimony in genetic programming. *Evolutionary Computation*, 3(1):17–38, 1995.

Index