# Preface

This thesis describes the results of the research I have done during my time as a research assistant in the group of Prof. Oliver Vornberger at the Department of Mathematics and Computer Science at the University of Osnabrück. It is the outcome of a process starting with developing a genetic algorithm for the layout generation of VLSI circuits. A set of hybrid genetic algorithms for constrained packing and placement problems has been designed, which do not have very much in common with standard genetic algorithms. In this process evolution occurred, and finally the best methods survived.

During my studies I was financially supported by the German Federal Ministry for Education, Science, Research and Technology (BMBF) in the project 'HYBRID–Application of Parallel Genetic Algorithms in Combinatorial Optimization' under grant 01 IB 405 E3. Due to this grant there have been four students, Gunther Imeyer, Frank Lauxtermann, Curd Reinert, and Andreas Rother, working on programming interfaces for my hybrid genetic algorithms for input/output. Especially in case of the VLSI layout generation problem, the visualization-tool was very helpful for inspecting the produced layouts and proving the routability of the obtained placements.

I thank Oliver Vornberger for offering to me the opportunity to join his group and introducing me to the exciting field of evolutionary computation. I thank Frank Lohmeyer, Frank M. Thiesing, and Paul Tymann for reading and correcting parts of this thesis. I thank Erik Goodman for being the second referee of this work. Special thanks are due to the person I left alone during the weeks while working in Osnabrück, my wonderful fiancée Ina, who will hopefully marry me some day.

Osnabrück, October 1996 Volker Schnecke

# Contents

**1 Introduction** **5**

**2 Genetic Algorithms** **9**
2.1 Natural Evolution . . . 9
2.2 Origin of Evolutionary Computation . . . 10
2.3 A Simple Genetic Algorithm . . . 11
2.4 Extensions of the Simple Genetic Algorithm . . . 12
2.5 Strategy Adaptation . . . 14

**3 Packing and Placement** **17**
3.1 VLSI Layout Generation . . . 17
3.1.1 Macro cell layouts . . . 18
3.1.2 Placement . . . 20
3.1.3 Routing . . . 22
3.2 Bin-Packing . . . 24
3.3 The Facility Layout Problem . . . 26

**4 Advanced Concepts** **31**
4.1 Shape-Functions . . . 31
4.2 Slicing-Trees . . . 35
4.3 Iterated Matching . . . 37

**5 Hybrid Genetic Algorithms for Packing and Placement** **41**
5.1 Tree-Structured Genotype Representation . . . 41
5.1.1 Mutation . . . 42
5.1.2 Recombination . . . 44
5.1.3 Selection and replacement . . . 46
5.1.4 Parallelization and strategy adaptation . . . 47
5.2 A Hybrid Genetic Algorithm for the Bin-Packing Problem . . . 51
5.3 A Hybrid Genetic Algorithm for the Facility Layout Problem . . . 52
5.3.1 The preparation of the input data . . . 52
5.3.2 The initialization of the individuals . . . 54
5.3.3 The fitness of an individual . . . 55

5.4 A Hybrid Genetic Algorithm for VLSI Layout Generation . . . . . . . . . . . 55
5.4.1 The genotype representation . . . . . . . . . . . . . . . . . . . . . . . 55
5.4.2 Floorplanning . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 57
5.4.3 Integration of routing . . . . . . . . . . . . . . . . . . . . . . . . . . 57
5.4.4 Computation of the global routes . . . . . . . . . . . . . . . . . . . . 58
5.4.5 Genetic operators . . . . . . . . . . . . . . . . . . . . . . . . . . . . 59

**6 Empirical Results 61**
6.1 The Environment . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 61
6.2 The Benchmark Sets . . . . . . . . . . . . . . . . . . . . . . . . . . . . 62
6.2.1 VLSI layout generation . . . . . . . . . . . . . . . . . . . . . . . . . 62
6.2.2 Bin-packing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 63
6.2.3 Facility layout problem . . . . . . . . . . . . . . . . . . . . . . . . . 63
6.3 The Benefit of the Iterated Matching . . . . . . . . . . . . . . . . . . . . 63
6.4 The Benefit of the Gene-Pool Recombination Operator . . . . . . . . . . . 71
6.5 The Effect of the Strategy Adaptation . . . . . . . . . . . . . . . . . . . 73
6.6 Scalability . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 74
6.7 Final Results and Comparison . . . . . . . . . . . . . . . . . . . . . . . 77

**7 Conclusion 83**

**A The best results 97**
A.1 The Bin-Packing Problem . . . . . . . . . . . . . . . . . . . . . . . . . 97
A.2 VLSI Layout Generation . . . . . . . . . . . . . . . . . . . . . . . . . . 107
A.3 The Facility Layout Problem . . . . . . . . . . . . . . . . . . . . . . . . 110

# Chapter 1

# Introduction

Optimization is one of the most important tasks that can be solved with the help of computers. It is defined as identifying the best of all possible solutions for a given problem. This set of feasible solutions builds a search space. There must be an order defined in this search space to enable the comparison of solution candidates, this order has to be based on the objective function of the problem. The solution to an optimization problem is the solution with the optimal, for example the minimal, value for this objective function.

In the simplest case the search space is continuous without any restrictions. Optimization problems of this kind are called free optimization problems, most of them can be solved exactly or approximately exact using numerical techniques. If the solution space is discrete, the corresponding optimization problems are said to be combinatorial. Combinatorial optimization is much more complex than finding solutions in a continuous space. Heuristic search techniques from artificial intelligence are very successful in solving combinatorial optimization problems.

The problems dealt with in this work are both combinatorial and constrained optimization problems. They are packing and placement problems with additional constraints. The unconstrained variant of these problems can be formulated as finding a non overlapping arrangement for a set of rectangular items on a minimal planar area.

The simplest problem is the two-dimensional bin-packing problem. Here a set of rectangular blocks has to be packed inside a bin with fixed width and unlimited height. The objective is to find a pattern with minimal packing height. Apart from the problem of arranging the blocks in a very compact mode, the constraint introduced by the fixed width of the bin has to be taken into account.

The second optimization problem is the generation of layouts for VLSI circuits. In this application rectangular cells, that represent building blocks of an electronic circuit, have to be arranged on a minimal layout area. Interconnection wirings for the signal-nets connecting these cells have to be routed on the layout surface between the cells. This interconnectivity of the cells has a high influence on the solution quality, since each additional track for a signal-net between any pair of cells might increase the overall layout area. It is a very complex optimization problem, since it is not possible to consider the exact demand of routing space during the placement of the cells.

The last constrained optimization problem is the facility layout problem. Here no fixed blocks are given, but flexible shaped rectangular facilities, which have to be placed on a planar site. These facilities have to be arranged in a compact mode, too, but the objective function is also determined by flow costs between the centres of the facilities. Additionally, the optimal shapes for the flexible facilities in a particular arrangement have to be determined.

For solving the layout generation task and the facility layout problem, usually multi-staged techniques are used. In each stage only parts of the constraints are directly considered, while the others are approximated. In the layout generation process first the cells are placed, and an estimated routing space is added, the routes for the interconnection of the signal-nets are determined in later stages. A crucial factor is the amount of routing space to be added in each part of the layout. Faulty estimations may lead to a non routable placement, which enforces a rearrangement of the cells. In the facility layout problem first the centres of the facilities are determined with regard to the objective function while the shapes of the facilities are approximated by circles. The rectangular shapes for the facilities are fixed at a later stage. In this work hybrid approaches are proposed that are able to deal with the different aspects of these optimization problems in one step.

The method used in this work to solve the constrained packing or placement problems are genetic algorithms. Genetic algorithms have adopted concepts from the natural evolution process: they deal with a set of individuals, that represent encodings of candidate solutions to the optimization problem. During the optimization process new individuals are created out of the better individuals of the current population by recombination and mutation occurring in fixed generation cycles. As in natural evolution, the characteristical attributes of the individuals are encoded and can be characterized as their genetic information. A genetic algorithm searches in the space of these solution encodings rather than in the native search space of the optimization problem. During the creation of new individuals, attributes are transmitted from the 'parents' to these offspring individuals without considering the underlying optimization problem. The only link to the optimization task is the computation of the fitness of the individuals, which is needed for a ranking of the candidate solutions represented by the population.

In the classical applications of genetic algorithms, a set of continuous parameters has to be optimized. There the representation usually is a string of genes (bits or floats), which defines the values for those parameters. The genetic algorithm traverses the space of the representations by creating new individuals out of others when crossing strings during recombination or mutating single genes randomly. The individuals created by those operations are correct codings of admissible solutions and build the population for the next generation.

When solving combinatorial optimization problems with genetic algorithms, a string-type coding of solutions might be possible, but not every string represents a feasible solution. When the gene-strings of two individuals are crossed during recombination, it is unlikely that the resulting offspring represent correct solution encodings. There are generally two possibilities to deal with this problem. In the first, incorrect individuals might be accepted and inserted into the population. When computing their fitness, a penalty is added to ensure that these individuals get a lower chance in the selection process. This yields convergence to a population consisting of only correct individuals during the optimization.

Another way to get rid of the incorrect individuals is to prevent their formation by including problem-specific knowledge in the operators, such that they always produce correct offspring. This approach is used in this work. No simple genetic algorithms are used, but hybrid approaches that incorporate heuristics to make the genetic search process more efficient.

This work introduces a novel representation scheme: the genetic information of the candidate solutions is encoded in binary trees, that define the arrangement of the items to be placed. During the optimization process, the genetic operators directly work on these trees. For construction of the initial individuals and during creation of offspring, several heuristics are used for 'implicit' hillclimbing. Hillclimbing has been proven to be very useful in genetic algorithms for combinatorial problems. For each offspring, neighboured encodings are inspected by trying small mutations and accepting the mutated individual, if an increase in fitness has been achieved, otherwise another mutation is checked. In the hybrid approaches to the packing and placement problems to be introduced in this thesis each single individual encodes a set of neighboured solutions, e. g. the global arrangement of the blocks to be placed is fixed, but all feasible orientations for them are stored in one individual. Out of these the best solution can be easily identified. The resulting solution quality corresponds to taking a given placement and using a hillclimber that successively rotates all blocks. Using the implicit hillclimbing approach introduced in this work is much more efficient, since non-useful rotations can be identified very early during the construction of an individual, thus only a tractable number of configurations has to be stored.

The contribution of this work is the combination of hybrid techniques and its inclusion into genetic algorithms for solving constrained packing and placement problems. The implementations of the hybrid genetic algorithms cover many innovative features including self adaptation of the search, multi-parent recombination, and a novel encoding scheme that enables the implicit hillclimbing. The general applicability of the concepts introduced in this work are shown in several benchmark problems, the obtained results are compared with others reported in literature.

The work starts with a general description of genetic algorithms for optimization. In the third chapter three constrained packing and placement applications are described, and an overview of common solution techniques is given. In the next chapter some concepts and heuristics are introduced, which are used in the hybrid genetic algorithms that are described in the fifth chapter. In chapter six experimental results for the implementations of the algorithms on a parallel computer are presented, and the results are compared to those of other approaches. The work ends with a conclusion.

# Chapter 2

# Genetic Algorithms

The optimization technique used throughout this work are genetic algorithms. In this chapter an overview of their general form is given. After a description of the natural evolution process on which genetic algorithms are based, a short overview of the development of evolutionary computation is presented. Then a simple genetic algorithm, based on simple coding and standard operators is described, and some important extensions are outlined. The main focus is on a recent concept, the self adaptation of the search. Several approaches to strategy adaptation in genetic algorithms are reviewed at the end of this chapter.

## 2.1 Natural Evolution

Genetic algorithms are based on natural evolution, which is responsible for the appearance of all life on our planet. It is a very slow process running for billions of years and is even occurring at this very moment. It is an open process of adapting all lifeforms to be able to exist in a steady changing environment. This adaptation is done by slightly changing species over several generations. Sometimes species even die out, because of their lack of adaptability. In 1859 Charles Darwin [Darw59] called this process the *natural selection*, it is mostly referred to as the principle of *'survival of the fittest'*. In that sense, the *fitness* of an individual is defined by its adaptation to the environment, so this is rather a relative measure for the quality of an individual than an absolute quality. For example, a polar bear has a high fitness in its native environment, since it is very well adapted to the cold temperatures due to its thick coat. Bringing this polar bear to the African savanna would certainly reduce its fitness.

In this macroscopic view of the evolution process, life exists in fixed generation cycles. Each individual grows up, usually reproduces itself by creating one or more offspring, and eventually dies. The offspring inherit attributes of their parents. These attributes define the appearance of the individual, its *phenotype*. The attributes are encoded and transmitted at the microscopic level of the natural evolution, the corresponding actions are biochemical processes. The genetic information is encoded in a large set of *genes*. These genes are the transfer units of heredity. Each gene can take a particular value, the *allele*. This value of a single gene is encoded by a sequence of four different *nucleotide bases*. The genes are comprised in one or more *chromosomes*, which are strings of *DNA (deoxyribonucleic acid)*[1]. A set of chromosomes in an organism that defines its complete genetic information is called the *genome*.

## 2.2 Origin of Evolutionary Computation

About 40 years ago several scientists started studying the possibility of simulating the natural evolution process and using it as an optimization tool. Apart from some very early experiments (cf. [Mitc96], p. 2), which have been given little or even no attention, there arose two mainstreams on both sides of the ocean. While there was a parallel evolution of both approaches with little contact during the early years, this has changed and recently concepts of both techniques can be found in any of them.

In Germany, Rechenberg and Schwefel introduced the *evolution strategies (Evolutionsstrategie, ES)*, which were originally used for continuous parameter optimization. The fundamental experiments, that took place at the Technical University of Berlin in 1964, included mutation and selection on a two membered population, one parent individual and one descent. In the current notion of the evolution strategies this is called a (1+1)–ES [BäHoSc91]. Later this was extented to the general $(\mu \overset{+}{,} \lambda)$–ES, with $\mu$ describing the number of parents, $\lambda$ describing the number of offspring, and '+' or ',' defining the selection modus. In a $(\mu, \lambda)$–ES the population for the next generation is composed of the $\lambda$ offspring, which means that no individual 'lives' longer than one generation, whereas in the $(\mu + \lambda)$–ES also parent individuals may be selected for the next generation. The current evolution strategies include multi-parent recombination schemes and self adaptation by incorporating the strategy parameters into the coding of the individuals [BäcSch93]. References to the evolution strategies are the books by the founders of this technique, which both recently appeared in new editions [Rech94, Schw95].

The *genetic algorithms* were invented by Holland at the University of Michigan in the 60s, and further extended in the 70s. In 1975 the first edition of his book entitled *'Adaptation in Natural and Artificial Systems'* [Holl92] was published. This work does not directly deal with optimization, it gives a formal description of what an adaptation process is, how adaptation takes place in complex systems, and how this can be simulated in computer systems. As opposed to the evolution strategies, Holland's model includes a population of chromosomes, which are usually bit-strings, and crossover, inversion, and mutation are used during the adaptation process. As a main concept he introduced the notion of *schemas* (similarity templates), that define hyperplanes in the representation space. During one generation, due to the *implicit parallelism*, a high number of schemas is processed, which is formalized in Holland's — today controversial — *schema theorem*. In the 70s genetic algorithms were almost exclusively studied by Holland and his group at the University of Michigan. Later the work was spread around the states by his students. Genetic algo-

[1] This is a very abstract and simple description of the microscopic view of the natural evolution process, a more detailed description of the biological background can be found in Bäck's book [Bäck96].

rithms have been made accessible to a larger audience in 1989 by the appearance of the first textbook entitled *'Genetic Algorithms in Search, Optimization & Machine Learning'* [Gold89] by Goldberg, a former student of Holland.

In recent years, many research groups have worked on all kinds of evolutionary computation, which is proven by the large number of conferences and journals on this subject. Today there exist some very famous research labs working on evolutionary computation. In the USA there are for example the group of Holland at the University of Michigan, the IlliGAL group of Goldberg in Illinois [IlliGAL], the GARAGe of Goodman and Punch at Michigan State [GARAGe], De Jong's GAG at George Mason University [GAG], and the group of Forrest in New Mexico [UNM]. A lot of work is done at the 'Navy Center for Applied Research in Artificial Intelligence' [NCARAI]. The Santa Fe Institute [SantaFe], dedicated to the study of complex adaptive systems is a private, independent, multidisciplinary research and education center, founded in 1984. In Germany there are — among others — the research group 'Adaptive Systems' of Mühlenbein at the GMD [AS], the group of Schwefel in Dortmund [SYS], and the group of Rechenberg at the Technical University of Berlin [BIONIK]. The research in the U. K. is mainly on engineering applications, for example Flemings group in the ACSED in Sheffield [ACSED], the Evolutionary Computing Group at the University of the West of England in Bristol [UWE], the COGS in Brighton [COGS], and the Evolutionary Computation Group at the University of Edinburgh [Edinb].

## 2.3 A Simple Genetic Algorithm

As mentioned before, a genetic algorithm deals with a set of candidate solutions to an optimization problem. These solutions are called *individuals*, they are combined in a *population*. As in natural evolution, a distinction is made between the appearance of an individual, the *phenotype*, and the encoding of the genetic information needed to characterize this individual, which is the *genotype*. As in nature, this genotype can consist of different *chromosomes*, that define different properties of the encoded individual. In the simplest form of a genetic algorithm this genotype is a single chromosome which is represented by a string of elementary data-types, preferably a binary string. Each position *(locus)* in this string is a *gene* that can have different possible settings, the *alleles*. These can take the values '0' and '1' in a simple genetic algorithm. Each individual is valued by a *fitness function*. This may be a relative quality criterion depending on the quality of the other individuals in the population. For optimization problems, this can be the value of the objective function for the solution encoded by this individual. Figure 2.1 shows the pseudo-code for a simple genetic algorithm like that introduced in [Mitc96].

The main operator in genetic algorithms is the recombination operator. Two 'parent' individuals are chosen due to a well defined selection method from the current population to produce two *offspring*. If the chromosomes are just strings, the recombination operator produces these offspring by choosing one or more crossing points in the chromosomes and mixing parts of both parents to generate each offspring chromosome. In this case the operator is usually called the *crossover* operator. As in natural recombination, an offspring

1. Generate randomly a population of $n$ chromosomes
2. Compute the fitness $f(x)$ for each chromosome $x$
3. Select pairs of chromosomes
4. For each mating pair do
   (a) Cross the chromosomes of both parents with probability $p_c$ to generate two offspring
   (b) Mutate each gene in an offspring chromosome with probability $p_m$
5. Replace the individuals of the current population by the offspring
6. If termination criterion is not fulfilled, go to step 2.

**Figure 2.1:** The simple genetic algorithm

inherits genes of both parents. Crossing is done with regard to the crossover rate $p_c$, for a crossover rate less than one, offspring may be direct copies of parents. Each gene in an offspring is then mutated according to the mutation rate $p_m$, which is usually quite small ($p_m = 0.001$ and $p_c = 0.6$ were indicated as useful settings in an early study by De Jong [DeJo75]). Mutation is done by toggling a gene. At the end of each *generation* (steps 2. to 6. in figure 2.1), the individuals of the current population are replaced by the offspring.

## 2.4 Extensions of the Simple Genetic Algorithm

Although this was not intended by Holland's original work, genetic algorithms have been accepted as a generally applicable optimization tool, especially in the engineering domain. But solving real-life optimization problems is much more complicated than only finding a feasible binary encoding for the solution candidates, and using the simple genetic algorithm. In 1995 Wolpert and Macready [WolMac95] introduced the phrase of the *'No free lunch theorem'*, which states that no generally applicable and really robust search methods exist. Thus, genetic algorithms are not the final optimization technique for all kinds of problems. However, even if genetic algorithms do not have magic power, they are a highly efficient optimization method that can deal with very difficult optimization problems, on which almost all other techniques fail. There are a lot of extensions and new trends in genetic algorithms research, a short overview of those concepts relevant to this work is given in the following paragraphs.

Regarding the genotype representation, many different coding techniques can be found in the literature. Although most work is still on binary or real encoding, it turns out that these simple coding schemes are not always suitable. In contrast to the early applications of genetic algorithms, today it is more common to design a feasible coding for the given optimization problem, rather than mapping the problem to a simple, well-known coding. Davis [Davi91] advocates the use of the most natural encoding for the problem to solve, and the adaptation of the genetic algorithm to this representation. He further argues to hybridize genetic algorithms, i. e. to incorporate as much problem-specific knowledge as possible when solving real-world problems.

The simple genetic algorithm as presented in figure 2.1 is a generational algorithm, i. e. all individuals in the current population are replaced by the offspring at the end of each generation. However, some offspring might be identical to their parents, since crossing and mutating is done with respect to the crossover- and mutation-rates. In a *steady-state* genetic algorithm, only a few individuals are replaced. Often an *elitism* replacement scheme is used, i. e. the best individuals are retained, while the offspring (or only the better offspring) replace less fit individuals. Elitism was first introduced and analyzed by De Jong [DeJo75].

There are many different selection operators that can be used to choose which individuals of the current population are chosen to produce offspring. The selection mechanism originally proposed by Holland [Holl92] is designed with respect to natural evolution. Individuals with a higher fitness get a higher chance to be selected for reproduction. The probability of an individual being selected is proportional to its fitness value divided by the sum of fitnesses of all individuals in the population. This selection technique is named *roulette wheel selection*, since it can be interpreted as spinning a roulette wheel with slices for each individual, which are proportional to its fitness. In *tournament selection* [GolDeb91] two or more individuals are randomly chosen of the whole population and one of these is taken as a parent stochastically by giving the fitter individual a higher chance to be selected. Tournament selection is much more efficient to implement, since only the selected individuals are ranked. This can be especially beneficial in a distributed genetic algorithm. *Truncation selection* [MühSch94] is used in stock breeding and is an elitist selection scheme. Only the $t$ % best individuals are taken as parents to generate offspring.

As shown in figure 2.1, in the simple genetic algorithm the chromosomes of two parents are crossed to produce the chromosomes of two offspring. This mirrors sexual recombination as it takes place in nature. However, for the use of the evolution process as an optimization technique, other recombination schemes have been proposed. Mühlenbein [Mühl89] uses a majority vote which he calls 'p-sexual voting recombination operator'. Here the value of a gene is transferred to the offspring chromosome, if it occurs in a fixed number of the p parent chromosomes, otherwise a random value is taken. Bersini and Seront [BerSer92] present a 'Simplex-Crossover' operator that considers three (binary coded) chromosomes during recombination. Eiben *et al.* [EiRaRu94] use different multi-parent recombination schemes for function optimization. They describe experiments with up to ten parent individuals. Although in most cases the use of more than two parents increases the performance of the algorithm, they are not able to give a guideline for the ideal number of parents and the mode of offspring creation. In the field of evolution strategies also multi-parent recombination, called *global recombination*, has recently been introduced [BäcSch93, Schw95]. Mühlenbein and Voigt [MühVoi95] introduce the notion of *gene-pool* recombination. Here a pool of the selected parent individuals is installed. For the computation of each offspring allele, two chromosomes of this pool are randomly determined and out of these the offspring allele is chosen like in two parent recombination.

Because evolution is a highly parallel process and since parallelization of a standard genetic algorithm can be done quite straightforward, many parallel implementations of genetic algorithms have been developed. In the simple genetic algorithm, a *panmictic* population structure exists. This means that any individuals of the current population can be combined to create offspring during recombination. Natural evolution is highly parallel with mating of only locally 'neighboured' individuals. This can be taken into account by the use of specific population structures for the parallel implementation [Mühl91, Gorg92]. Generally, it can be distinguished between *migration models*, and *diffusion models*. In migration models the population is divided into a set of subpopulations *(demes)*. These subpopulations are located on *islands* and are optimized in parallel. Individuals periodically migrate to randomly chosen islands. If these islands are organized in a fixed topology and individuals migrate only to neighboured islands, this population model is called the *stepping stone model*. In the *diffusion model*, the population is distributed uniformly over a large territory. Here no fixed islands are defined, but mating only takes place in smaller subregions. This is likewise denoted the *isolating by distance model*. It is not clear, which of these population models is the best, but using a non-panmictic population in a genetic algorithm increases diversification, thus enhances search efficiency and decreases the risk of premature convergence.

## 2.5 Strategy Adaptation

The performance of each genetic algorithm depends on a set of control parameters like population size, crossover and mutation rates, and on the selection strategy. While there was some work done in determining a globally optimal, robust set of control parameters for a wide range of applications [DeJo75, Gref86], recently many approaches to self-adapting genetic algorithms have been published. Here no static parameter set is determined, but an adaptation of parameters during the optimization process takes place.

Fogarty [Foga89] was one of the first people to do work in this area. He describes different modes for the variation of the mutation rate during the optimization of an engineering application. In his test cases the benefit of a variable mutation rate mainly depends on the composition of the initial population.

Esbensen and Mazumder [EsbMaz94] describe an advanced implementation of Esbensen's genetic algorithm for macro-cell placement [Esbe92]. Their algorithm, *SAGA*, represents a mixture between a genetic algorithm and a simulated annealing algorithm. Depending on the progress of the search, the algorithm gradually switches to a simulated annealing algorithm by increasing the mutation rate and reducing the population size down to only one individual.

Smith and Fogarty [SmiFog96a] include a self-adaptation of the mutation rate by encoding it in each individual, like Bäck has already done before [Bäck91]. They use a steady-state genetic algorithm and investigate several replacement and selection strategies. In their approach a 'clutch' consisting of a fixed number of mutated clones of an offspring generated by recombination is created. Out of these, one individual is chosen and inserted into the population. They show the different behaviour on a set of test problems and compare it to standard genetic algorithms. Their experiments confirm that convergence velocity and quality of standard genetic algorithms can be considerably enhanced when adapting the mutation rate. More recently, they have expanded their approach by the use of a recombination operator that adapts the size of transmitted blocks from parents to offspring [SmiFog96b].

Julstrom [Juls95] describes an approach for adapting the probabilities for crossover and mutation operators during the search. He uses a steady-state genetic algorithm and the offspring are either created by mutation or crossover. Each operator is assigned a credit depending on the contribution it has recently made by creating offspring better than the population median. For that purpose a tree is stored for each individual describing the operators that created its ancestors. He presents results for the optimization of a function and for a small instance of the *traveling salesman problem (TSP)*. In both cases crossover turned out to be the more important operator, and in contradiction to the common opinion, its importance increases with search time.

Adaptation can be controlled by the evolution process itself by using a (meta-) genetic algorithm for the optimization of the parameters. The approach of Freisleben and Härtfelder [FreHär93] uses a two-level genetic algorithm. They adapt a set of 19 values, including real-valued parameters and so-called 'decisions'. One decision represents e. g. the particular selection strategy which can be chosen out of a set of four different strategies. In their parallel master/slave model, a set of secondary genetic algorithms (slaves), which deal with the target optimization problem is controlled by a primary genetic algorithm (master). They introduce a concept called 'indicators' to determine the influence of several decisions on the optimization process. The fitness of a parameter set in the primary genetic algorithm is characterized by the increase of the fitness gained by the secondary genetic algorithm used this particular parameter set. Bäck [Bäck94] also uses a two-level evolutionary algorithm to yield a maximum convergence velocity. As in the previous approach it is based on a master/slave model. After a fixed number of function evaluations, the slaves report their final best solution as a quality criterion to the master. He presents experimental results for a simple parameter optimization problem.

Wang *et al.* [WaGoPu96] propose a hierarchy of genetic algorithms and even suggest to adapt the genotype representation by using different representations in the subpopulations. They present experimental results for a two-level genetic algorithm run on several function optimization problems, similar to previously mentioned techniques using a meta genetic algorithm for adaptation [FreHär93, Bäck94]. In contrast to those approaches, the parallel slaves do interact during the search by migration of individuals between subpopulations (slaves).

Schlierkamp-Voosen and Mühlenbein [SchMüh94] introduce the concept of *competing subpopulations*. It is a self adaptation approach where the sizes of subpopulations are adapted, and these subpopulations pursue different strategies. Successful strategies are used by larger populations, while the size of less productive populations decreases. They optimize a number of test functions, and their strategies are distinguished by mutation range and recombination mode. Their approach is a parallel genetic algorithm implementation with migrating individuals between the subpopulations, but there is no meta genetic algorithm adapting the strategies used by the subpopulations, since these are fixed during the run. In [SchMüh96] this work is extended to adapt the overall population size during the search.

Hinterding *et al.* [HiMiPe96] describe a genetic algorithm for function optimization that uses two different levels of adaptation: they adapt the mutation strength by encoding it in the chromosome, as in the evolution strategies [BäHoSc91], further they adapt the population size. During the search they use three independent genetic algorithms with vastly different population sizes. These sizes are adapted either due to the best fitness or the best fitness improvement obtained over a fixed period.

Although some of the previously mentioned approaches adapt parameters of the genetic algorithms, especially mutation strength (step-size), at individual level, most adaptation schemes work at the level of subpopulations. Here the whole population is divided into several parts, which are optimized by more or less independent genetic algorithms that use different strategies. In the approaches with a centralized (master) control process, which is a genetic algorithm itself [FreHär93, Bäck94, WaGoPu96], the fixation of the parameters for this meta-algorithm entails a new problem. However, in contrast to the competition approach [SchMüh94, SchMüh96], the strategies of the parallel genetic algorithms are not fixed during the search.

# Chapter 3

# Constrained Packing and Placement Problems

There exists a wide range of combinatorial optimization problems dealing with packing, placing and cutting. For packing problems, a set of items, usually two or three-dimensional blocks, have to be packed inside a room with minimal area or space, respectively. The constrained packing problem described in this chapter is the two-dimensional bin-packing. A bin of fixed width is given, and a set of rectangular boxes has to be packed inside it in a way yielding a minimal packing height. For the placement problems considered in this work, a set of items has to be placed on a surface due to some specific constraints. In the VLSI layout generation task, rectangular modules of an electronic circuit have to be placed on the layout surface of a chip and connected by signal-nets, that are routed on the surface between these modules. To solve the facility layout problem, a set of flexible shaped facilities must be placed on a planar site in a mode to optimize the workflow between these facilities. In this chapter the three applications are described in detail, and an overview of known heuristic solution methods is given. The focal point is on approaches based on genetic algorithms.

## 3.1 VLSI Layout Generation

VLSI (Very Large Scale Integration) refers to a technology that enables the integration of more than a million transistors on a single chip. The design cycle for these VLSI-chips consists of different consecutive steps from high-level synthesis (functional design) to production (packaging) [Sher93]. All these steps contain a lot of non-numerical, compute bound problems. Conceivably the most complex task of all is the *physical design*. This process can be characterized as the transformation of a circuit description into a physical layout. The *layout* describes the geometric representation of all components of the circuit and the shapes of the interconnection wires. Out of this layout, the *masks* for the different layers needed for fabrication are generated.

Treating circuits with some hundreds of thousands of transistors has only become possible by the availability of Computer Aided Design (CAD) tools. Due to the growing complexity of the design process, there is a strong interest in tools for physical design automation with no or only little human interaction. Automation of the design process increases the level of integration and enhances the chip performance. Further it reduces the time to market, which is maybe the most important reason for the steady strong interest in new approaches to VLSI physical design automation.

### 3.1.1 Macro cell layouts

For the layout design of microprocessors, that represent the largest circuits, the *macro cell* design style is used. This is the most complex style. It is mainly applied for mass produced chips, since it yields highly optimized layouts. *Semi-custom* layout generation, used for standard-cells or gate-arrays, deals with more restrictions, which reduces the complexity of the design task, but also decreases the quality (level of integration) of the layouts [SaiYou95].

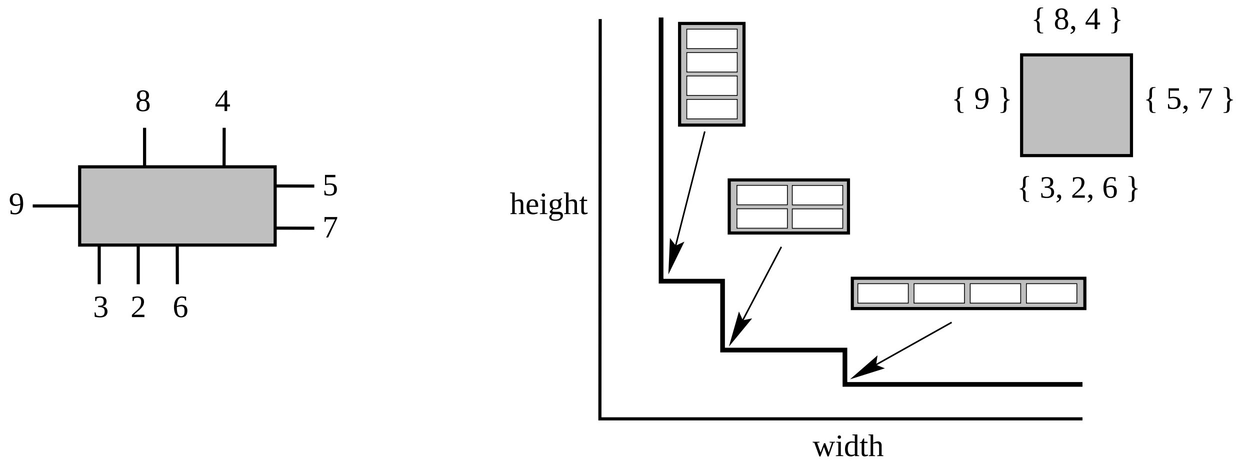


**Figure 3.1:** A fixed cell (left) and a flexible cell with three minimal implementations due to different arrangements of four sub-cells (right)

At the beginning of the physical design process the circuit is *partitioned* to generate some ten macro cells, that represent functional units. On the border of each cell, *terminals* (pins) are located for the connection of *signal-nets*. A *netlist* specifies the connectivity of the cells. From the designers point of view there exist two different kinds of macro cells: *fixed* cells, which have fixed dimensions and fixed terminal positions on their borders (cf. fig. 3.1, left), and *flexible* cells, which have different implementations with different aspect ratios, resulting from the hierarchical construction of the cells. The admissible implementations for flexible cells are defined either by the area and upper and lower bounds for width or height of the cell, or by a *shape-function* [Otte83], which describes the shapes that can be chosen for it. Figure 3.1, right, shows a flexible cell that consists of four sub-cells. The shape-function is characterized by three minimal shapes, but every point above the curve defines an admissible implementation for this cell. Since the terminal positions of a flexible cell vary with its different implementations, only a list of nets for each of its borders is given.

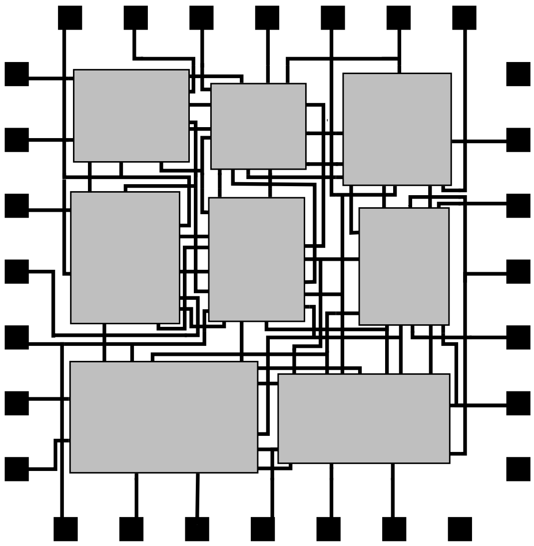

**Figure 3.2:** A schematic representation of a layout

During layout generation the cells have to be placed on the layout surface, and the signal-nets have to be routed on the space between the cells. On the border of the layout, *pads* have to be placed for the I/O connections of the chip (see figure 3.2). The main objective is the minimization of the area of the rectangle that circumscribes all components. Additionally there are timing constraints for some nets, which enforce maximal admissible wirelengths for these nets. Due to its complexity, the generation of macro cell layouts is usually done in four sub-steps [Sher93]:

1. In the *floorplanning* phase the cells have to be placed on the layout surface, and exact implementations for flexible cells have to be chosen.

2. After placement the *global routing* has to be done. In this step the 'loose' routes for the signal-nets are determined.

3. In the *detailed routing* the exact routes for the interconnection wires in each channel between two macro cells have to be computed.

4. The last step is the *compaction* of the layout, where it is compressed in all dimensions so that the total amount of area is reduced.

This classical approach to the layout generation is strongly serial with many interdependencies between the sub-steps. For example, while placing the cells, there must be an estimated amount of space provided around them to complete the routing later on. In case of faulty estimation, routing cannot be completed inside the reserved routing regions. This is either realized during the computation of the global routes or during channel routing. In the latter case the layout generation process has to backtrack and different global routes for some nets have to be chosen. If even global routing is impossible, the cells have to be rearranged, i. e. the process has to backtrack to the floorplanning task.

As already mentioned before, the division of the layout generation process results from the complexity of the whole optimization problem. Nevertheless, most of the sub-tasks are still intractable so that only heuristic approaches can be used.

---

**Given:**

set of $n$ cells $\mathcal{C} = \{c_i \,|\, i = 1, ..., n\}$

$c_i = (w_i, h_i, T_i)$ for a fixed cell $c_i$, $w_i$ and $h_i$ are width and height, resp.,

$T_i$ is list of terminals for the borders of the cell

$c_i = (S_i, T_i)$ for a flexible cell $c_i$, $T_i$ is set of terminals for the borders of the cell

$S_i = \{(w_j, h_j) \,|\, j = 1, ..., k\}$ is a shape-function,

$k$ is the number of implementations for cell $c_i$,

$w_j$ and $h_j$ are the width and height, resp., of the $j$-th layout-alternative

set of signal-nets $\mathcal{N} = \{n_i \,|\, i = 1, ..., m\}$,

$n_i = \{t_1, ..., t_s\}, t_l \in T_j$, i. e. $t_l$ is terminal of cell $c_j$

**Goal:**

placement with minimal area, i. e. with minimal area of the circumscribing rectangle,

positions and shapes for the cells and routing area between them to fulfill

routing of all signal-nets

---

**Figure 3.3:** The placement problem in VLSI physical design of macro cell layouts

### 3.1.2 Placement

During *placement* the macro cells have to be positioned on the layout surface in such a manner that no cells overlap each other and that there is enough space left between them to complete the interconnections specified by the signal-nets later on. The *floorplanning* is a generalization of the placement problem. It deals with flexible cells, which have to be sized during (or after) placement to provide a layout with an overall minimal area. A *floorplan* is a partitioning of the layout surface into different rooms, inside of which the flexible cells are placed (cf. figure 3.4).

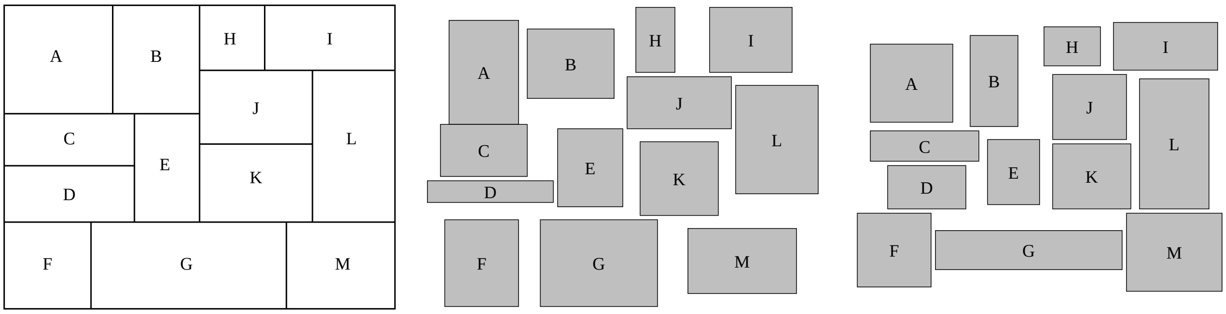


**Figure 3.4:** A floorplan (left) and two corresponding placements with different shapes for the particular cells

Several approaches to the placement or floorplanning problem exist. All these approaches have to estimate the costs for the routing of the signal-nets. This is done by considering an approximation of the overall wirelength when rating a given placement, and by adding an estimated amount of routing space around the cells. Generally, it can be distinguished between *constructive* and *iterative improvement* methods [ShaMaz91].

In *force-directed placement* cells that are connected by signal-nets exert an attractive force on each other, which is proportional to the number of these nets and the distance between the cells. The ideal positions for the cells can be computed numerically by solving a set of equations, which corresponds to finding a equilibrium in a minimum energy state [AnJoKi82]. Although the original approach was constructive, iterative methods based on force models exist [ShaMaz91]. Usually, the objective function in force-directed placement is based on the estimated total wirelength rather than on the layout area, thus the packing aspect of the placement problem is less inspected.

*Partition based methods* are constructive and compute a placement by recursively dividing the set of cells. At the same time, the available chip area is partitioned, and each set of cells is assigned to one of the components. Mostly a min-cut heuristic is used, thus the number of nets cut in each division is minimized, which yields highly connected cells to be placed near together. There are different modes to choose the sequence of the cut lines (e. g. cut oriented, quadrature, and bisection placement), that provide different cutting patterns with regard to the arrangement of rooms on a floorplan [Breu77]. The main problem of the basic approach is that the locations of the external connections are not considered during placement of a sub-circuit. To deal with this, Dunlop and Kernighan [DunKer85] introduced a terminal propagation technique based on inserting dummy terminals on the cut lines.

A very popular technique to compute the placement is *simulated annealing* [Sech85, WoLeLi88]. It is an iterative improvement method, which simulates the behaviour of atoms during the cooling schedule of molten metals. It starts with a randomly created arrangement of the cells and successively computes new configurations by moving or exchanging cells. A new solution is accepted depending on its quality and on the 'temperature'. During the early stage of optimization, when this value is rather high, even inferior new solutions are accepted. This assists the optimization process in avoiding to get stuck in local optima. Simulated annealing yields high quality placements, but needs an excessive amount of computation time.

The classical work that uses genetic algorithms for floorplanning has been done by Cohoon *et al.* [CohPar86, CoHeMaRi91]. They present a parallel genetic algorithm for floorplan design and use a weighted sum of the total area and the estimated wirelength as the objective function. A placement is described by a binary tree with the leaves representing the cells and the inner nodes defining the arrangement of the partial placement characterized by the corresponding subtrees. The genotype is encoded as a normalized polish expression resulting from a post-order traversal of this tree. They have implemented different recombination operators, that work either on this string without considering the structure of the tree, or directly transmit subtrees from parents to offspring. Results are presented for artificial circuits with up to 25 cells.

Chan *et al.* [ChMaSh91] introduce a bit-matrix representation. The layout area is divided into a number of quadratic regions. The placement for a single cell is represented in a binary encoding of the information about the occupied squares and the orientation of this cell. The total placement information is combined in a bit-matrix, each line in this matrix describes the placement for a single cell. Cells are allowed to overlap each other during the optimization, which is considered by adding a penalty to the fitness of an incorrect individual. Routing is included by estimating the wirelength. Recombination is done by constructing an offspring out of quarterized parent matrices. The authors present results for benchmark circuits with up to 49 cells.

Esbensen [Esbe92] describes a genetic algorithm for macro cell placement where the genotype is encoded as a binary tree. Each node of the tree represents a cell, and due to a given node order, a placement can be gradually generated by decoding the genotype. The quality of a placement is determined by the layout area, for layouts with equal area an estimated wirelength is taken into consideration. Later Esbensen and Mazumder [EsbMaz94] combined this algorithm with simulated annealing, which produces better results than the pure genetic algorithm. Due to complexity constraints, only (real) circuits with up to 11 cells and 203 nets are computed.

### 3.1.3 Routing

The aim of the routing phase is to find the geometrical layouts for all nets. In the floorplanning phase space on the layout surface has been provided for the routing of the signal-nets. This space can be described as a collection of *routing regions*. Each region has a fixed *capacity*, i. e. a maximum number of wires that can be routed through this region, and a number of terminals with fixed positions on the borders of the adjacent cells.

Due to complexity, routing is done in two sub-phases. In the *global routing* each net is assigned to particular routing regions. A common way to compute this allocation is based on graph algorithms. Here the total routing space is described by a graph: the

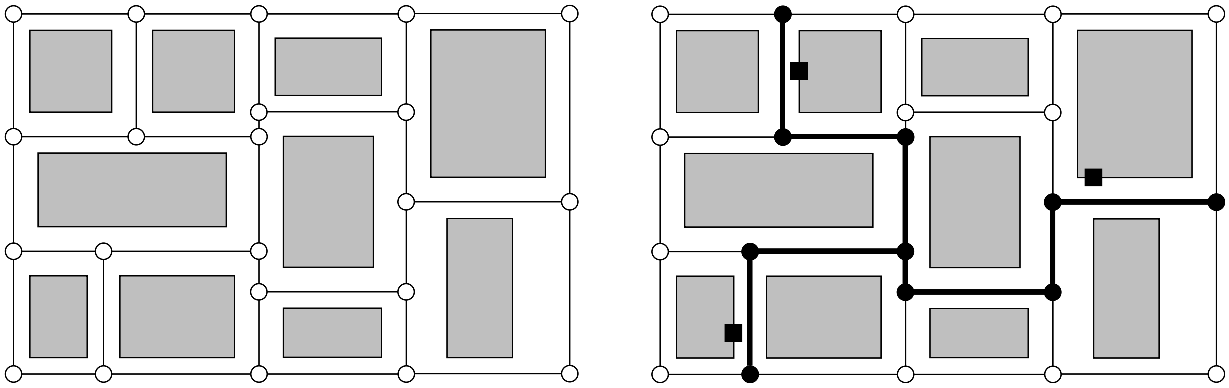

**Figure 3.5:** A routing graph (left) and a global route for a three terminal net (right)

edges of this graph represent the routing regions and are weighted with the corresponding capacities; the vertices are the connections between two regions. The global route for a net is described by a list of routing regions (cf. fig. 3.5). The main objective is to find the global routes for all nets of the circuit with a minimal overall wirelength, and none of the capacities of any routing region being exceeded. For two terminal nets shortest paths algorithms are used, while for nets with three or more terminals a minimum rectangular *steiner tree* is computed. The described technique is usually cited as the sequential approach to global routing. Other approaches are based on integer-programming, hierarchical decomposition, or random search techniques like simulated annealing or genetic algorithms [Leng90, SaiYou95].

After global routing is done, for each routing region *(channel)* all nets routed through it are known. In the *detailed routing phase*, the exact shapes for the wires have to be determined (cf. fig. 3.6). This is done incrementally, i. e. one channel is routed at a time in a predefined order.

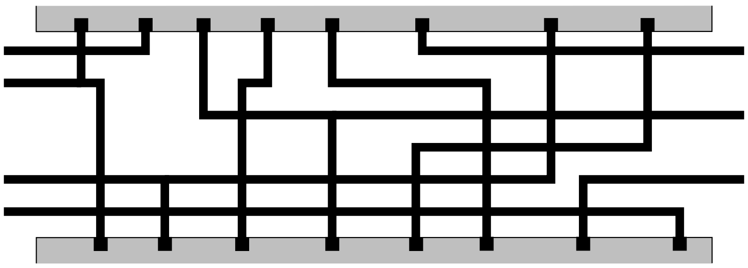

**Figure 3.6:** The detailed routing inside a channel

The complexity and routability of a layout depend on the number of layers which can be used for the completion of the interconnections. In macro cell layouts there are usually two layers and routing is done in the manhattan-model, i. e. there are only horizontal and vertical line segments. The simplest model is the restricted manhattan-routing, where one layer is used for vertical, the other for horizontal wires and the nets change the layer when changing their direction. For detailed routing solutions exist based on greedy methods, graph algorithms or hierarchical approaches [Sher93, SaiYou95].

---

**Given:**

set of $n$ rectangular blocks $\mathcal{B} = \{b_i \mid i = 1, ..., n\}$

$b_i = (w_i, h_i)$ with $w_i$ and $h_i$ are width and height, resp., of block $b_i$

two-dimensional bin with fixed width and unlimited height

**Goal:**

packing pattern of all blocks inside the bin with minimal packing height

---

**Figure 3.7:** The two-dimensional bin-packing problem

## 3.2 Bin-Packing

In this section the simplest of the problems dealt with in this work is described. This is the *two-dimensional bin-packing problem* (figure 3.7). A set of rectangular blocks is given, that have to be packed inside a bin with fixed width and unlimited height. The objective is to find a packing pattern containing all blocks inside the bin with a minimal height. Figure 3.8 shows an example with 11 blocks.

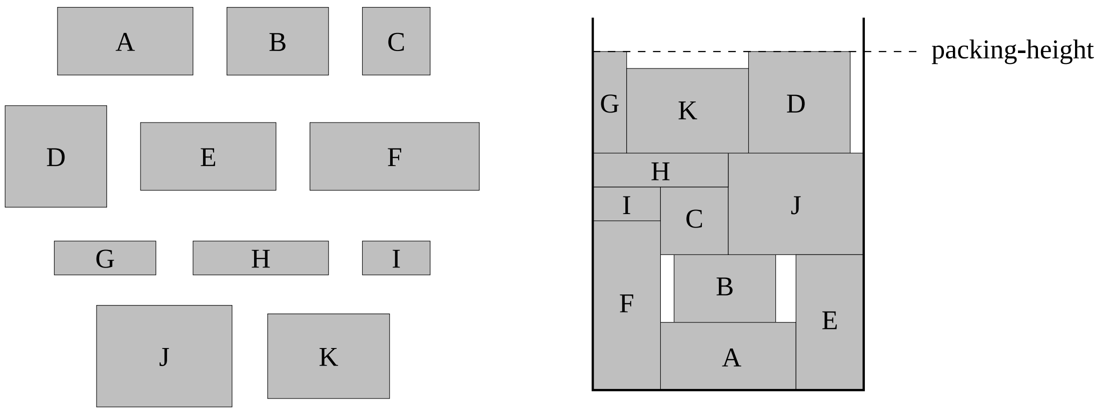


**Figure 3.8:** A solution for a bin-packing problem with 11 blocks

A simplification of this problem is the one-dimensional bin-packing problem, which differs from the two-dimensional version in the fact that all rectangular blocks have the same width, thus can be arranged in slices inside the bin. A different interpretation of this problem is having a set of bins of fixed size and stacking all blocks upon each other inside the bins. Even the one-dimensional bin-packing problem is NP-complete [GarJoh79]. Extensions are three dimensional packing problems, the applications are palette or container loading.

The two-dimensional bin-packing problem can also be considered as a special case of the *cutting stock problem*. In the general cutting stock problem, a set of demand rectangles is given, that have to be cut out of one or more larger stock rectangles. The demand rectangles are valued and there are constraints regarding the maximal number of any kind of these rectangles. The cutting stock problem has a lot of applications in the glass and lumber industry.

Wang [Wang83] proposed a heuristic for solving the cutting stock problem, which is based on successively composing demand rectangles or partial patterns to a larger rectangle characterizing a new partial solution. During this construction, only solutions with internal waste below a given threshold are accepted. The choice of this threshold defines the attainable solution quality. The major drawback of this approach is the increase of computation time, when a high solution quality is aspired. Vasko [Vask89] uses a different algorithm to compute an upper bound for the threshold before running Wang's original algorithm. Oliveira and Ferreira [OliFer90] improve Wang's algorithm by using a more exact estimate for the expected waste. For each partial pattern they estimate the minimal waste a solution will have, if it includes this pattern.

There exist branch & bound algorithms for packing and cutting problems, e. g. by Whitlock and Christofides [WhiChr77] or Kröger and Vornberger [KröVor90], but their applicability is limited to problem instances with up to 20 blocks. Morabito *et al.* [MoArAr92] and Daza *et al.* [DaAlDi95] describe approaches based on heuristic search algorithms from Artificial Intelligence, the *And/Or graphs*.

Kröger [Krög93, Krög95] uses a parallel genetic algorithm to solve the two-dimensional bin-packing problem. He uses a slicing-tree representation for a packing pattern, the genotype encoding is a string obtained by a preorder traversal of this tree. A comparison of the genetic algorithm with a set of other heuristics is made and several population structures are analyzed for the parallel implementation. He introduces the concept of meta-rectangles which are temporary fixed patterns that are treated as atomic units during the ongoing search. Test instances with sizes from 25 to 60 blocks are solved.

Similar to Kröger's concept of meta-rectangulars is the approach proposed by Goodman *et al.* [GoTeKu94] for compaction, packing or nesting problems. In their *hierarchical chromosome representation*, the elements to be placed or packed are composed in a hierarchical mode during the optimization. The genetic operators work on these groups of items rather than dealing with single items.

**Given:**

set of $n$ facilities $\mathcal{F} = \{f_i \,|\, i = 1, ..., n\}$

$f_i = (a_i, lr_i, ur_i)$

$a_i$ is area of facility $f_i$

$lr_i$, $ur_i$ are lower and upper bound, resp., for aspect ratio of facility $f_i$

flow matrix $M = ((m_{ij}))_{i,j=1,...,n}$ with $m_{ij}$ is the flow between facilities $f_i$ and $f_j$

**Goal:**

arrangement and shapes for the facilities with minimal total flow costs $\sum_{i,j=1}^{n} d_{ij} \cdot m_{ij}$

with $d_{ij}$ is distance between centres of gravity of facilities $f_i$ and $f_j$

shapes $S = \{(w_i, h_i) \,|\, i = 1, ..., n\}$

with $w_i$ and $h_i$ are width and height, resp., of facility $f_i$

and $w_i \cdot h_i = a_i, \quad lr_i \leq \frac{h_i}{w_i} \leq ur_i$

**Figure 3.9:** The simple facility layout problem with non–identical facilities

## 3.3 The Facility Layout Problem

The *facility layout problem* deals with a set of facilities that have to be placed on proper positions on a planar site. These facilities have fixed areas and usually ranges for their shapes are provided, like in the case of flexible cells in VLSI layout generation (see section 3.1.1, page 18). For each pair of facilities a *flow* is given. The objective is to minimize the overall *flow costs*, while satisfying the given constraints (see figure 3.9). The flow costs are the sum over all products of the flow with the particular distance between two facilities.

The facility layout problem is an extension of the well known *quadratic assignment problem* [PaReWo94]. There, a set of slots and a set of items, both of cardinality $n$, a distance matrix $D = ((d_{ij}))_{i,j=1,...,n}$, and a flow matrix $F = ((f_{ij}))_{i,j=1,...,n}$ are given. An entry $d_{ij}$ of $D$ describes the distance between slots $i$ and $j$, while an entry $f_{kl}$ of $F$ defines the flow between two items $k$ and $l$. The solution of the quadratic assignment problem is a bijective assignment $\pi$ of items to slots with overall minimal flow costs $\sum_{i,j=1}^{n} d_{ij} \cdot f_{\pi(i)\pi(j)}$. The quadratic assignment problem is NP-complete [GarJoh79].

Applications of the facility layout problem are manufacturing cells layout, architectural space planning, and offices layout. Also the floorplanning task during the VLSI layout generation can be seen as a facility layout problem with the cells representing the facilities to be placed, and the flow between two cells is characterized by the number of nets to be routed between these cells.

There exist several variants of this problem, which can be classified by the constraints to be considered [MelGau96]. In the simplest case, the *identical facilities layout problem*, all facilities have the same shape, in opposite to the *non–identical facility layout problem* which is the general case with different shaped facilities. In the latter case, the shapes can be either fixed, or ranges for the aspect ratios of the facilities are given together with their areas. *Dead-space ratio* might be included for some facilities, which defines the ratio of area that must kept empty inside the rectangular space a facility is placed on. This results in L-shaped rooms on the layout. Prespecified areas may also exist. These represent already occupied areas on the planar site on which facilities cannot be placed. That might be pillars or elevator shafts in the case of facility layout in a manufacturing hall, for example. Another constraint can be the shape or area of the site on which the facilities have to be placed. One can also think on an extension of the problem with more than one site given, e. g. when doing office layout for a multi-storey building.

While there are a lot of different approaches to the several variants of this problem, in the following some recent approaches to the simple non–identical facility layout problem are reviewed. This optimization problem can be formulated as shown in figure 3.9. There are two different ways to measure the distance between two facilities located at points $(x_1, y_1)$ and $(x_2, y_2)$: the *manhattan distance* $|x_1 - x_2| + |y_1 - y_2|$ and the *euclidian distance* $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. For the placement mode of a facility, it can be distinguished between *free* and *rigid* orientation. Facilities with free placement can be rotated before placing, thus for such a facility $(a_i, lr_i, ur_i)$, a less restricted range for the aspect ratio can be considered: the lower bound is $\min\{lr_i, \frac{1}{ur_i}\}$, and the upper bound is $\max\{ur_i, \frac{1}{lr_i}\}$.

Van Camp *et al.* [vCCaVa91] use a non linear programming method to solve the simple facility layout problem with non–identical facilities. In addition to the formulation in figure 3.9, the area of the site on which the facilities have to be placed, is given. In their approach an initial layout is generated by solving a relaxed model of the original problem. For that purpose, the facilities are considered as circles. The area of a circle is larger than the actual area of the corresponding facility. In the first stage, the centres of these circles are evenly spread on the layout site with regard to the flow between them. This is done without considering the boundaries of the circles, i. e. the circles are allowed to overlap each other. In the second stage, the exact positions of the circles are computed by 'moving' them iteratively on the layout surface. At this phase, overlapping is considered by a penalty term in the objective function. In the third stage, the facility layout problem is solved iteratively starting with the previously generated placement. The problem is formulated in a non linear model that uses a continuous representation for both the positions of the facilities and their shapes. For the optimization process they transform the constrained form of this problem into an unconstrained form. They present results for problem instances with up to 13 facilities.

A hierarchical approach to the facility layout problem is presented by Tam and Li [TamLi91]. First, the facilities are iteratively merged in clusters with regard to the maximal sum of flows between the clustered facilities. The maximal cluster size is limited, since the following phases are only practicable for small groups with less than 15 facilities. In the second phase an initial placement for the facilities in each cluster is computed. For that purpose, all facilities are assumed to be circles with an area slightly larger than the original size of the facilities. To compute a relative arrangement, a force field method is used. Similar to the force-directed placement in VLSI (see section 3.1.2, page 21), each pair of circular facilities is attracted by a force, the magnitude of which is proportional to the square of their distance. To avoid overlapping, this force gets repulsive, if the distance is below the sum of the radii, which corresponds to an overlap. The computation of the initial placement is formulated as an unconstrained optimization problem, which is solved numerically. In the last step the layout is refined. In this phase the circular facilities have to be transformed into rectangular blocks which fulfill the given constraints. The constraints are incorporated in a non-linear system of equations that are solved numerically. Although they state that the inclusion of occupied sites is possible in their approach, only results for instances with up to 30 facilities for the simple non-identical facility layout problem are presented. For the instances with 20 or more facilities, a clustering method is used, i. e. in the first phase clusters with up to 10 facilities are generated, which are independently placed. The layouts for these clusters are combined in a bottom-up merging process without any further refinement of the internal layouts.

Tam [Tam92a] describes a genetic algorithm for the facility layout problem that is based on a slicing-tree representation (slicing-trees will be described in detail in section 4.2). The leaves of the tree characterize the facilities, and the inner tree nodes define their arrangement. In comparison to the already mentioned approaches that use slicing-trees for VLSI-layout generation and bin-packing (sections 3.1.2 and 3.2, resp.), in this algorithm the structure of the tree is fixed. It is determined during an initial clustering process, where facilities are hierarchically paired based on average linkage. In each step those single facilities or clusters of facilities are paired that are linked by the highest flow. In the second phase, a genetic algorithm computes the optimal labeling for the inner nodes of the tree, which define the arrangement of the facilities. For that purpose the genotype representation is a string containing these labels. There exist four different labels for each node that define both the cut direction and the assignment of the sub-layouts to the resulting sites. Out of a given slicing-tree, a floorplan is gradually generated by dividing the site into smaller regions during a top-down traversal of the tree. In this approach also dead-space area can be considered. Layouts with up to 30 facilities are computed. Tam [Tam92b] also implemented a simulated annealing algorithm for the facility layout problem, which uses the same representation as the genetic approach. The same set of instances is solved, but there is no comparison made. From the point of layout quality, both approaches seem equivalent.

Kado *et al.* [KaRoCo95] have implemented several genetic algorithms for the facility layout problem, most of them are similar to the previously mentioned approaches. All their methods are based on the use of binary slicing-trees, either with fixed structure, as computed during a clustering process like Tam has done [Tam92a, Tam92b], or with encoding the tree structure by the use of a postfix notation like Cohoon *et al.* have reported in their approach to VLSI floorplanning [CohPar86, CoHeMaRi91]. Some of their programs also include other heuristics. They solve a set of 15 problems with up to 30 facilities, all of them taken from other publications. Although they present results for some of these problems that are better than the previously known best solutions, they are not able to determine the overall best approach. The quality gained by the particular methods mainly depends on the size of the problem. However, in their opinion the number of facilities is not the only factor characterizing the complexity of a particular problem instance.

The main difference of the facility layout problem in comparison to the bin-packing problem introduced in section 3.2 is the fact that the blocks in this case are fully flexible facilities. Only the area and some constraints for the aspect ratio are given for each item to be placed. This has led to multi-staged approaches as described above. There are two general methods: the first technique considers the facilities as circles during the computation of an initial placement, the other constructs a slicing-tree, out of which a layout is generated. In chapter 5 a genetic algorithm for the simple facility layout problem will be presented which is based on a slicing-tree representation. The slicing-tree is constructed in a bottom-up fashion and takes into account different shapes for the facilities during their composition.

# Chapter 4

# Advanced Concepts

In this chapter some non-genetic concepts are introduced, that will be used in the hybrid genetic algorithms to be described in chapter 5. The first two techniques, the *shape-functions* and the *slicing-trees*, are common techniques from VLSI physical design and have been already mentioned in section 3.1. The concept of shape-functions is generalized in this work to take into consideration the orientation of blocks when using shape-functions to describe a hierarchical composed pattern. The third methodology presented in this chapter is the iteration of a well-known graph-theoretical technique, the computation of a *weight matching* on a complete graph. This heuristic can be used to generate high-quality packing or placement patterns, and it ideally supports the characterization of this patterns by slicing-trees.

## 4.1 Shape-Functions

*Shape-functions* have been first defined by Otten in 1983 [Otte83] and named *shape constraints* there. They are used in VLSI-design to characterize different shapes for cells of a circuit. In its simplest form, a shape-function is a stepwise function that defines admissible shapes for a cell, i. e. the relation of area and aspect ratio.

If the cell has a fixed size and a fixed orientation, figure 4.1 (left) shows a feasible shape-function like it was originally introduced by Otten. Since this size in the context of VLSI-design only represents a minimal area that is occupied by the cell and can be extended in both dimensions by empty space, by definition all points in the shaded area are considered as admissible implementations for this cell. The right picture in figure 4.1 shows the shape-function for a cell of size (2,3) units with no fixed orientation. The possible orientations (upright and upset) conduct to two minimal layouts with (2,3) and (3,2) units for this particular cell.

For many building-blocks in VLSI physical design, especially macro cells, the shape is not fixed during the computation of a layout. Because these cells are hierarchically composed, the total area of the contained subcells is known, but not their detailed arrangement and the exact wiring inside. For these flexible cells, a shape-function is either piecewise

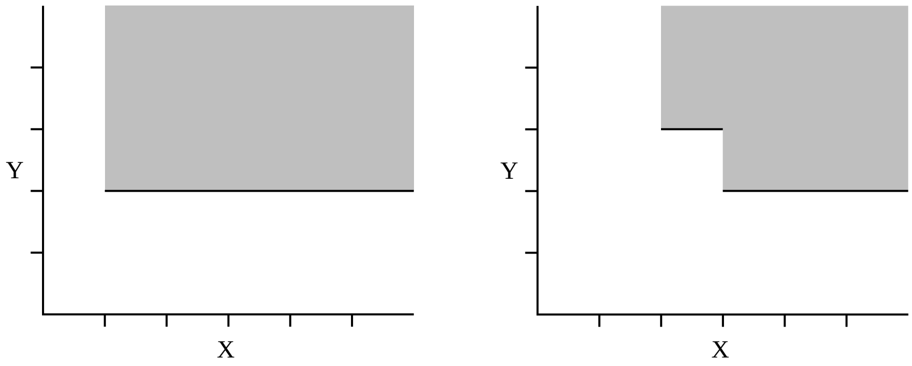


**Figure 4.1:** A simple shape-function for a fixed oriented cell with a minimal size of 1 unit in the X-dimension and 2 units in the Y-dimension (left), and a shape-function for a cell with free orientation, which yields minimal size of (2,3) or (3,2) units, respectively (right)

linear, as shown in the left part of figure 4.2, or given by the minimal size for each dimension and the overall area, like represented in figure 4.2 (right). For the discrete case, the shape-function is completely characterized by those coordinates which describe the minimal implementations. In the following we only consider this type of shape-functions.

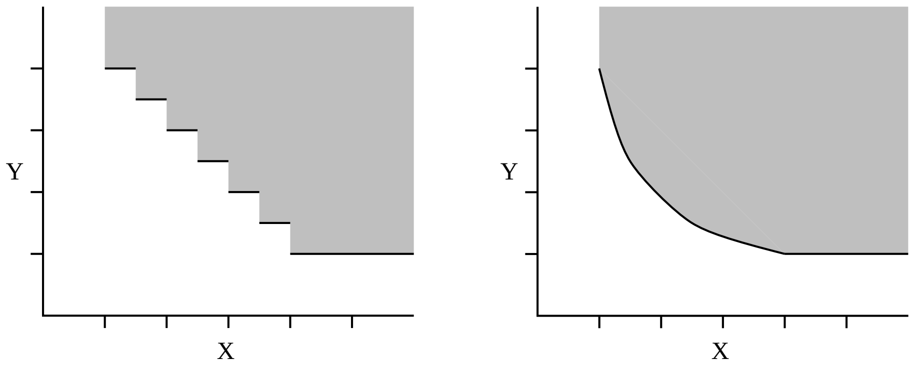


**Figure 4.2:** A simple shape-function for a flexible cell with seven discrete minimal shapes (left), and a shape-function as a continuous bounding function for a cell with minimal width and height of one unit and a given overall area (right)

When combining two flexible cells, the shape-function of the composed — of course again — flexible cell can be easily calculated in the discrete case. Figure 4.3 shows this construction for two cells with three and five different implementations. Fixing the composition mode (upon or side by side) yields a maximal number of 15 combinations for either

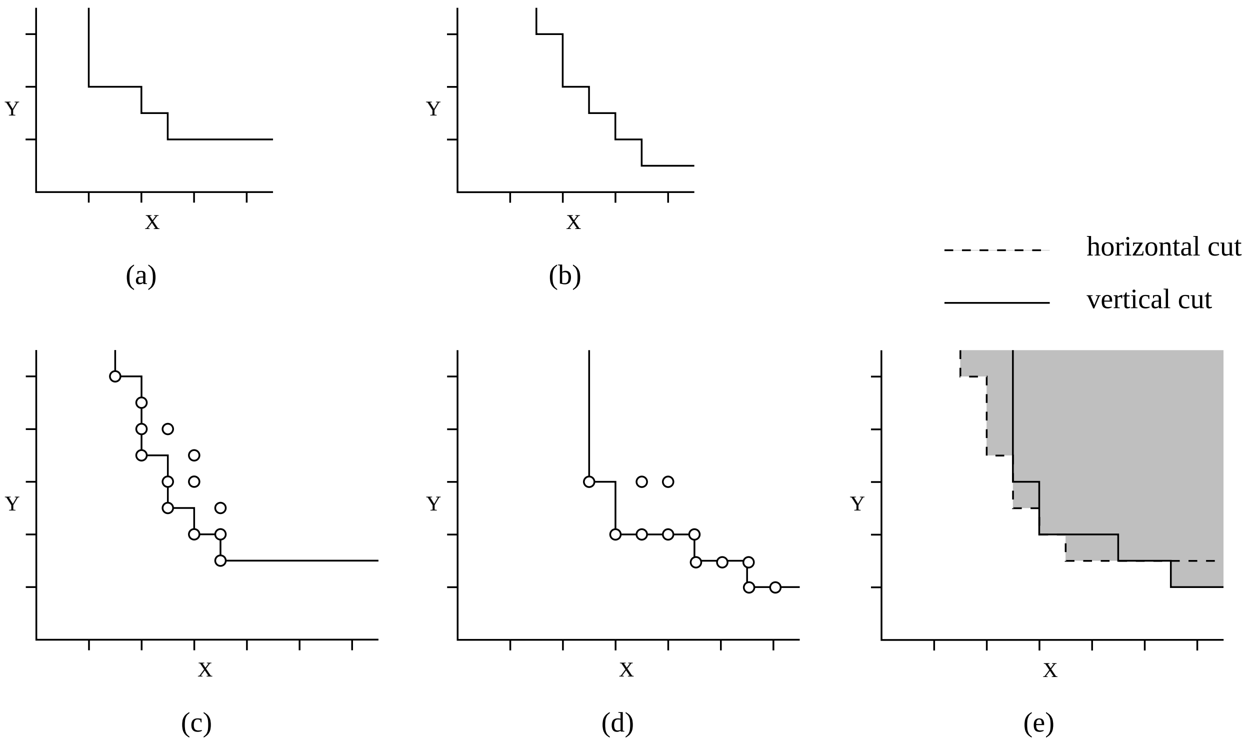


**Figure 4.3:** The shape-functions for two cells (a), (b), the different aspect ratios and the resulting shape-function for a cell consisting of these positioned upon each other (c), side by side (d), and the general shape-function (e)

case. As it can be gathered from figure 4.3, (c) and (d), there are 13 and 12 different aspect ratios for the composed cell. Since most implementations are dominated by others, the resulting shape-function is completely defined by 4 and 5 coordinates, respectively.

Zimmermann [Zimm88] extended the notion of a composed shape-function by considering the *cut direction*. The definition for the addition of shape-functions can also be made by the notion of cuts instead of the composition of two cells. Here, dividing a cell into two subcells is done by cutting the cells horizontally or vertically. Thus, in the following the notion *horizontal cut* is synonymously used for positioning two cells one upon another, and *vertical cut* for positioning the cells side by side, respectively. Both alternatives can be considered by calculating the shape-functions for both cases and overlaying them. As shown in figure 4.3 (e), the resulting general shape-function is defined by the coordinates for the minimal implementations and the cut direction they result from. Again, due to dominated coordinates, not all combinations need to be stored in the example presented in figure 4.3. But in contrast to the previously described redundancy when adding shape-functions, here worst case examples can be created without any dominated implementations at all (see [Müll90], p. 64).

In the context of packing, when two blocks are joined, the resulting pattern defines a new block. In the following a block containing a set of blocks, thus describing a packing pattern is called *meta-block*. While Zimmermann extended the notion shape-functions by incorporating the cut direction for each coordinate, to describe packing patterns, there is another possibility to generalize the definition of shape-functions: When two blocks are combined to a meta-block, apart from putting them side by side, or upon each other, there exist different orientations for the blocks. In packing problems, there are two different orientations for each block: upset and upright. Considering those for both blocks yields four different implementations for a pattern consisting of these two blocks after fixing the cut direction (figure 4.4). Note that there again exists a dominated implementation: combination (II) is covered by combination (I), since both have the same height, but combination (II) has a larger width than combination (I).

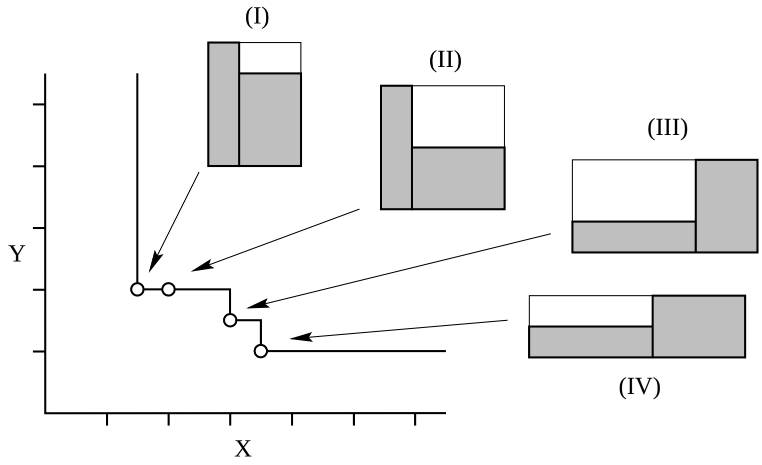


**Figure 4.4:** Considering upset and upright orientations during the combination of two blocks placed side by side

If a packing pattern or a placement is created by adding shape-functions at each level to get the shape of the resulting meta-block in the next level, storing the cut direction for each coordinate is unnecessary. Only considering one arrangement, e. g. placing both blocks upon each other, is enough: due to the consideration of all feasible orientations at each level, at the next level a side by side arrangement is taken into account by rotating this meta-block by 90 degrees.

When joining two cells to a partial layout in physical VLSI-design, due to the terminals on the borders of the cells, four orientations for each cell have to be considered. These represent the rotation of a cell by 0, 90, 180, and 270 degrees, respectively. While the 180 and 270 degree rotations of the cells do not provide different shapes at principle, when composing two cells, considering an estimation for the routing space inside the channel between the cells yields different shapes for these cases, too. Figure 4.5 shows different implementations for a partial layout resulting from positioning two cells side by

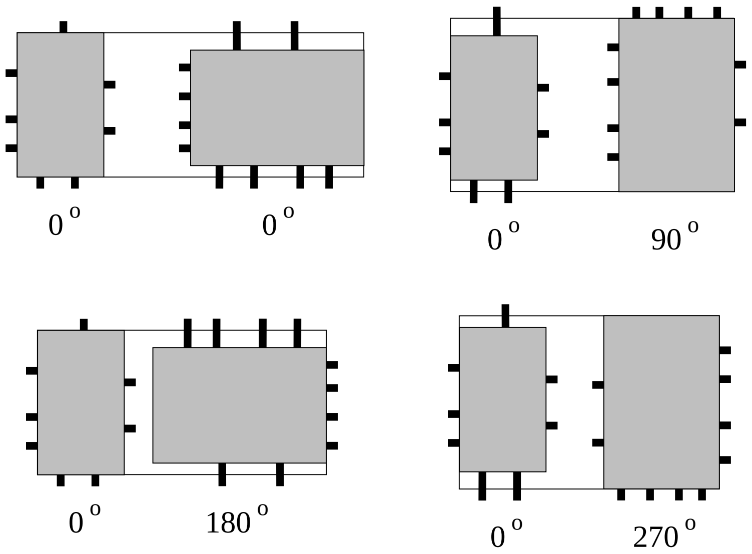


**Figure 4.5:** The four possible combinations resulting from only the right cell being rotated, the different widths are caused by the addition of routing space in the channel between the cells

side, and considering only the four possible rotations for the right cell. Depending on the number of nets inside the vertical channel between the cells, different amounts of routing space are added. This yields different widths even for those cases that provide equivalent implementations, if routing is not included.

## 4.2 Slicing-Trees

In physical VLSI-design, floorplanning is one of the main tasks (see section 3.1.2, page 20). Here, the cells have to be placed on a layout surface, and for flexible cells the dimensions have to be fixed to yield a layout with an overall minimal area. This optimization process can be done in two phases, first the cells have to be placed, and after that they have to be sized. For that purpose, the relative arrangement of the cells has to be described, which is done by dividing the layout area into rectangular rooms and assigning one of the cells to each room. The use of *slicing-trees* is a common way to describe such a floorplan. A slicing-tree is a hierarchical description of the arrangement of the rooms inside a floorplan and thus describes the relative placement of the modules (see figure 4.6).

The simplest form of these trees are the binary slicing-trees, that do always characterize slicing floorplans. Due to Stockmeyer [Stoc83], a floorplan is named *slicing*, if it is either a basic rectangle (room for a single cell) or there is a line segment (a *slice*), that divides the enclosing rectangle into two pieces such that each piece is slicing. Hence, a slicing floorplan can be recursively cut by so-called *guillotine cuts*.

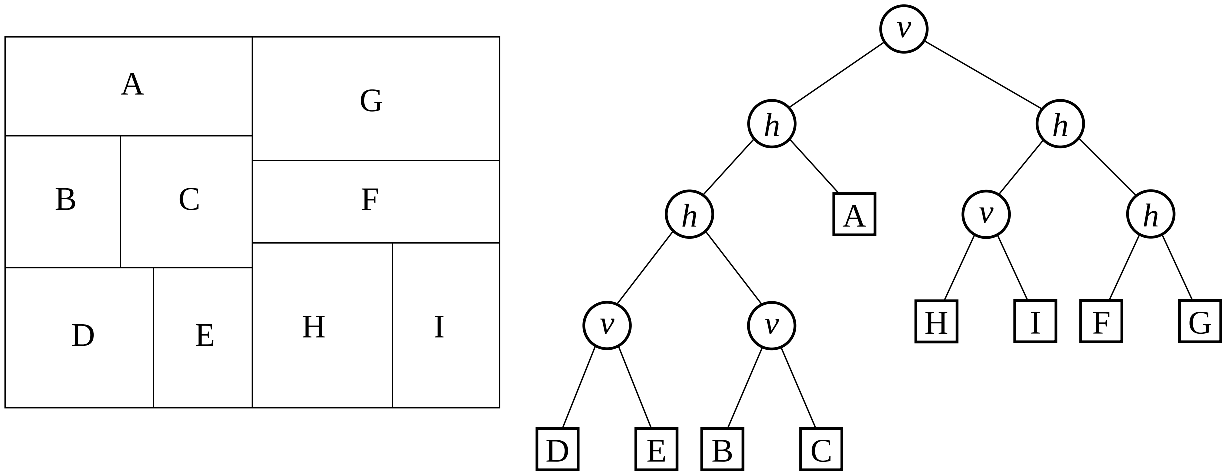


**Figure 4.6:** A floorplan (left) and one possible slicing-tree (right)

The left pattern in figure 4.7 is a slicing floorplan, the other graphs represent two minimal non-slicing patterns. To describe the middle pattern with a tree, at least a five-ary tree is needed, for the right pattern, a seven-ary tree must be used [Müll90].

For the binary case, as shown in figure 4.6, each inner node of the slicing-tree is labeled with the cut direction $v$ and $h$ for vertical and horizontal cuts, respectively. The computation of a minimal area floorplan, i.e. identification of those shapes for all cells yielding an overall minimal area for the whole layout, can be done in polynomial time with an algorithm presented by Stockmeyer [Stoc83]. For the case of binary slicing-trees with unoriented cuts, the floorplan-sizing can also be done in polynomial time, on condition that the tree is node-balanced. This result was presented by Lengauer and Müller [LenMül87], for general trees they show that the sizing problem is weakly NP-complete.

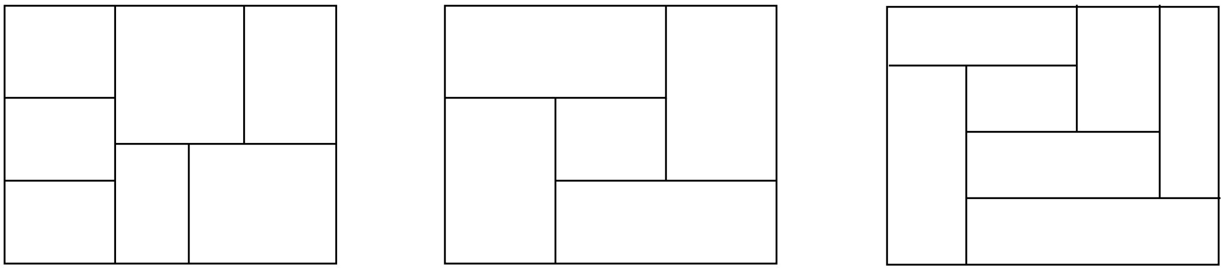

**Figure 4.7:** A slicing (left) and two non-slicing floorplans (middle, right)

## 4.3 Iterated Matching

The method described in this section can be used to hierarchically pair items of a given set to yield a global maximal quality, measured over all pairings at each level. The basic concept used for this is the graph-theoretical computation of a *maximum weight matching* on a weighted graph. A *matching* is a set of pairwise disjunct pairs of adjacent vertices in this graph. The weight of the matching is the sum of the weights of the edges between comprised vertices.

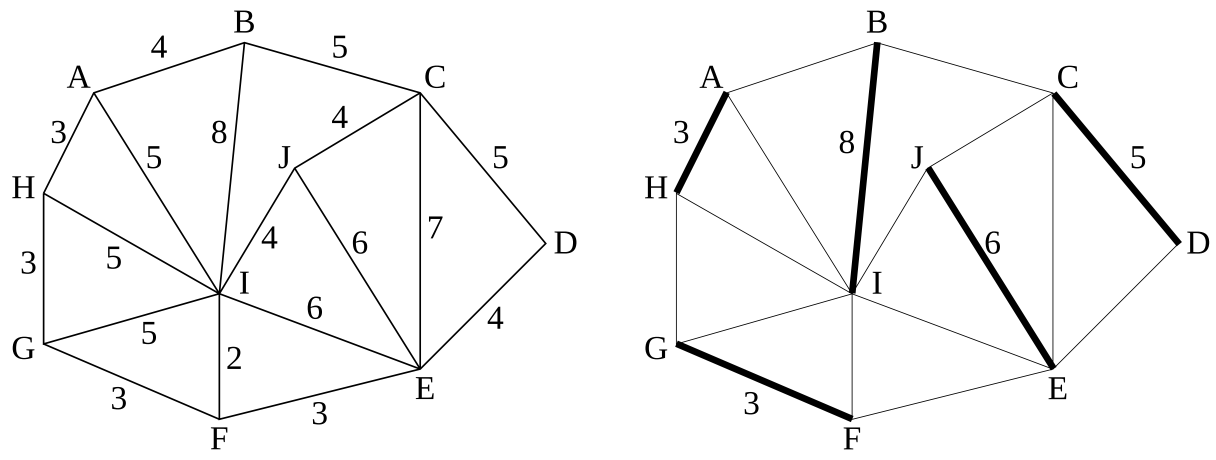


**Figure 4.8:** A weighted graph (left) and a matching with weight 25 (right)

Figure 4.8 shows a sparse graph (left) and a matching consisting of the pairs (A,H), (B,I), (C,D), (E,J), and (F,G), which results in a weight of 25 for this matching. The computation of the maximum weight matching on a complete graph with $n$ vertices can be practicably done in time $\mathcal{O}(n^3)$, the asymptotically fastest algorithm runs in time $\mathcal{O}(n^2 \log n)$ [Leng90].

The iterated computation of maximum weight matchings can be applied as a heuristic to solve combinatorial optimization problems, e. g. cutting stock [FriVor94], Traveling Salesman Problem [Lauxt96], or VLSI-Placement [SchVor96a]. For the two-dimensional packing problems, a pattern with minimal waste, i. e. minimal area of the circumscribing rectangle, has to be computed. As defined in section 4.2, for the construction of slicing patterns, the blocks can be hierarchically composed to partial patterns in a slicing-tree. Here at each level, blocks have to be paired to yield matchings with minimal total waste. For that purpose a complete graph can be constructed with vertices representing the blocks, and edges weighted with a function value describing the quality of a pairing of both adjacent blocks. For packing problems this might be the ratio of occupied area inside the rectangle circumscribing both blocks, if these were arranged next to each other. A maximum weight matching on this graph defines a set of block pairings with an overall minimal waste inside the corresponding patterns.

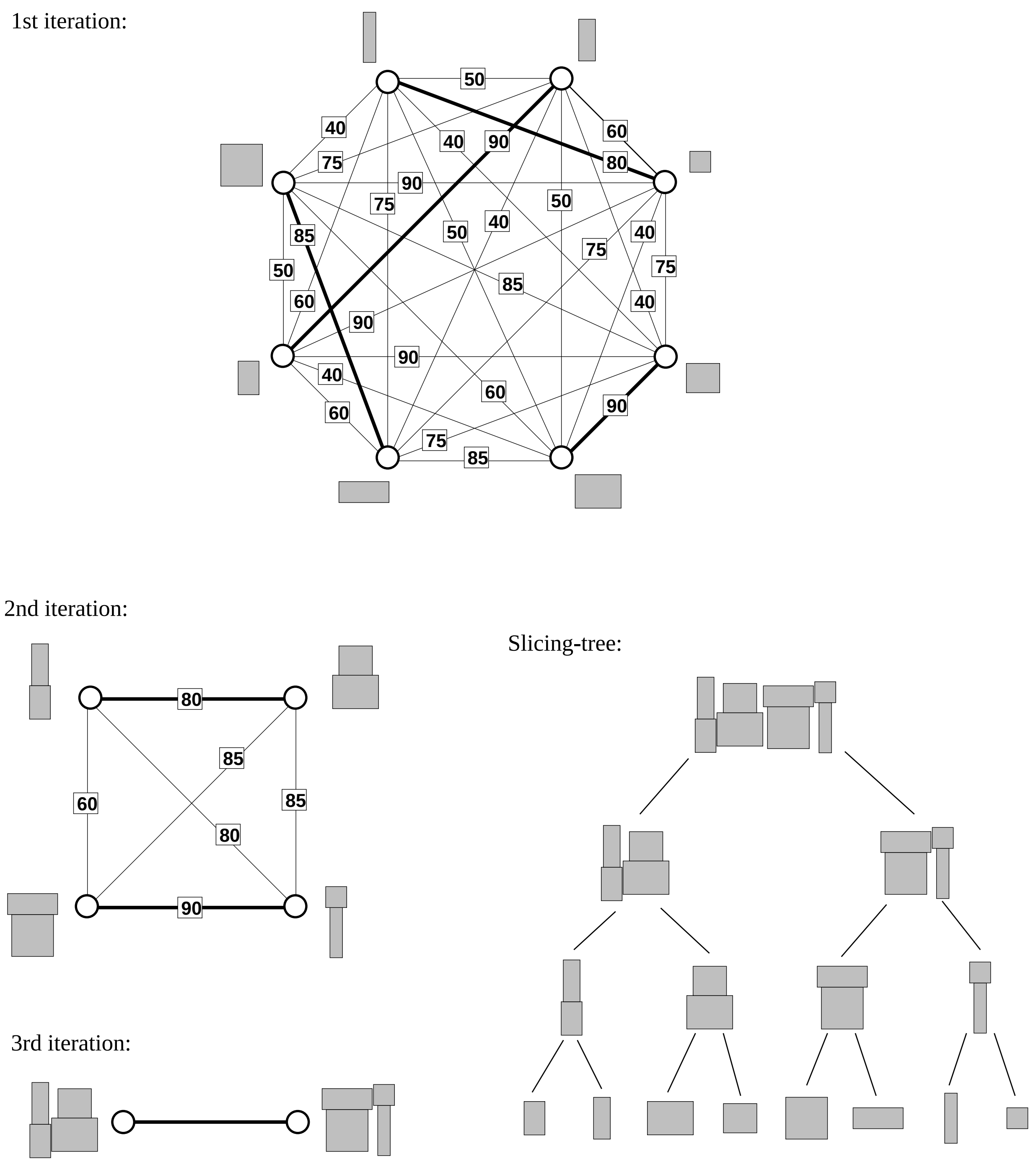


**Figure 4.9:** The construction of a packing pattern for 8 blocks using the iterated matching heuristic

To generate a complete packing pattern containing all blocks, this procedure has to be iterated. In the second iteration each vertex in the complete graph to be constructed characterizes a pattern consisting of two blocks, as it has been determined in the first iteration. Again, edges are weighted with values describing the quality of feasible patterns, now consisting out of four blocks each. If the matching in the previous iteration has not been perfect, i. e. not all of the blocks have been paired, some vertices in the second iteration still represent single blocks. This process is repeated until the last two patterns are combined to provide a pattern defining a packing scheme containing all blocks. Figure 4.9 shows the application of the iterated matching for finding a high quality arrangement of eight blocks, thus taking three iterations. As it can be gathered from this figure, during each iteration one level of the slicing-tree characterizing this pattern is computed.

For the placement problems, other criteria can be chosen to value a possible pairing during the iterated matching. The function determining the edge weights of the matching graph during the computation of an arrangement for a set of facilities in the facility layout problem may be based on the flow between the adjacent facilities. This takes care that facilities with high flow will be placed next to each other on the layout site.

# Chapter 5

# Hybrid Genetic Algorithms for Packing and Placement

In this chapter the problem-specific implementations of genetic algorithms for placement and packing problems are presented. A common concept of all approaches is the tree-structured genotype representation. In all algorithms the methods introduced in chapter 4 are used for implicit hillclimbing. After the description of the general genetic operators for tree-structured genotypes, the parallelization and the strategy adaptation are explained. The first hybrid genetic algorithm to be introduced is that for the two-dimensional bin-packing problem, since this is the simplest of the three applications. Then the specific aspects of the algorithm for the facility layout problem are described, and at the end of this chapter the problem-specific genetic algorithm for the VLSI layout generation task is presented. This approach integrates the computation of the global routing into the placement task, thus deals with the concurrent optimization of two tasks, which are usually separated due to complexity constraints.

## 5.1 Tree-Structured Genotype Representation

From chapters 3 and 4 it is obvious that a solution for the optimization problems treated in this work can be described by a pattern that defines the arrangement of the items to be placed or packed. Thus, the phenotype representation is this pattern with some additional information, e. g. the routing information for the VLSI layout generation problem. Slicing-trees have been already introduced in chapter 4 to represent cutting patterns or placements. In the simplest case these trees are binary, which corresponds to slicing patterns that can be generated by a hierarchy of guillotine cuts starting from the whole placement. Thus, the binary slicing-tree is a proper genotype coding for these problems.

While trees have already been used in genetic algorithms for bin-packing [Krög93, Krög95] or VLSI floorplan optimization [CohPar86, CoHeMaRi91], in those approaches the genotype is still encoded by a string. This string represents a pre- or postfix-representation of the tree, and the genetic operators often work on this string without considering the structure of the tree. During recombination, partial arrangements of blocks have to be transmitted from parent individuals to offspring. The corresponding operation is the inheritance of subtrees from the parent individuals. Encoding the tree in a postfix string complicates this operation, since in that case the string has to be decoded into the slicing-tree for recombination. After that, the offspring tree has to be transformed into a string representing the offspring chromosome. There is no reason for this except the analogy to the natural evolution process, where the genetic information is encoded in a DNA-string. When directly using the slicing-tree as the genotype representation, there is no need for further decoding or encoding this tree when applying genetic operators to it.

The use of trees as a genotype coding is already well known from *genetic programming* [Koza92]. Here the size of the individuals in a population is vastly different, and there are no restrictions with regard to the structure of the trees. For the representation of patterns like it is applied in this work, the trees differ from those used in the genetic programming in one main point: all trees have a fixed size, and they have the same set of leaves. Due to that, the application of the operators is much more complicated as in the genetic programming. Problem-specific operators are used, which always ensure that only correct offspring are generated.

### 5.1.1 Mutation

In most implementations of genetic algorithms, mutation is the less used, but not the less important operator in comparison to recombination. Each gene in an offspring that has been created by recombination, might be mutated with respect to the mutation probability (or mutation rate), which is usually proportional to the size of the problem (length of the chromosome). The mutation operator produces genetic information that is new to the population, or re-introduces information that has been eliminated by selection, but might be helpful during the current state of the search process.

There are two straightforward mutation operators that change the structure of a tree. Figure 5.1 shows an example for an operator that exchanges two parts of a tree. Part A is a single leaf, while B is a subtree containing three nodes. At the phenotype level, this corresponds to exchanging the block represented by leaf A with the placement or pattern for the set of blocks characterized by subtree B. For this mutation operator three different cases can be distinguished:

1. exchanging two leaves of the tree
2. exchanging a leaf with an inner node (a subtree)
3. exchanging two inner nodes, or exchanging two subtrees, respectively

The second mutation operator (figure 5.2) picks up a part of the tree (leaf A) and inserts it at a different position (x). This corresponds to cutting a block or a partial placement out of the complete packing pattern and moving it to a different position. Note that this transformation cannot be done by only exchanging leaves or subtrees, thus represents an essential mutation operator.

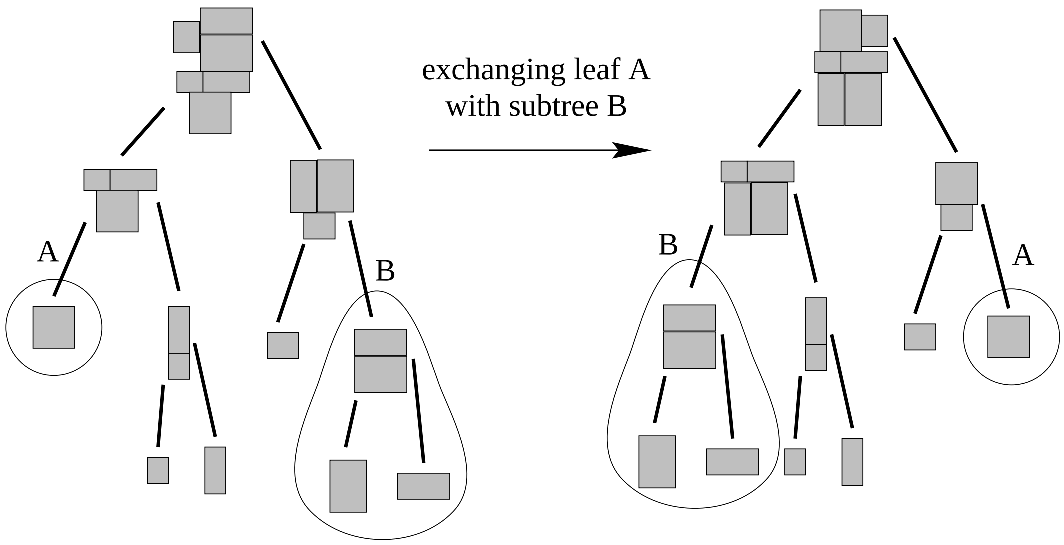


**Figure 5.1:** Mutation by exchanging subtrees

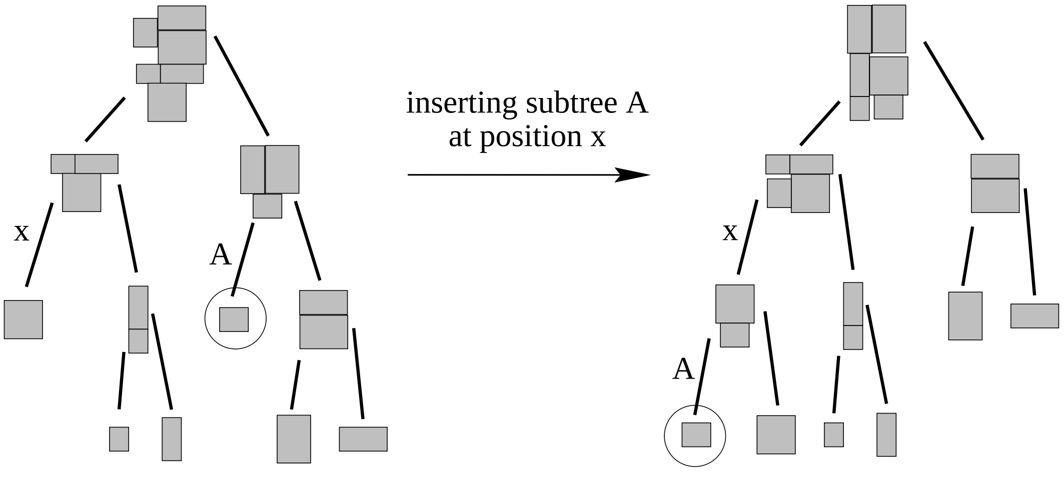


**Figure 5.2:** Mutation by changing the structure of the tree

A third mutation operator does not directly change the structure of the tree, but converts the pattern described by this tree by changing the orientation of one of the blocks. In figure 5.3 the orientation of the pattern defined by the left subtree of the root node is changed from an upset orientation to an upright orientation.

In these examples, as for the remainder of this work, blocks or sub-patterns in a tree defining a placement or a packing pattern are always placed upon each other. The pattern characterized by the right successor of an inner tree node is always positioned upon the

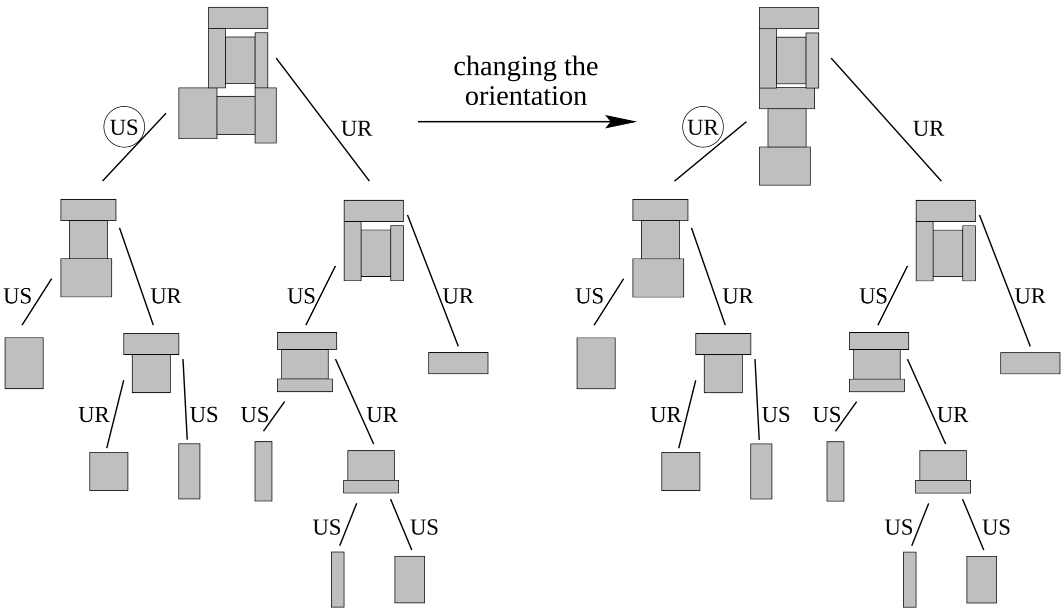


**Figure 5.3:** Mutation by changing the orientation of a sub-pattern

pattern characterized by its left successor, when combing both parts to a pattern or meta-block.

### 5.1.2 Recombination

Recombination is the most important search operator in genetic algorithms. Usually, it is a sexual operator, that constructs one or two offspring out of the genetic information encoded in two parent individuals. The obvious way for creation of one offspring out of two tree-structured individuals is to combine some disjunct subtrees of both parents to a tree for the offspring. Figure 5.4 shows the operation of this recombination operator. One offspring is produced out of parts of two parent individuals. A set of disjunct subtrees in both parents is randomly chosen and builds a pool of building blocks out of which a new individual is composed. Leaves that are not contained in these subtrees are additionally inserted into this pool. During the creation of the offspring, these subtrees are combined to a complete tree. Here the iterated matching can be used to identify sets of good pairs of these subtrees.

The recombination, as shown in figure 5.4, combines disjunct subtrees of only two parent individuals when it creates an offspring. Two-parent or sexual recombination is inspired by the natural evolution process. However, this must not be the best recombination scheme in artificial evolution like genetic algorithms. In section 2.4 some approaches to recombination with more than two parents have been described. Bersini and Seront [BerSer92] used three parents, similar to Mühlenbein's majority vote [Mühl89], and Eiben *et al.* [EiRaRu94]

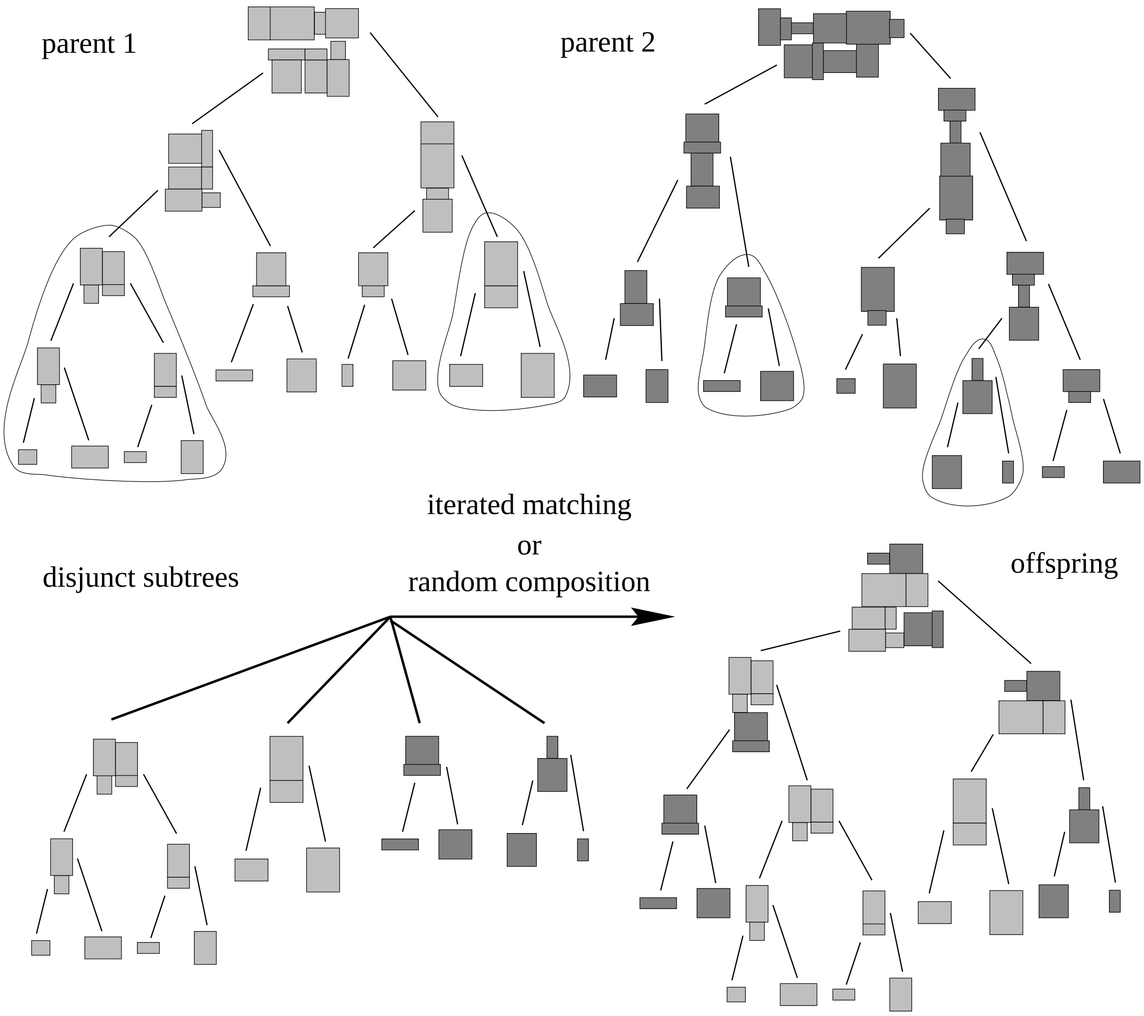


**Figure 5.4:** The recombination operator chooses disjunct subtrees in both parents and combines them to an offspring tree

use up to ten parents. In the field of evolution strategies, multi parent recombination has also been introduced and is called 'global recombination' there [BäcSch93, Schw95]. In this work a different recombination scheme is suggested. It is called *gene-pool recombination* with regard to the theoretical concept of the same name which has been established by Mühlenbein and Voigt [MühVoi95].

In this recombination scheme, the genetic information of a given set of parents is inserted into the pool, out of which the genotypes of the offspring are created. This information is encoded in the structure of the tree, and, consequently, parts of it in every subtree. Thus every subtree of a parent individual is inserted into the pool, figure 5.5 shows the set of these subtrees for a particular parent individual. Note that the marked subtree consisting of three nodes occurs three times, since it is included in two more subtrees. Because offspring creation is done by randomly choosing disjunct subtrees out of the pool, this subtree will

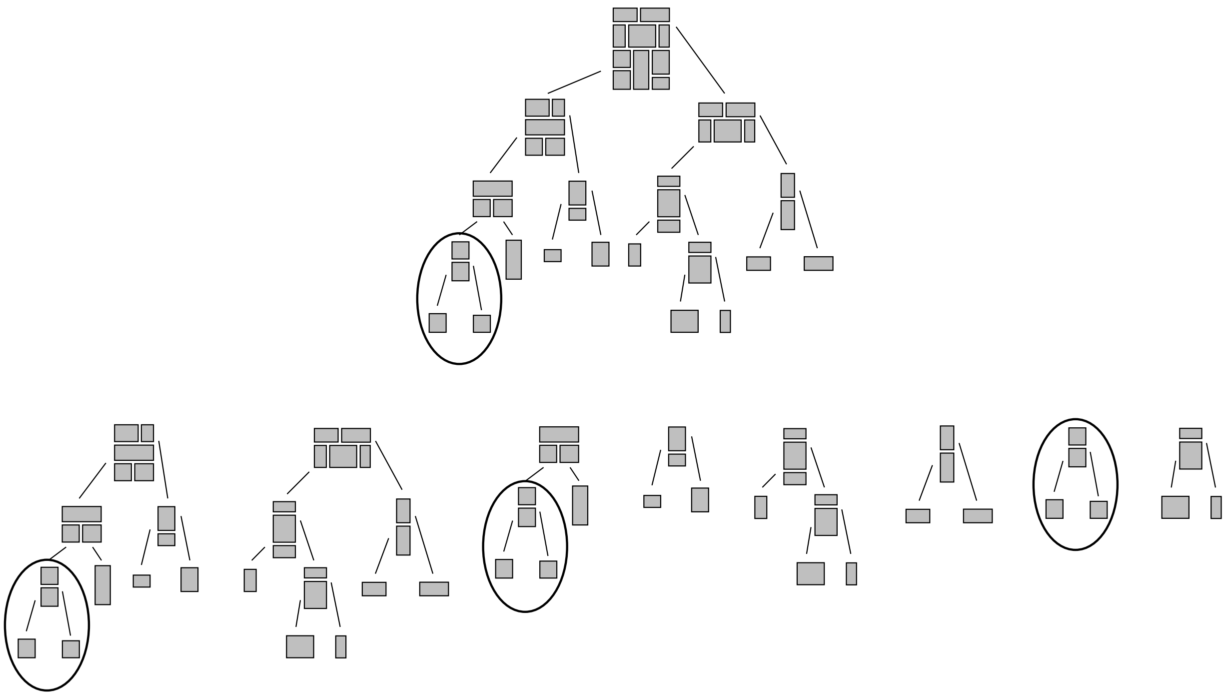

**Figure 5.5:** A parent tree (top) and all of its subtrees which are inserted into the gene-pool

get a higher chance to be selected. If this subtree is already included in more than only one of the parent individuals, there exist even more trees in this pool which include that tree. This corresponds to Goldberg's *building block hypothesis* [Gold89]. These building blocks are short, low-order, and highly fit *schemata* that are sampled, recombined, and resampled during the search. Like those building blocks, smaller trees get a higher chance of being contained in one or more of the offspring that are created out of the genetic material in the pool.

### 5.1.3 Selection and replacement

For the choice of individuals as parents for the recombination operation, truncation selection is used, i. e. only the fitter individuals are selected for recombination. For mutation, any of the individuals in the population can be chosen with equal probability. A steady-state type of genetic algorithm is used, thus an individual may survive for longer than only one generation. Individuals are replaced at the end of each generation, if the qualities of the offspring are better, or if these are quite different from all the members of the current population. A replacement scheme based on diversity between offspring and parent has already been successfully used by Freisleben and Merz [FreMer96] in a genetic algorithm for the traveling salesman problem. There the number of non-common edges in the tours encoded in parent and offspring are taken as a measure for the diversity of both individuals.

The difference between two individuals in this application is computed at the genotype level: the measure for diversity is the number of subtrees that do not contain the same set of leaves. Although this measure does take into consideration neither the structure of the trees, nor the orientations of the blocks, it is efficient to compute and characterizes the extent of rough diversity.

An offspring is always inserted into the population if its fitness is higher than the fitness of the best individual. In this case the individual with the worst fitness in the whole population is replaced. If an offspring is not better than the best individual, that individual in the current population is identified which is most similar to this offspring. If their diversity is below a given threshold, i. e. they encode quite similar solutions, the better one of both individuals is kept in the population. If the offspring is diverse from all individuals, it encodes a lot of new genetic information and thus enters the population by replacing the least fit individual without consideration of its own fitness.

---

For all islands in parallel do

1. Initialize local population of size $n$
2. For $m$ generations do
   (a) select the $t\,\%$ best individuals
      - insert all subtrees of these individuals into gene-pool
      - create $n \cdot p_{rec}$ offspring by recombination

   (b) copy $n \cdot p_{mut}$ randomly chosen individuals of the current population
      - choose one mutation operator for each of these offspring
      - mutate offspring

   (c) replace some less fit individuals of the current population by fitter or vastly different offspring
3. Choose one individual randomly to migrate to a neighboured subpopulation
4. If adaptation interval is not expired, go to step 2.
5. Adapt strategy parameters
6. If termination criterion is not fulfilled, go to step 2.

---

**Figure 5.6:** The outline of the adaptive parallel genetic algorithm used in this work

### 5.1.4 Parallelization and strategy adaptation

All results presented for the problems considered in this work are achieved with parallel implementations of genetic algorithms. A stepping stone model is used, i. e. there exists a

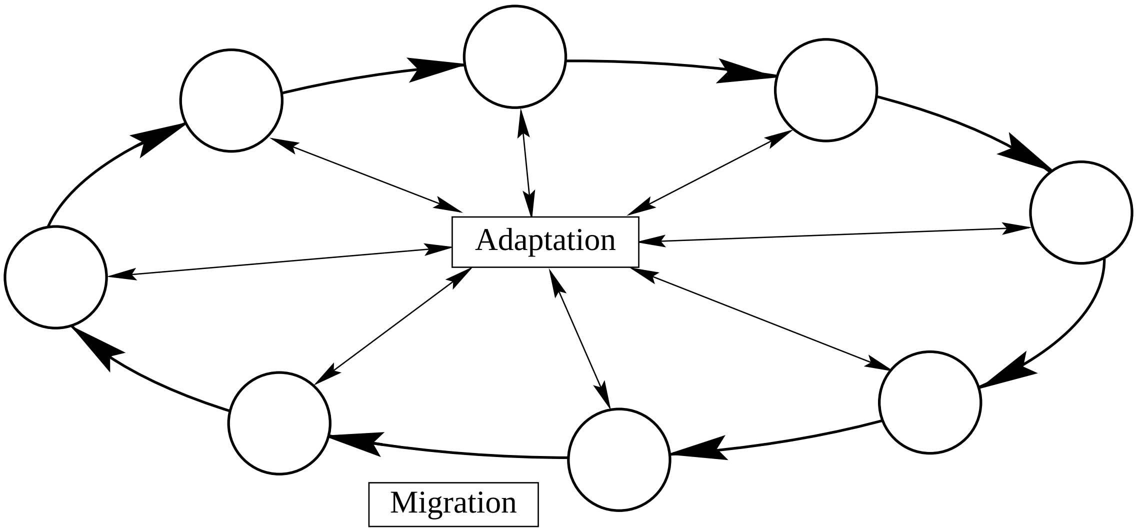


**Figure 5.7:** The parallel genetic algorithm uses a ring-structure for the parallelization and a centralized adaptation mechanism

set of subpopulations located on islands, that are connected via a fixed topology, on which individuals migrate between these subpopulations. A sequential version of the hybrid genetic algorithm is executed on each island. After a fixed number of generations, one randomly selected individual migrates to a neighboured island. The islands are ideally distributed on a system of parallel processors and are executed in parallel. They are connected by a ring topology, via this topology the migration takes place. The individuals migrate on a fixed direction to the next island on this ring (see figure 5.7). An outline of the adaptive parallel algorithm is given in figure 5.6.

While the number of islands, the size of the subpopulations, and the migration interval are fixed during one run of the genetic algorithm, there is a set of strategy parameters that are adapted during the search:

- the recombination rate $p_{rec}$, correlated to this is the mutation rate $p_{mut} = 1 - p_{rec}$
- the frequency of the different mutation operators
- the use of the iterated matching during recombination

In section 2.5 (page 14) different approaches to the adaptation of strategy parameters in genetic algorithms have been described. These approaches can be classified by the level at which the adaptation takes place: this can be either at the level of a single individual, at the level of subpopulations, or at the level of the whole population.

In the following a strategy adaptation scheme based at the subpopulation level is described. It is comparable to the approaches based on a hierarchy of genetic algorithms like the approaches by Bäck [Bäck94], Freisleben and Härtfelder [FreHär93], or Wang *et* *al.* [WaGoPu96], since in those approaches populations in parallel are executed with different parameters. It is similar to the last of these approaches, since this is also a parallel genetic algorithm with individuals migrating between the populations. While this is also true for the *competing subpopulations* of Schlierkamp-Voosen and Mühlenbein [SchMüh94, SchMüh96], in that case the strategies are fixed during each run.

In this work a steady state genetic algorithm is used, and in each generation the same number $n$ of offspring is generated ($n$ is also the population size). Since recombination and mutation are treated as equivalent operators, each offspring is generated either by recombination or mutation. Thus, the recombination rate is one of the parameters that characterize a strategy.

As described in section 5.1.1, there exist different mutation operators, which are applied with different frequencies. The vector defining the frequency of a mutation operator, or — in other words — the probability for a particular operator to be chosen, is also part of a strategy.

The iterated matching heuristic as it has been introduced in section 4.3, is helpful to generate high quality building-blocks when composing a solution. However, the benefit of this heuristic depends on the problem size and the stage of the optimization process. It is very helpful for problem instances with a large number of items to be placed or packed, especially during the beginning of the optimization. But since it is a deterministic process, it appears as being disadvantageous at the later stage of the search (examples for this will be presented in chapter 6). Thus, a switch is included in the strategy parameter set that defines whether the iterated matching is used during recombination or not.

One of the main details of self-adaptation of genetic algorithms is the quality, due to which the different strategies are ranked. Freisleben and Härtfelder [FreHär93] use the increase of the fitness over a fixed period, while Bäck [Bäck94] uses the absolut value of the objective function after a given number of evaluations as a quality criterion. Wang *et al.* [WaGoPu96] propose two different criterions, the first is based on the gain of the average fitness, while the other is similar to this increase with a focus on the absolute current fitness. The criterion used by Schlierkamp-Voosen and Mühlenbein [SchMüh94, SchMüh96] is also based on the fitness of the best individual of each subpopulation. To avoid inefficient oscillation of the subpopulation sizes (they adapt the size of the subpopulations and not the strategies), it is taken into account in how many generations of a given time window the best individual of a subpopulation is the globally best individual in the whole population.

Since the usual size of each subpopulation in the implementations of the hybrid genetic algorithms that will be described in the following sections is only ten individuals, a quality criterion based on the fitness of the best individual is disadvantageous. This value might be too much influenced by migration of individuals. Although the migrating individuals are randomly chosen from each subpopulation, a progress measure based on the best fitness may be falsified, since migration occurs more often than adaptation. The quality on which the ranking of the strategies is based depends on the *response to selection*, as it is used in livestock breeding [Falc81] and in the *Breeder Genetic Algorithm (BGA)* [MühSch94]. It describes the difference between the population mean fitness values of two successive generations. The response is also defined as the product of heritability and the selection differential. The livestock breeder use the latter definition to estimate the response during the breeding process [Falc81]. The simple definition of the response as the progress of the mean fitness in the population is an ideal candidate for a quality criterion to rank the strategies during the adaptation process. Since even inferior offspring are accepted according to their diversity from the individuals in the current population, only the response measured over the 50% best individuals is taken as the quality criterion.

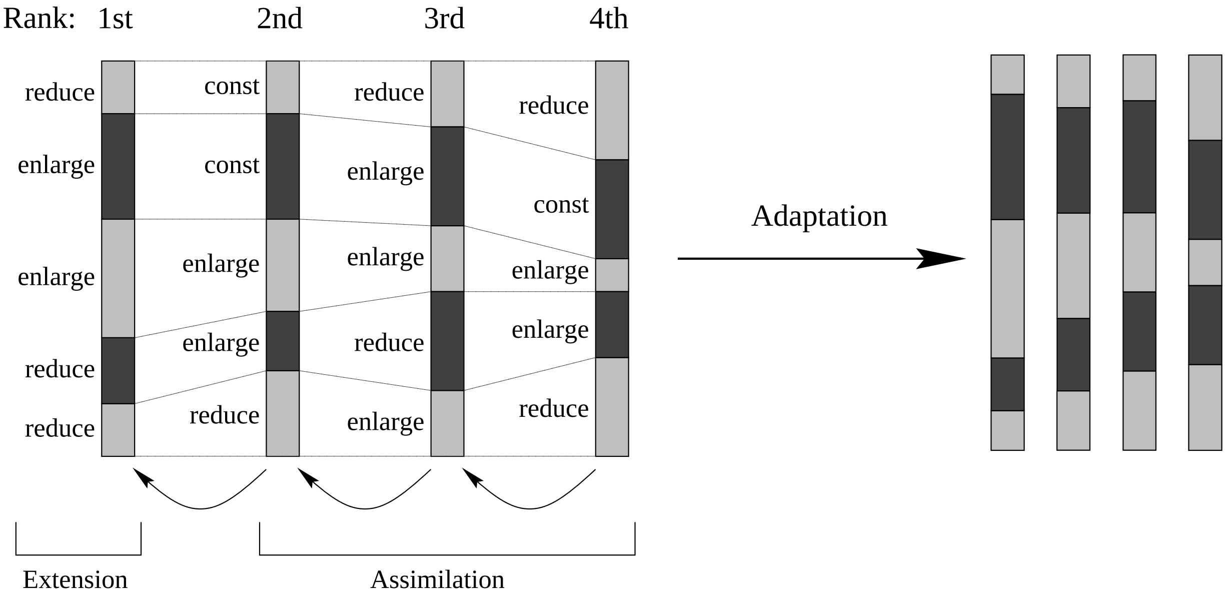


**Figure 5.8:** The adaptation of four parameter vectors, for the best strategy the characteristical parameters are extended, the values of the second, third, and fourth parameter set are assimilated to the corresponding values of the first, second, and third strategy, respectively

The strategies are initialized both with moderate parameter settings, and with extreme parameter settings, e. g. only using one out of all possible mutation operators. After a fixed number of generations, the *adaptation interval*, the strategy parameters and statistical information of all islands are collected. The vectors describing the strategy parameters are ranked according to the average response to selection each island gained over those generations. Starting with the last strategy in the list, all parameters are adapted to the corresponding values of the next better strategy, as shown in figure 5.8. There an example of four ranked parameter vectors is given. Suppose the shown vectors define the frequency of five different mutation operators. During the adaptation, the values for frequencies in the second to the fourth vector are assimilated to the particular value in the next better strategy, i. e. if a particular operator in the next better strategy has a higher frequency, than its frequency in the current vector is enlarged. The characteristical values of the best vector are strengthened, i. e. if the frequency of a particular operator is quite small in this strategy, then it is further decreased.

The strategy adaptation described above starts with different wide spread parameter sets. After some adaptation intervals, a successful strategy is recognized, and the remaining strategies are assimilated to this one. Since the best of all strategies is steadily extended, it is avoided that eventually all strategies get equal. If no progress is made on one island, which is noticed by a response equal to zero or even a negative response, its strategy is getting stuck. In this case it is adapted to one of the initial settings instead of being assimilated to one of the current strategies.

This strategy adaptation scheme is quite similar to those based on a meta genetic algorithm. In comparison to them it is faster in reacting to gained progress on particular islands by assimilating all strategies to that used on this successful island. In the meta genetic algorithms proposed for adaptation, this is crucial. Since it is a genetic algorithm, on the one hand it takes some time to recognize a good strategy, on the other hand diversity in the population consisting of all parameter vector has to be ensured to avoid premature convergence of the adaptation process.

## 5.2 A Hybrid Genetic Algorithm for the Bin-Packing Problem

The bin-packing problem is the simplest of the three constrained packing and placement problems dealt with in this work. The only constraint is the width of the bin, inside of which all blocks have to be packed with an overall minimal height. A packing pattern is described by a binary slicing tree, thus only guillotinable patterns are generated.

This tree is composed in a bottom-up fashion. The orientations of the blocks combined in an inner node, or — in other words — the cut direction, is kept flexible during the creation of the tree. Generalized shape-functions, as introduced in section 4.1, are used to consider all possible orientations of the blocks during the construction of the tree. However, due to the given width of the bin a pattern has to be fitted inside, some implementations can be already skipped: for a bin with width $w_{bin}$ and an overall block area of $a = \sum_{i=1}^{n} w_i h_i$, $a/w_{bin}$ is a lower bound for the height of an optimal packing pattern. When considering patterns with up to 10 % waste during the search, implementations with height or width larger than $\max\{1.1 \cdot w_{bin}, 1.1 \cdot a/w_{bin}\}$ can be neglected, since these would lead to patterns with more than 10 % waste.

After the construction of the tree, its root defines several patterns. Out of these the pattern with minimal height is chosen. During a top-down traversal of the tree the orientations of all blocks that conduct to this optimal pattern can be identified, since each minimal implementation encoded in the shape-functions is labeled with the orientation of the comprised blocks. The packing height defines the fitness of the individual.

Not all operators defined in section 5.1 are used during the optimization process. The mutation operator which rotates blocks or meta-blocks is not used, since all admissible orientations are taken into account, and the globally best orientations for the particular structure of the slicing-tree are identified during the top-down traversal. After the application of an operator, not only the inner nodes of the tree on the way from those nodes, where a change occurred, up to the root node have to be recomputed, but the whole tree has to

1. Bottom-up construction of the slicing-tree:
   - random pairing of blocks
   - composition of general shape-functions with encoding of all possible block orientations
   - neglecting meta-blocks with infeasible shapes with regard to bin width
2. Choice of implementation with minimal height
3. Top-down traversal of the tree to fix orientations of the blocks
4. Packing height defines fitness

**Figure 5.9:** The operations needed for the evaluation of an individual in the hybrid genetic algorithm for the two-dimensional bin-packing

be updated. If during recombination the orientations inside the transmitted subtrees were adopted by the offspring, a large potential is not utilized: in that case only fixed patterns are transmitted, which might not match very well. If transmitting the subtrees without fixed orientations, the optimal pattern for the resulting offspring tree can be identified.

## 5.3 A Hybrid Genetic Algorithm for the Facility Layout Problem

The complexity of the facility layout problem, as it has been introduced in section 3.3 (page 26), is mainly caused by the high degree of freedom due to the flexibility of the shapes of the facilities. Most of the previously reported approaches solve this problem by using a multi-stage method. Facilities are initially considered as circles to compute a preliminary placement and are then refined to assume a rectangular shape. In the following a genetic algorithm for the layout of non–identical facilities is presented, which uses concepts from VLSI floorplanning. As in the application described before, a layout is represented by a binary tree with non-fixed orientations for the blocks during its bottom-up creation. Different shapes for the facilities are encoded in shape-functions, like it is usual for flexible cells in VLSI design.

### 5.3.1 The preparation of the input data

The input to the simple non–identical facility layout problem are the area, a lower and upper bound for the aspect ratio for all facilities, and a matrix defining the flow between each pair of facilities (cf. section 3.3). Since in this work only discrete shape-functions are used to deal with flexible blocks during the creation of a solution, a set of feasible shapes

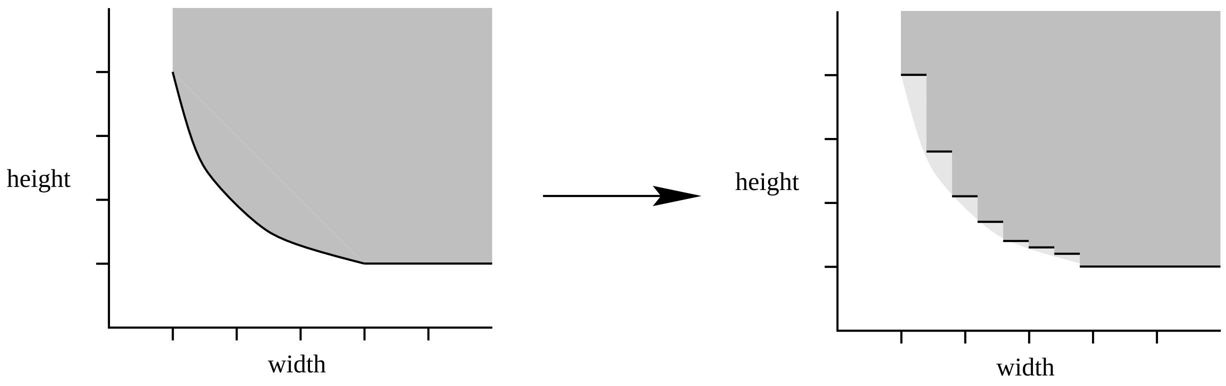


**Figure 5.10:** The transformation of a continuous shape-function to a discrete shape-function with eight minimal implementations

for each facility has to be generated. Figure 5.10 shows the transformation of a continuous shape-function into a discrete shape-function. Note that the widths of the admissible shapes are evenly distributed among the admissible interval, while the distribution of the heights is skew. Since in this work instances of the facility layout problem including only free facilities are considered, each facility can be rotated before placing. This leads to blocks with equally distributed heights after placement on the layout site.

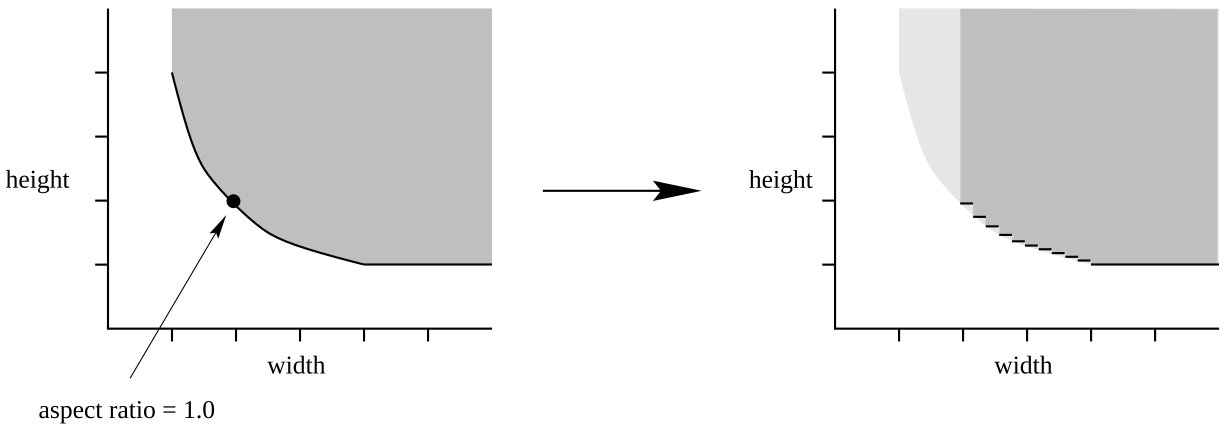


**Figure 5.11:** The transformation of a continuous shape-function to a discrete shape-function for a facility with a free orientation and a range for the aspect ratio that includes 1.0

Due to the non-fixed orientation, the discrete shape-function constructed above still contains some redundancy, if a quadratic shape is included. When considering a flexible facility $f_i$ with an aspect ratio range between $lr_i = 0.5$ and $ur_i = 1.5$, a ratio of 1.5 for an upset orientation when placing the facility can be simulated by an upright placement with aspect ratio of 0.67. Thus, only a range from $\min\{lr_i, 1/ur_i\}$ to 1.0 is needed for the aspect ratio of flexible facilities with free orientations in this case. Figure 5.11 shows the construction of a discrete shape-function for this case.

### 5.3.2 The initialization of the individuals

Like in the genetic algorithm for the two-dimensional bin-packing, the slicing-tree defining the placement of the facilities is created in a bottom-up fashion. A clustering of the facilities is computed by use of the iterated matching, the quality of a pairing of two simple facilities is defined by the flow between them. When composing clusters of facilities, the sum of the flows between the comprised facilities in both clusters defines the quality for this pair of clusters.

During the composition of an inner node of the tree, two feasible orientations for each of the facilities (blocks) or clusters of facilities (meta-blocks), respectively, are considered by using a generalized shape-function, as already described for the solution of the bin-packing (section 5.2). Since each block in the facility layout problem can assume several different shapes, the number of encoded implementations in the shape-function for a meta-block grows much faster than for the bin-packing problem with fixed blocks. Due to this, care has to be taken to keep the granularity (i. e. number of encoded minimal implementations) of the shape-functions tractable. E. g., when composing two meta-blocks with 20 minimal implementations in either shape-function, up to $4 \cdot 20 \cdot 20 = 1600$ different combinations might exist for the resulting meta-block. However, most of these implementations are dominated by others. To deal with this, a simple trick is used: if a shape-function encodes more than, say, 30 minimal implementations, at the next level only every second implementation is considered. This decreases the number of admissible shapes for a meta-block containing the block represented by this shape-function. Of course, this also reduces the accuracy of the overall solution, but it keeps the computation tractable. To generate really optimal shapes for the facilities, one has to use continuous shape-functions, which is computationally impracticable.

1. Bottom-up construction of the slicing-tree:
   - pairing of blocks based on sum of flows between comprised facilities
   - composition of shape-functions with consideration of all possible block orientations
   - reduction of number of implementations by neglecting some implementations
2. Choice of implementation with minimal area of the circumscribing rectangle
3. Top-down traversal of the tree to fix orientations and shapes of the facilities
4. Transformation of tree into geometrical layout
5. Computation of flow costs

**Figure 5.12:** The operations needed for the evaluation of an individual in the hybrid genetic algorithm for the facility layout problem

### 5.3.3 The fitness of an individual

After the construction of the slicing-tree that defines a complete layout for all facilities, the shape-function for the root of this tree characterizes a set of different placement patterns. Apart from the incorporating of all possible orientations for the blocks and meta-blocks, several identical patterns with different shapes for the facilities are included. For each encoded pattern the shape of the circumscribing rectangle can be directly determined from the shape-function. Since the main objective is the minimization of the overall flow costs, the centres of gravity for all placed facilities have to be computed. For that purpose, a transformation of the tree into a geometrical layout has to be done. Because this is a quite complex task, it is impracticable to do this for each implementation defined in the shape-function of the root node. It is supposed that a layout with an overall minimal area, i. e. with a minimal circumscribing rectangle, defines a high quality placement with small flow costs. Hence, this area is taken as an approximation for the objective function. The implementation with a minimal circumscribing rectangle is chosen, and during a top-down traversal of the tree all facilities are sized.

Given the slicing-tree with orientations for all nodes and sized leaves, the layout can be generated, and the coordinates for the centres of gravity of all facilities needed for the computation of the flow costs can be computed. The flow costs define the fitness of the layout represented by this particular individual.

## 5.4 A Hybrid Genetic Algorithm for VLSI Layout Generation

In the following an integrated approach to macro cell layout generation is presented. As described in section 3.1.1, due to complexity the layout generation is usually divided into several consecutive sub-tasks. Despite of the high interdependencies between both tasks, the global routing is separated from placing and sizing of the cells. During placement only an estimated amount of routing area is added between the cells. The global routes are determined with respect to the fixed capacities of the routing regions. In the hybrid genetic algorithm to be described in this section, a placement is encoded as a binary tree, and the exact positions of the cells are not fixed until the global routing is completed. Due to this there are no capacity restrictions during the computation of the global routes, and so always the shortest paths are chosen for the signal-nets.

### 5.4.1 The genotype representation

A layout is encoded by the corresponding binary slicing-tree. Due to the binarity of the tree, only slicing layouts are generated. As shown in figure 5.13, for a particular placement the leaves of this tree represent the macro cells (blocks), and an inner node (meta-block) describes a partial placement composed of the parts characterized by its successors. While

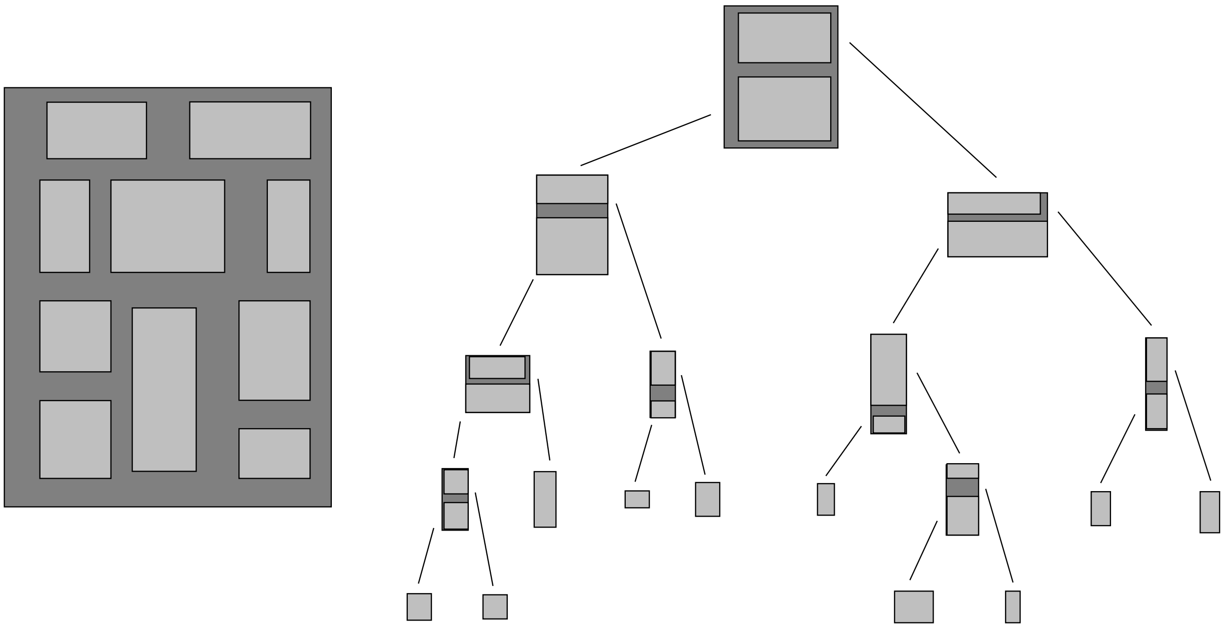

**Figure 5.13:** A placement (left) and its genotype representation as a binary slicing-tree with routing space added inside each meta-block (right)

the tree defines the relative arrangement of the cells, additional information is stored for each of its inner nodes:

- the shape for the meta-block
- the amount of routing space inside the channel
- the orientation of the two combined blocks

For the root node additionally the routing space on the four borders of the layout is stored.

The tree is constructed in a bottom-up fashion. When building a meta-block out of two blocks (or meta-blocks), the orientations of both blocks are fixed, and an estimated amount of routing space is inserted inside the channel between them. Encoding the orientations of the blocks in the shape-functions, like it is done in the hybrid genetic algorithms for bin-packing and the facility layout problem, is impracticable, since for the estimation of the routing space the terminals on the borders of each meta-block have to be known. If the particular circuit contains flexible macro cells, all non-dominated implementations for partial placements are stored in shape-functions for the inner nodes during the bottom-up construction of the tree.

### 5.4.2 Floorplanning

When combining two flexible blocks during the initial construction of the tree, there exist different shapes for the resulting meta-block depending on the admissible implementations for both blocks. Fixing the implementations for the blocks contained inside the meta-block would avoid this problem. Unfortunately, it is not clear at this time which shapes are globally optimal. Only local decisions can be made, for example those implementations for the blocks may be chosen that minimize the wasted space inside this particular meta-block. Locally this is the best choice, but a different shape may be better to minimize the area of the whole layout. Due to this, all shapes for a meta-block yielding from the combination of two flexible blocks after fixing their orientations are stored. Continuing this on the way to the root of the tree does not produce an exponential growth for the number of stored shapes, because most implementations for the meta-blocks are dominated (see section 4.1).

After the construction of the last meta-block, which is defined by the root, the tree represents a complete layout. Since there is also a shape-function for the root node, this single tree defines several layouts with different shapes. Out of these implementations, the layout with minimal area can be determined. During a top-down traversal of the tree, in each inner node those implementations for both successors are identified, which add up to the optimal shape for the meta-block represented by this node. After this traversal of the tree, for all flexible blocks those shapes are chosen, that make up an optimal layout, i. e. the flexible blocks are sized.

### 5.4.3 Integration of routing

Because of the large size of the search space it is profitable to start with a non randomly created initial population, which already contains high quality building blocks. This is again done by use of the iterated matching heuristic. Since in the layout generation problem the constraint defined by the routing of the signal-nets is even more important than the solution of the incorporated packing problem, the function defining the edgeweights of the matching graph is the number of nets with terminals on both adjacent blocks. Thus, in the first iteration of the matching process, those blocks are paired that represent macro cells with an overall maximal number of common nets.

In the following iterations the edges of the matching graph are weighted with the number of nets with terminals in both of the partial layouts characterized by the adjacent meta-blocks. The effect of this heuristic is comparable to the result of force-directed or partition based computation of a placement (section 3.1.2): macro cells with a lot of common signal-nets are placed near together.

While the iterated matching mainly reduces the overall wirelength on the layout, during the initial bottom-up construction of the tree routing is also considered by adding space in the channel inside each meta-block. When fixing the orientations of two blocks during the composition of a meta-block, those orientations are chosen that reduce the area and yield a maximal number of terminals to be directly connected inside the channel of this meta-block. For that purpose, the channelwidth is estimated by inserting a track for each net that has to connect terminals on both blocks, or — when combining meta-blocks — inside both meta-blocks. This number of tracks is reduced by the number of nets which have terminals on both borders of the channel (cf. figure 5.14). Thus the estimated channelwidth is minimal for those orientations that enable the direct connection of a maximal number of nets inside the channel. This channelwidth is added to the height of the meta-block.

All sixteen combinations are checked, and those orientations are chosen that yield a minimal area meta-block, including the estimated channelwidth. If the resulting meta-block is flexible, all sixteen possible shape-functions are computed, and the channelwidth is added to them (fig. 5.14). In this case those orientations are chosen that define the minimal area for the average of all implementations encoded in this shape-function.

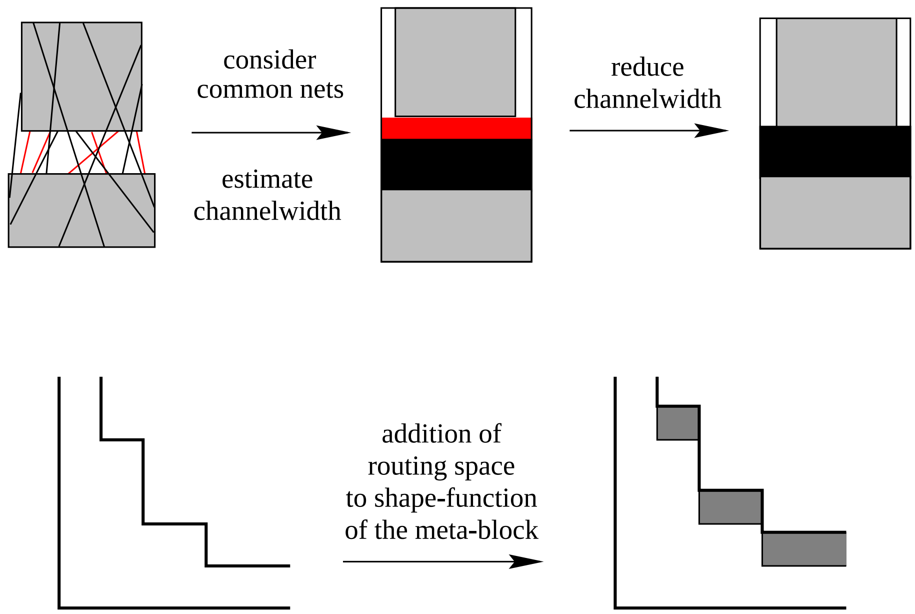


**Figure 5.14:** The insertion of estimated routing space inside a meta-block (top) and its addition to the shape-function (bottom)

### 5.4.4 Computation of the global routes

After the construction of the tree and the floorplanning, the routing graph as shown in figure 5.15 (left) is constructed. As in the usual method to compute the global routing (see section 3.1.3, page 22), the routing space is now represented by a set of routing regions. Such a region may be a channel between two blocks, or a partition of a channel inside a meta-block that combines two meta-blocks. In the latter case a region is a section in the channel, which is bounded by two orthogonal adjacent channels, i. e. it describes the tract on which two cells have contact inside this channel. To compute a global route for a net, the shortest paths between its terminals in the routing graph are determined. In contrast to the usual computation of the global routing, as described in section 3.1.3, the channels do not have any fixed capacities, but only an estimated width. The exact positions of the blocks are not fixed at this time, thus only pseudo-optimal shortest paths are computed. After the computation of all global routes, for each region the number of nets routed through it is known. For each net inside a region one track is added, which leads to the addition of an upper bound for the actual demand of routing space. The width of a channel is then set to the width of the widest of all regions that are included inside this channel. Since after floorplanning the exact terminal positions on all blocks are known, instead of adding one track for each net, a better heuristic can be used at this point to reduce the channelwidth.

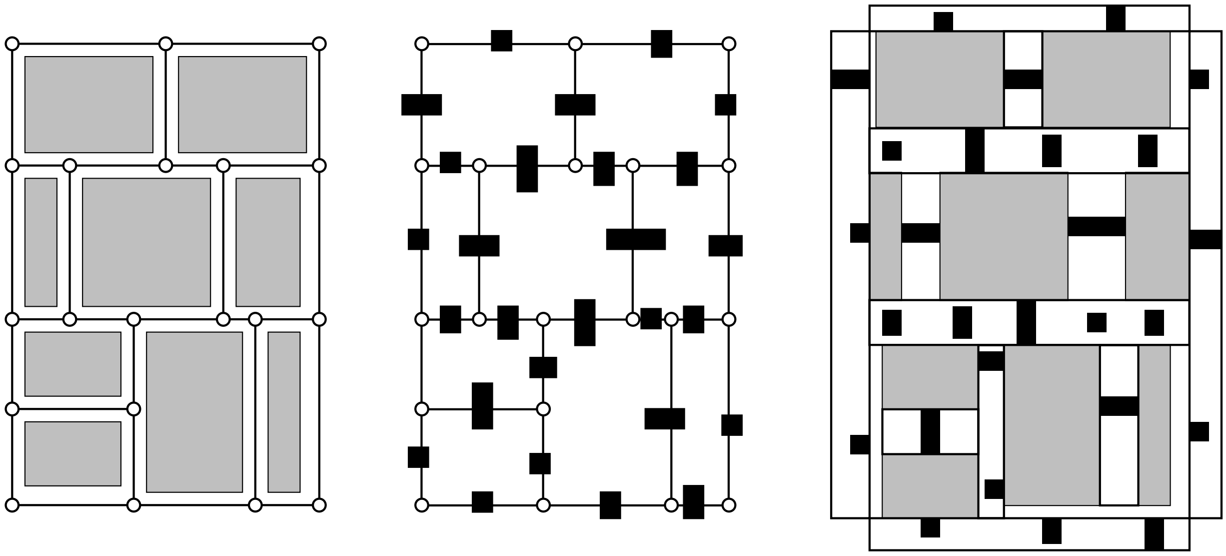

**Figure 5.15:** A routing graph (left), the width for the regions after the computation of the global routes (middle), and the resulting channelwidths (right)

When all channelwidths are determined, a bottom-up traversal of the slicing tree is done to add them to the heights of the meta-blocks. Then the area of the routing space on the border of the layout is added to the layout area, too. After that, the shape of the meta-block represented by the root node of the tree describes the shape of the total layout, thus the area of this meta-block defines the fitness of the corresponding individual.

### 5.4.5 Genetic operators

In the hybrid genetic algorithm for VLSI layout generation all mutation operators that have been defined in section 5.1.1 are used. Since the orientations of the blocks are fixed during their composition due to a local decision, which might not be the globally optimal combination, the mutation operator which randomly rotates blocks or meta-blocks is essential. In contrast to the update process after the operation of the mutation or recombination operators in the other applications, in this application not the complete tree is updated, but the inner nodes of the transmitted subtrees adopt the orientation labels form the corresponding nodes in the parent trees. Thus, a recomputation of the orientations is only done for all nodes in the offspring tree on the way from the changed parts to the root of the tree.

---

1. Bottom-up construction of the slicing-tree:
   - pairing of blocks based on the number of common nets
   - computation of the shape-functions for a resulting meta-block
   - addition of an estimated channelwidth
   - fixing the best orientations for two blocks to be combined in a meta-block
2. Choice of layout with minimal area of the circumscribing rectangle
3. Top-down traversal to size flexible cells
4. Transformation of the tree into a routing graph
5. Computation of global routes for the signal-nets
6. Determination and addition of channelwidth
7. Bottom-up traversal for computation of overall layout area

---

**Figure 5.16:** The operations needed for the evaluation of an individual in the hybrid genetic algorithm for VLSI layout generation

# Chapter 6

# Empirical Results

In this chapter empirical results for implementations of the hybrid genetic algorithms outlined in the previous chapter are presented. There is no data for sets of trails for each algorithm given to determine the best values for those parameters that are not adapted by the strategy adaptation. Performance results are shown for selected examples to provide a survey of the benefit of the used heuristics and their costs. Further, the advantages of the strategy adaptation and the gene-pool recombination are outlined by computational results. Some data to enable the estimation of the overall runtime and the scalability of the approaches is given. For each problem a set of benchmark instances is solved, and the obtained solution quality is compared with the results of other approaches which have been published before.

## 6.1 The Environment

All results presented in this chapter have been computed on *Parsytec GC/PP* parallel computers based on *Motorola MPC 601 PowerPC* processors with a clock speed of 80 MHz. The computations have been done on both a 64 processor machine located at the *$PC^2$ (Paderborn Center for Parallel Computing)* [PC2] at the University of Paderborn and a 128 processor machine at the Technical University of Chemnitz.

Unless otherwise stated, the standard algorithm is run on a net of 16 processors each holding a subpopulation of 10 individuals, which results in a population size of 160 individuals. In each generation a fixed number of offspring, equal to the population size, are generated. Migration of randomly chosen individuals occurs every three generations, the subpopulations are ranked for the adaptation every nine generations. All values shown represent average values of at least ten runs.

| | *xerox* | *hp* | *ami33* | *ami49* |
|---|---|---|---|---|
| **# cells** | 10 | 11 | 33 | 49 |
| **# nets** | 203 | 83 | 123 | 408 |
| **# terminals** | 698 | 309 | 452 | 958 |
| **# terminals per net** | 3.43 | 3.72 | 3.67 | 2.35 |
| **# I/O terminals** | 2 | 45 | 42 | 22 |
| **cell area** $[mm^2]$ | 19.4 | 8.83 | 1.16 | 35.1 |

**Table 6.1:** The benchmark circuits for the layout generation problem

## 6.2 The Benchmark Sets

### 6.2.1 VLSI layout generation

The hybrid genetic algorithm for the layout generation problem has been tested on real-life circuits chosen from a benchmark suite, which has been released for design workshops in the early 90s and is often referenced in the literature as the *MCNC-benchmarks*. They were originally maintained by *North Carolina's Microelectronics, Computing and Networking Center* [MCNC], but are now located at the *CAD Benchmarking Laboratory (CBL)* at North Carolina State University [CBL]. These benchmarks are all problems of the field of macro cell layout, the characteristics of the circuits are shown in table 6.1. These circuits only include fixed macro cells.

| | **# block** | **total block area** | **block size** | | **aspect ratio** | |
|---|---|---|---|---|---|---|
| | | | **average** | $\sigma$ | **average** | $\sigma$ |
| r25_1 | 25 | 10673 | 426.9 | 345.1 | 3.71 | 4.97 |
| r25_2 | 25 | 10274 | 411.0 | 419.8 | 4.61 | 6.60 |
| r25_3 | 25 | 10196 | 407.8 | 268.8 | 3.15 | 5.40 |
| r35_1 | 35 | 15028 | 429.4 | 378.2 | 2.55 | 2.14 |
| r35_2 | 35 | 12180 | 348.0 | 329.1 | 5.88 | 7.22 |
| r35_3 | 35 | 12243 | 349.8 | 331.8 | 5.68 | 7.34 |
| r45_1 | 45 | 19344 | 429.9 | 338.5 | 3.76 | 3.96 |
| r45_2 | 45 | 16216 | 360.4 | 333.8 | 6.17 | 7.93 |
| r45_3 | 45 | 13266 | 294.8 | 294.0 | 5.24 | 6.48 |
| r60_1 | 60 | 24866 | 414.4 | 362.9 | 3.61 | 3.96 |
| r60_2 | 60 | 27409 | 456.8 | 351.8 | 4.01 | 6.09 |
| r60_3 | 60 | 27904 | 465.1 | 358.1 | 3.34 | 3.58 |

**Table 6.2:** The benchmark set for the two-dimensional binpacking-problem

### 6.2.2 Bin-packing

The benchmark instances for the two-dimensional bin-packing problem are taken from the work of Kröger [Krög93, Krög95]. The number of blocks ranges from 25 to 60, the width of the bin is 100. Table 6.2 shows some statistical data for the 12 instances. For the size of the blocks and their aspect ratios the average values and the standard deviation $\sigma$ are presented. The size of the blocks varies over a wide range, and from the average values for the aspect ratio it can be gathered that the shapes of the blocks are quite unequal: an average aspect ratio of 4.01 for r60_2 means that the height of a block of this instance is on the average by a factor of four larger than its width.

### 6.2.3 Facility layout problem

| | # facilities | facility area | | aspect ratio range | | flow | |
|---|---|---|---|---|---|---|---|
| | | average | $\sigma$ | average | $\sigma$ | average | $\sigma$ |
| TL91–5 | 5 | 21.0 | 9.3 | 0.79 | 0.81 | 2.20 | 1.89 |
| TL91–6 | 6 | 20.4 | 8.6 | 0.82 | 0.74 | 2.40 | 2.65 |
| TL91–7 | 7 | 18.0 | 9.9 | 1.04 | 0.87 | 2.52 | 2.40 |
| TL91–8 | 8 | 17.7 | 9.3 | 0.99 | 0.82 | 2.75 | 3.14 |
| TL91–12 | 12 | 17.2 | 8.0 | 0.76 | 0.76 | 2.64 | 2.84 |
| TL91–15 | 15 | 16.3 | 8.4 | 0.64 | 0.72 | 2.83 | 2.82 |
| TL91–20 | 20 | 13.5 | 8.8 | 0.56 | 0.67 | 2.99 | 2.95 |
| TL91–30 | 30 | 11.9 | 8.4 | 0.45 | 0.59 | 2.55 | 2.78 |

**Table 6.3:** The benchmark instances for the facility layout problem

The eight instances of the facility layout problem have been proposed by Tam and Li [TamLi91] and consist of up to 30 facilities. Almost all of the facilities are flexible, and the area and a lower and upper bound for the aspect ratio are provided for each of them. There are neither constraints given for the planar site on which the facilities have to be placed, nor are there dead-space ratios. Especially the two largest problems represent very complex instances due to the high number of facilities. Even the authors introducing this benchmark set were not able to present solutions for the instances TL91–20 and TL91–30.

## 6.3 The Benefit of the Iterated Matching

The computation of a weight matching on a complete graph has been introduced in section 4.3 as a technique to generate globally optimal pairings for a large set of items. The iterated variant of this method excellently supports the creation of slicing-trees defining a packing or placement pattern.

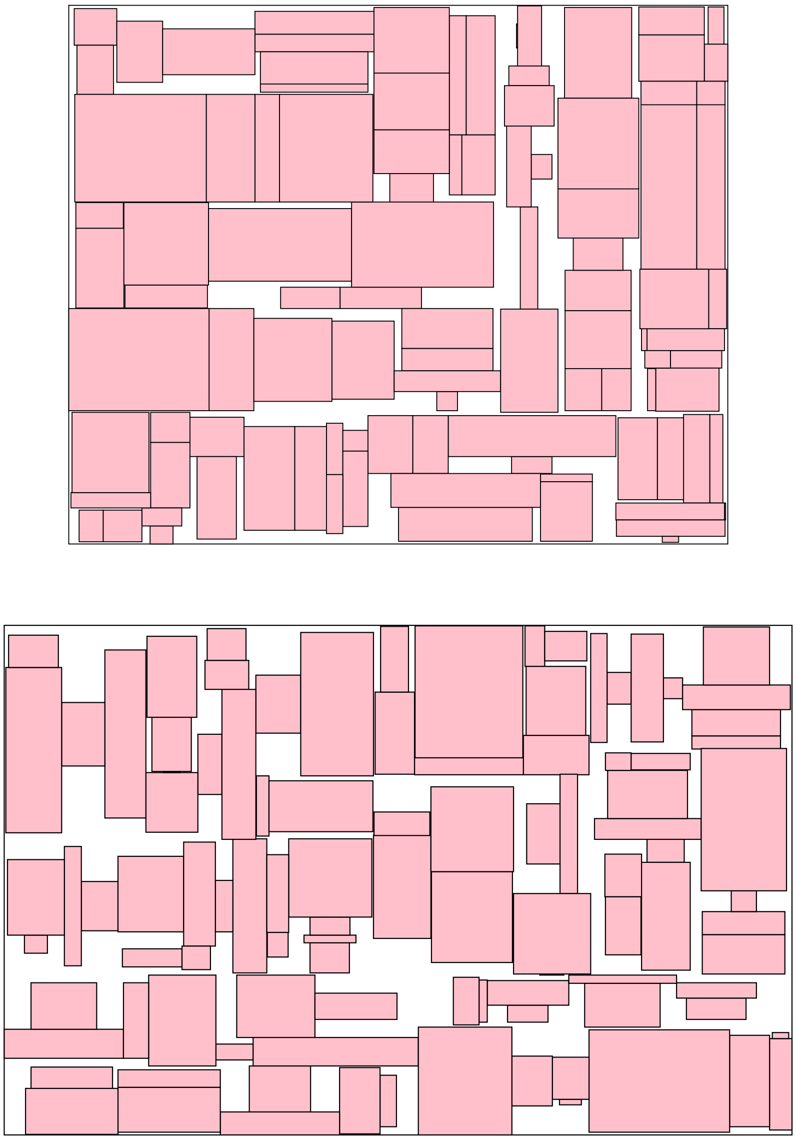

**Figure 6.1:** Packing patterns created without (bottom) and with the use of the iterated matching (top) for an unconstrained packing problem with 100 blocks

Although in this work only constrained packing problems are considered, in figure 6.1 two patterns for an unconstrained packing problem are shown. It ideally visualizes the effect of the iterated matching for the creation of packing patterns. Both patterns consist of the same set of 100 blocks and have been generated with the initialization procedure of the hybrid genetic algorithm for the two-dimensional bin-packing problem, but without any given bin width. Each of the presented patterns is the best pattern encoded in the particular slicing-tree. It that has been created with the use of generalized shape-functions, thus both patterns represent the optimal solutions for the given tree structure. While during creation of the lower pattern the blocks have been randomly paired, the iterated matching heuristic has been used for generation of the tree representing the upper pattern. The function determining the edge weights in the corresponding matching graph was defined by the minimal waste arising in the meta-block for a particular pairing. For the computation of this function, all possible pairings are checked: all combinations of the implementations encoded in the shape-functions are generated with all feasible orientations. The minimal absolute waste of all combinations defines the quality of a pair. Thus, in each iteration a set of pairs is generated with a supposed[1] overall minimal waste.

The lower pattern has an overall waste of 34.1 %, while the pattern generated with the use of the iterated matching heuristic includes only 19.0 % of wasted space. There exist a lot of meta-blocks consisting of four or even more blocks which do not contain any waste at all. This gives a genetic algorithm which uses the iterated matching for the creation of the initial individuals an advantage in comparison to one that starts with a randomly created population. In the first case the population already contains a lot of high quality building blocks. In contrast to that, hardly none of the meta-blocks in the randomly created solution would be included in an optimal packing pattern. However, there is the danger to mislead the optimization process, if due to the matching blocks are combined, that are not paired in the optimal solution. On the one hand, it is a long way for a genetic algorithm to identify good pairings when starting with random creation, on the other hand, breaking up a false pairing, especially if it does not contain any waste inside the resulting meta-block, is very unlikely.

The iterated matching heuristic is not used in the hybrid genetic algorithm for the two-dimensional bin-packing problem, since it is not possible to take into consideration the constraint introduced by the fixed width of the bin.

| **area** | *xerox* | | *ami*33 | | *ami*49 | |
|---|---|---|---|---|---|---|
| $[mm^2]$ | **random** | **match** | **random** | **match** | **random** | **match** |
| **best** | 33.9 | 49.8 | 5.78 | 6.45 | 85.0 | 85.7 |
| **worst** | 76.3 | 52.5 | 16.75 | 9.41 | 226.1 | 115.4 |
| **avg** | 45.5 | 51.0 | 10.16 | 7.51 | 132.2 | 98.0 |
| $\sigma$ | 6.4 | 0.7 | 1.83 | 0.68 | 23.9 | 8.2 |

**Table 6.4:** Statistical data of 1000 generated individuals for the layout generation problem in consideration of the use of the iterated matching heuristic

Table 6.4 shows statistical data for a set of 1000 randomly generated initial layouts for the benchmark VLSI circuits. It is seen that the standard deviation in those cases

[1] The waste is minimal for a particular combination. In the final solution different orientations for the blocks might be chosen, which results in larger waste.

where the individuals are created with the use of the iterated matching is quite small in comparison to this value for the random case. This is because the iterated matching is a deterministic process, that always pairs the same blocks in each individual. Variation is only caused by routing, since in some cases more than one route with the same minimal length for a net exist. Figure 6.2 shows two routes for a two terminal net, which have exactly the same length during the computation of the global routing. As it can be seen, there is no difference for the shape of the marked meta-block combining blocks C, D, and E, although the height of the meta-block consisting of blocks C and D differs in both cases. The height of the marked meta-block is dominated by block E. The total layout area differs in both cases, if due to the route shown in the right picture the routing region between blocks D and F becomes the widest region in the shaded channel and thus determines its width. In that case block E moves out of the original circumscribing rectangle, so that the overall layout area is larger than the area in the first case. Note that the final route in the right case is slightly larger than in the left case. This cannot be foreseen during the computation of the global routing, since the exact channelwidths are determined after having chosen the global routes for all nets.

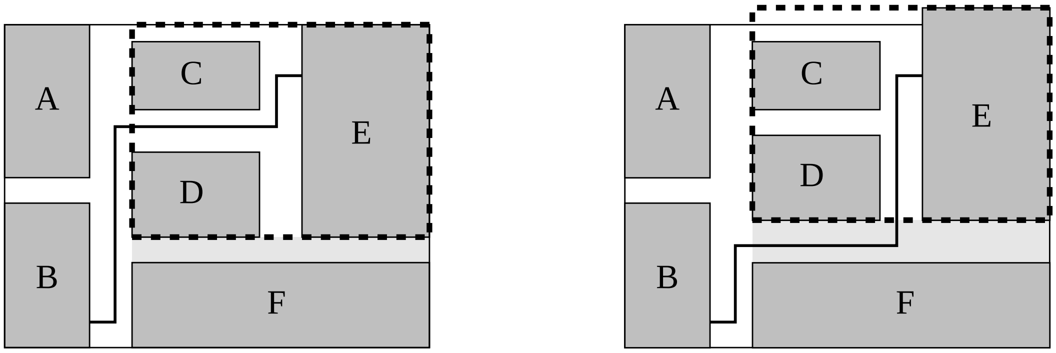


**Figure 6.2:** Two different routes with the same length for a two terminal net

Similar to its ability to generate patterns containing small, high quality building blocks in the unconstrained packing problem described before, the iterated matching ideally supplements the genetic algorithm in case of the layout generation problem, too. When pairing blocks randomly, cells with a high interconnectivity would be arbitrary spread on the layout surface. They have to be arranged during the optimization process in a manner that cells with a high number of common nets are placed next to each other. This can be easily done by the iterated matching based on the number of common nets of two blocks or meta-blocks as described in section 5.4.3.

While the benefit of the use of the iterated matching during creation of the initial individuals is obviously, it can be also used during recombination. Here, a set of subtrees is given, which have to be hierarchically combined in the slicing-tree of the offspring. Figures 6.3 and 6.4 show the convergence velocity for the optimization of the layouts for

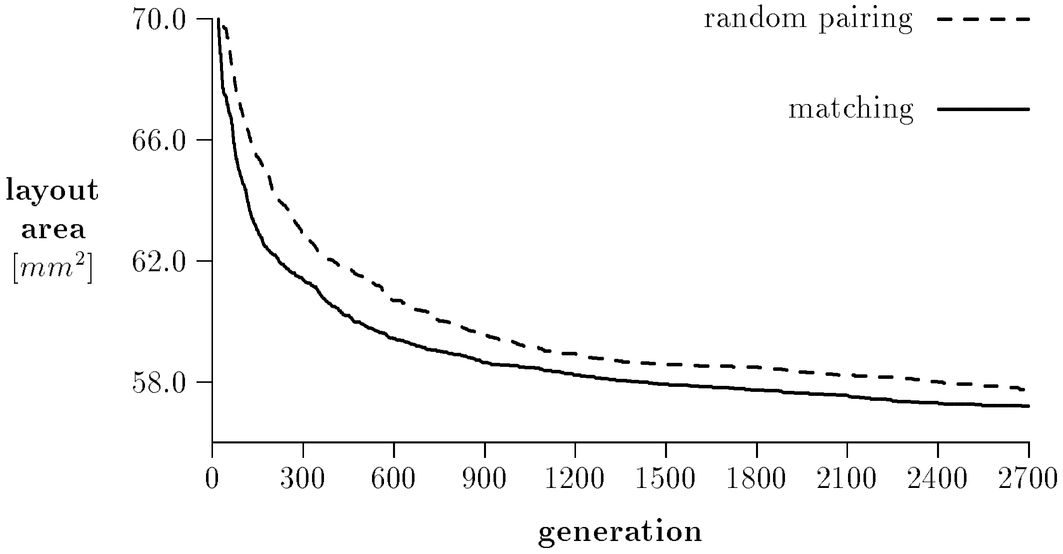


**Figure 6.3:** The performance with respect to the use of the iterated matching heuristic during recombination for the layout generation of circuit *ami*49 (avg. of 10 runs in each case)

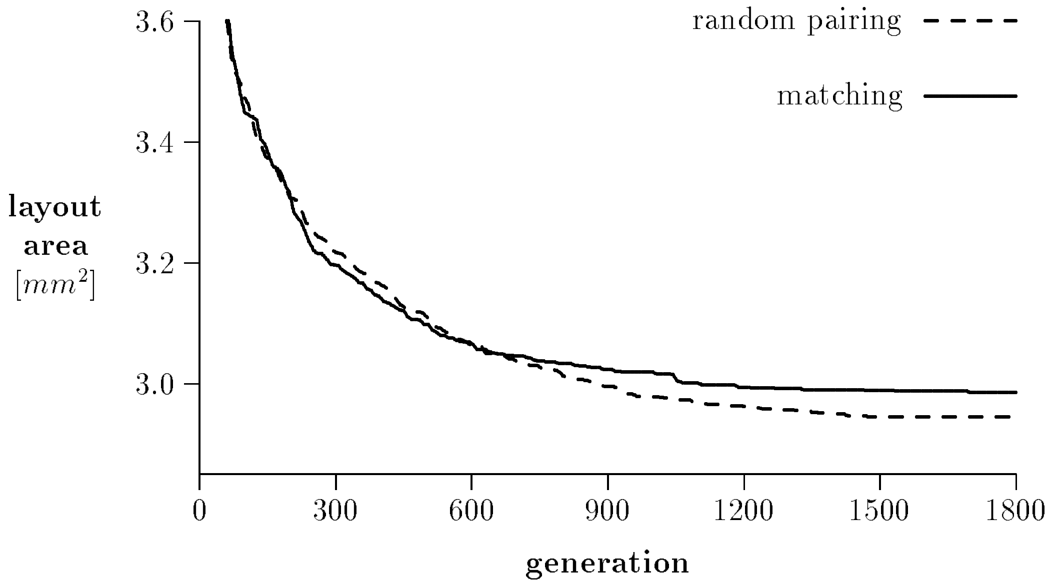


**Figure 6.4:** The performance with respect to the use of the iterated matching heuristic during recombination for the layout generation of circuit *ami*33 (avg. of 10 runs in each case)

two different circuits. To avoid side effects, the recombination and mutation rates have not been adapted during the computation.

Figure 6.3 presents the performance of the genetic algorithm for circuit *ami*49. The dashed curve describes the progress of the layout area for runs of the algorithm without use of the iterated matching during recombination. The solid curve shows the behaviour of the optimization process with use of the iterated matching during recombination. This version shows a better convergence velocity. Thus, for this circuit the iterated matching seems to be beneficial during recombination, too. In comparison to this, for circuit *ami*33 (figure 6.4) the iterated matching is only advantageous in the early stage of optimization. Here, after 600 generations the version of the genetic algorithm that randomly pairs blocks during recombination clearly outperforms the other one. The origin for the different behaviour is based on the number of cells and on the connectivity: circuit *ami*49 not only consists of more cells than *ami*33, but includes more than three times the number of signal-nets (408 to 123 nets, respectively). Furthermore, most signal-nets in circuit *ami*49 only connect two terminals, whereas the average number of terminals for a net in circuit *ami*33 is 3.67.

From these results it can be concluded that it is advantageous to adapt the use of the iterated matching during runtime in the hybrid genetic algorithm for the VLSI layout generation problem. The benefit of this heuristic depends on the number of blocks, the number of generations, and the success of the recombination operator. In the full implementation the use of the iterated matching is adapted with regard to the response, like all other strategy parameters, too.

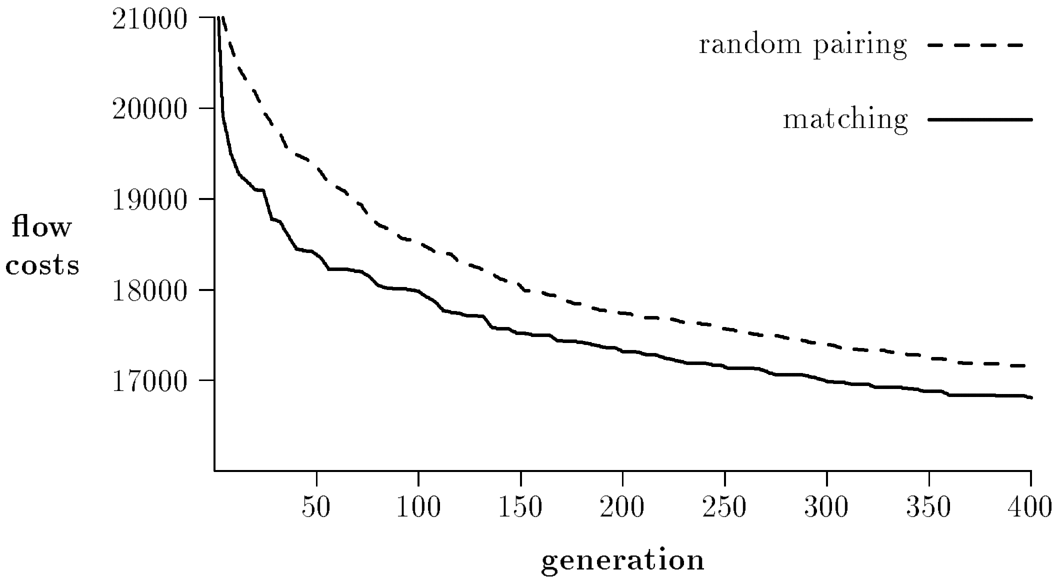


**Figure 6.5:** The performance with respect to the use of the iterated matching heuristic during recombination for the facility layout problem instance TL91-30 (avg. of 10 runs in each case)

For the facility layout problem the iterated matching ideally supports the hybrid genetic algorithm: while during the computation of the tree with use of generalized shape-functions the optimal arrangement of the facilities for this tree is identified, this arrangement is only optimal with regard to the incorporated packing problem. Using the iterated matching based on the flow between two facilities or the sum of flows between comprised facilities, respectively, it iteratively clusters those facilities that have a high flow and thus takes care that these facilities are placed near together. In contrast to the layout generation problem, the iterated matching in the hybrid genetic algorithm for the facility layout problem is beneficial also during recombination, especially for larger problem instances. Figure 6.5 demonstrates this for instance TL91-30. This is evident when comparing the objective functions of both applications. In the layout generation task the routing is already considered when placing the cells and directly influences the layout area, and thus also the fitness of a particular individual. In the hybrid genetic algorithm for the facility layout problem, the layout area is taken as an approximation to the objective function. Without the iterated matching, the search is less efficient, since during the construction of individuals only the packing aspect of the problem is considered. When taking care that facilities with high flow are placed near together, and then identifying a configuration with minimal area, this definitely supports the algorithm to create high quality individuals. Nevertheless, the layout area is a good approximation for the exact flow costs when choosing the final sizes and orientations of the facilities after the construction of the tree. Figure 6.6 demonstrates the correlation between layout area and flow costs for all implementations

| | **time** $[ms]$ | | |
|---|---|---|---|
| | **matching** | **random** | |
| $hp$ | 19.3 | 17.0 | 113.5% |
| $xerox$ | 23.2 | 22.0 | 105.5% |
| $ami33$ | 132.1 | 119.8 | 110.3% |
| $ami49$ | 348.9 | 290.1 | 120.3% |
| TL91-5 | 112.8 | 81.2 | 138.8% |
| TL91-6 | 206.5 | 150.0 | 137.7% |
| TL91-7 | 241.3 | 120.8 | 199.8% |
| TL91-8 | 309.6 | 182.0 | 170.1% |
| TL91-12 | 763.3 | 379.3 | 201.2% |
| TL91-15 | 808.4 | 460.5 | 175.5% |
| TL91-20 | 1330.9 | 859.8 | 154.8% |
| TL91-30 | 2033.7 | 1502.5 | 135.4% |
| x100 | 3062.3 | 2157.5 | 141.9% |

**Table 6.5:** The overhead in computation time due to the iterated matching, 100% represents the computation time with random pairing

encoded in the tree of the best individual for instances TL91-20 and TL91-30. The values are sorted by the layout area, and it can be seen that the layout with minimal area also has the minimal flow costs.

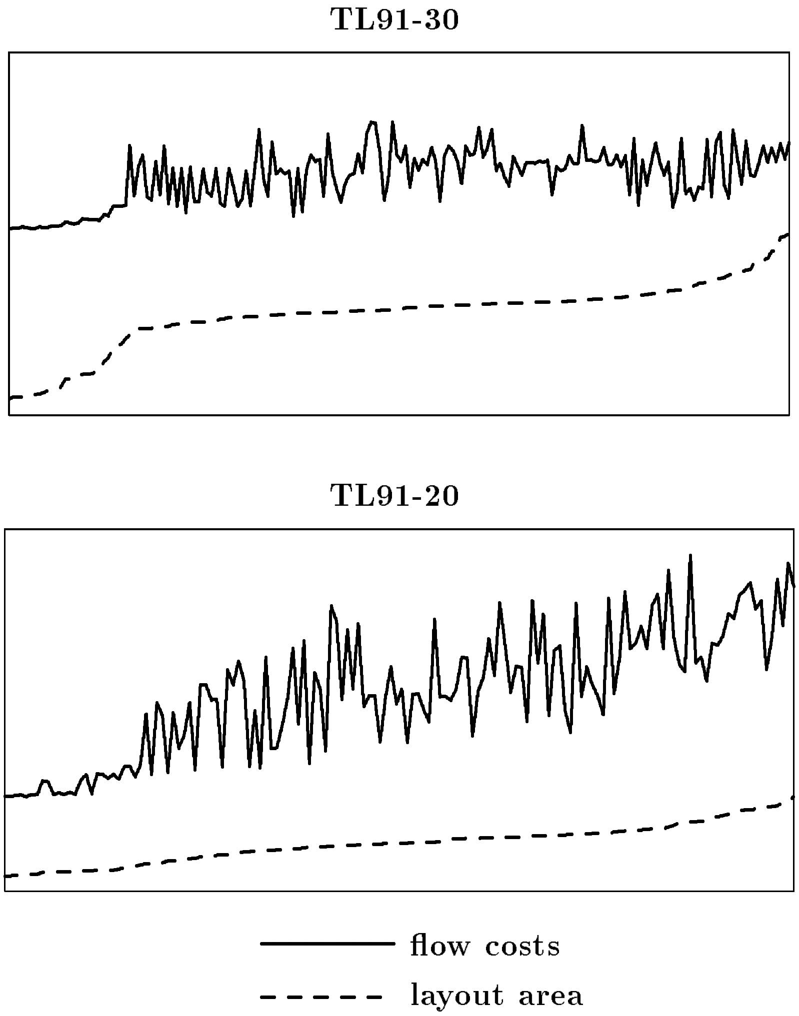


**Figure 6.6:** The layout area is a feasible approximation for the flow costs in the facility layout problem; shown are the sorted layout areas with the corresponding flow costs encoding in the best individual after 1000 evaluations

Since the computation of the matching on a complete graph, like it is done in the current implementation, has cubic runtime complexity, one might accept a higher computation time for individuals created with use of the iterated matching. Table 6.5 compares the computation times for creating initial individuals with and without use of the iterated matching. The overhead seems to be neglectable, even for the larger problems. The last line in the table presents the computation time for the unconstrained packing problem with 100 blocks shown in figure 6.1. In this case the iterated matching starting with 100 blocks in the first iteration has an additional overall computation time of about one second. Still the major part of the computation is done on composing and ordering the different implementations during the construction of the generalized shape-functions.

## 6.4 The Benefit of the Gene-Pool Recombination Operator

The usual recombination operator in genetic algorithms crosses the chromosomes of two parent individuals to generate the chromosomes for one or two offspring. In this work a gene-pool recombination scheme has been introduced: here the genetic information that is represented by subtrees of the parent individuals is inserted into a pool out of which the offspring are created. As described in section 5.1.2, small high quality building blocks in the parents get a higher chance to be included in the offspring. This chance even increases, if the pool contains equal subtrees of more than one individual.

Figure 6.7 shows the different behaviour of the genetic algorithm depending on the recombination scheme. The results are average values of ten runs for both cases. The instance TL91-20 of the facility layout problem was solved and the configuration was the same in both cases apart from the recombination scheme. The five graphs at the left hand side show data obtained by use of a two parent pool for recombination, thus corresponds to the common two parent recombination scheme. The graphs at the right hand side present data for runs that compose a gene-pool of the subtrees of the five best individuals in the current population.

The upper graphs show the progress of the fitness of the best individual. The gene-pool recombination leads to a better convergence velocity, and the final fitness achieved by this variant are superior flow costs of 7430.6 in comparison to 7489.9 for the two parent variant. For the performance of the operators, the average best fitness value achieved by the particular operator in each generation during the adaptation interval is presented. While the progress of the qualities achieved by mutation is almost the same in both cases, the progress of the recombination schemes differs vastly, especially at the later stage of the optimization. In the two parent recombination scheme the quality of the offspring decreases after 150 generations. This lower performance is recognized by the strategy adaptation process, and the recombination rate is turned down in this case. The ranking of strategies during adaptation is based on the achieved response, which is the progress in the average fitness for the 50% best individuals in each subpopulation, measured during the adaptation interval. The course of the response is shown in the lowest graphs in figure 6.7. Note that the curve in case of two parent recombination is much more oscillating than the curve for the gene-pool recombination. At the end the response even gets negative, which points to a more random search process in comparision to the variant that uses the gene-pool recombination scheme.

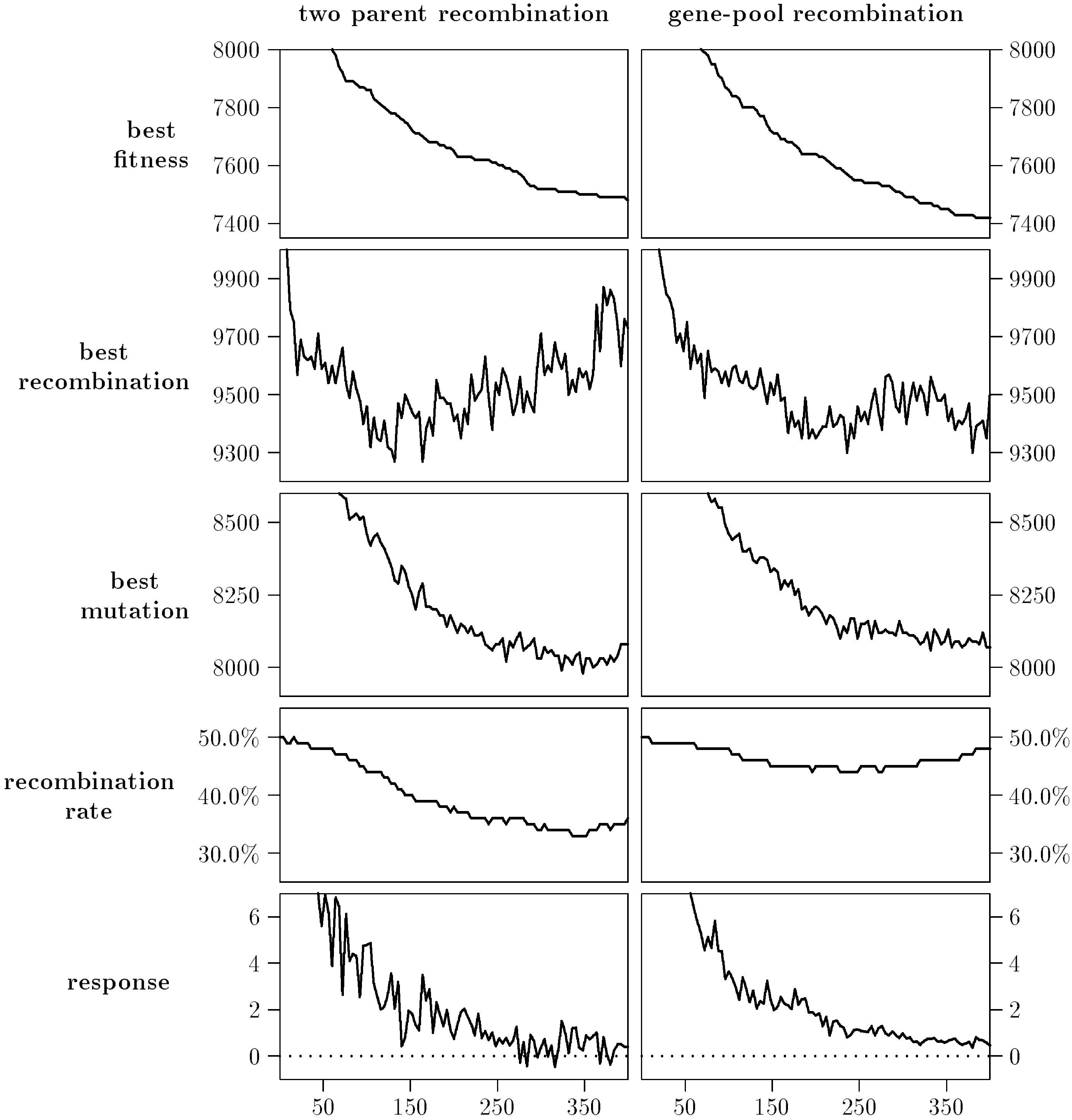


**Figure 6.7:** The different behaviour of the implementation of the hybrid genetic algorithm for facility layout problem TL91-20 depending on the recombination scheme (8 islands, avg. of 10 runs in each case)

## 6.5 The Effect of the Strategy Adaptation

One effect of the strategy adaptation has already been mentioned in the previous section. The adaptation reacts to the lower efficiency of the two parent recombination scheme by preferring strategies with lower recombination rate. This entails a decrease of the overall recombination rate, as shown in figure 6.7.

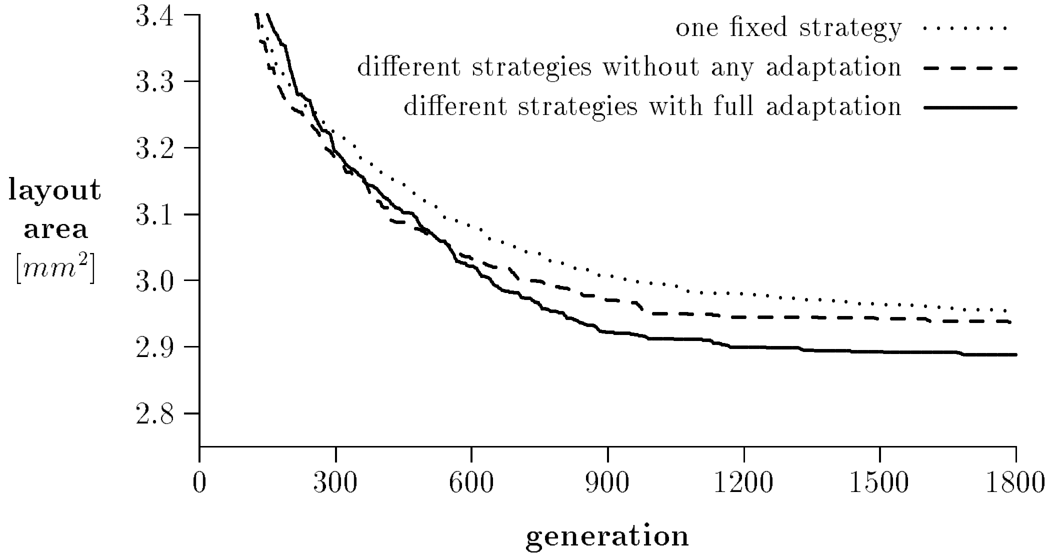


**Figure 6.8:** The performance of the hybrid genetic algorithm for the layout generation for circuit *ami*33 depending on the use of different strategies and adaptation

Figure 6.8 demonstrates the effect of the strategy adaptation during the layout generation for circuit *ami*33. The performance of three different variants of the parallel algorithm is compared. The upper curve shows the progress of the algorithm with subpopulations that are computed in parallel on 16 islands. Migration occurs every three generations, thus it is a full parallel implementation. In contrast to the other variants, all islands pursue the same strategy. The strategy parameters are set to average values of the initial parameter vectors of all islands in the full version. More exactly, crossover and mutation rate are set to 0.5, each mutation operator has the same probability to be chosen, and the iterated matching is used during recombination on half of the islands. The performance of this parallel genetic algorithm is already improved, if the islands use different strategies, as it is visualized by the middle, dashed curve. Although the islands pursue different strategies, it has to be noticed that no adaptation takes place, i. e. the strategy parameters are kept fixed during the whole optimization process. The best performance is obtained by the full version of the adaptive parallel genetic algorithm, as it is shown in the lowest curve. The islands start with the same set of parameters as in the previous case, but in this case the parameters are adapted during the search.

## 6.6 Scalability

From the descriptions of other approaches to solve the constrained packing and placement problems in chapter 3 it can be gathered that the scalability of most methods is quite poor. An example are branch & bound techniques, which due to special heuristics guide the search to the most promising regions of the solution space. For VLSI layout generation a branch & bound approach [OnTaTa91] is superior to all other techniques for the circuits dealt with in this work. However, this approach is impracticable, if a circuit contains flexible cells, which enormously increases the complexity of the optimization problem. For the bin-packing problems, branch & bound based methods only scale up to 10 blocks [KröVor90]. Solving bin-packing instances with up to 60 blocks is impossible due to the fact that there are not any efficient lower bounds for the solution quality. The reviewed approaches to the facility layout problem mostly deal only with up to 10 or 15 facilities, the only approaches that are able to handle up to 30 facilities are the genetic algorithms implemented by Kado [Kado95].

| | **number of facilities** | **full shape-functions** | | **with clipping** | |
|---|---|---|---|---|---|
| | | **time** $[ms]$ | **shapes** | **time** $[ms]$ | **shapes** |
| TL91–5 | 5 | 134.0 | 136.7 | 81.2 | 70.5 |
| TL91–6 | 6 | 287.3 | 187.9 | 150.0 | 90.4 |
| TL91–7 | 7 | 868.5 | 318.0 | 120.8 | 73.1 |
| TL91–8 | 8 | 920.3 | 304.0 | 182.0 | 87.0 |
| TL91–12 | 12 | 4146.0 | 578.5 | 379.3 | 122.1 |
| TL91–15 | 15 | 6640.2 | 823.9 | 460.5 | 122.5 |
| TL91–20 | 20 | 17264.8 | 924.5 | 859.8 | 157.9 |
| TL91–30 | 30 | 53393.5 | 1288.9 | 1502.5 | 193.6 |

**Table 6.6:** Computation time and number of implementations encoded in an individual for the facility layout problem (avg. values of 100 randomly created individuals)

Table 6.6 gives an overview of the scalability of the hybrid genetic algorithm for the facility layout problem. The values of this table are visualized in figure 6.9. As described in section 5.3.1, the given range for the aspect ratio of the facilities is considered by transforming this information into discrete shape-functions. In the implementation of the algorithm a shape-function for a flexible facility encodes 10 shapes. The third and the fourth column of the table and the coordinates marked with a star ($\star$) in the figure represent the poor scalability of the approach, if all non-dominated implementations are stored in each inner node of the tree representing a placement. The number of shapes given is the average number of implementations stored for the root node in the tree. It provides an estimation of the number of different placements encoded in a single individual.

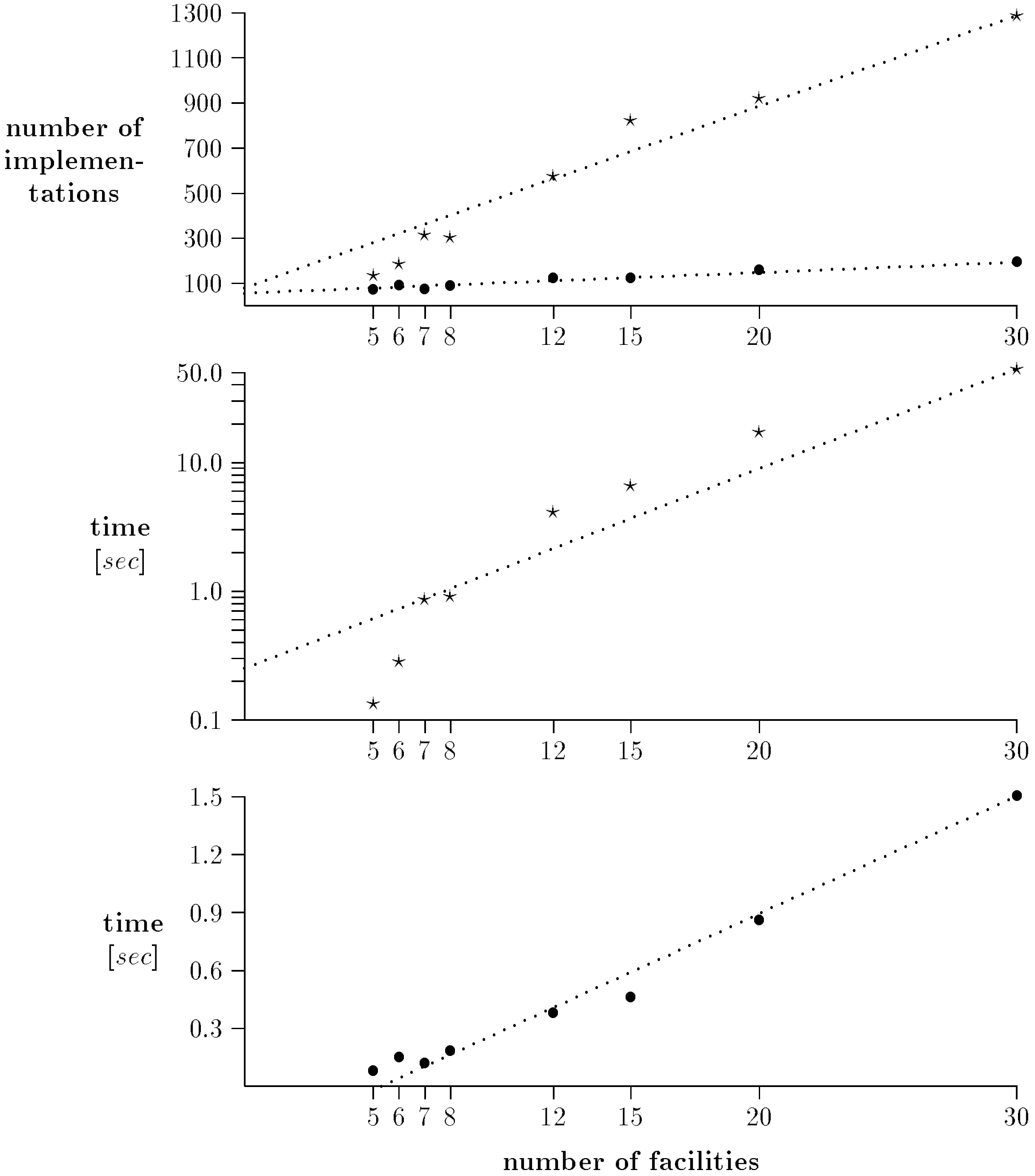


**Figure 6.9:** The number of shapes and the resulting computation time for one fitness evaluation in the hybrid genetic algorithm for the facility layout problem with consideration of all non-dominated implementations in the shape-functions ($\star$) and the corresponding values for clipping implementations ($\bullet$)

| number of blocks | with clipping | | | | full shape-functions | |
|---|---|---|---|---|---|---|
| | bin-width + general | | bin-width | | | |
| | time $[ms]$ | shapes | time $[ms]$ | shapes | time $[ms]$ | shapes |
| 25 | 85.1 | 30.8 | 125.1 | 57.5 | 293.0 | 128.9 |
| 35 | 142.9 | 30.5 | 265.7 | 62.1 | 699.8 | 166.7 |
| 45 | 209.2 | 29.5 | 385.7 | 67.1 | 1251.6 | 209.5 |
| 60 | 321.8 | 25.9 | 610.3 | 60.3 | 3158.9 | 297.1 |

**Table 6.7:** Computation time and number of encoded patterns in a single individual for the two-dimensional bin-packing problem (average of 300 values for each entry)

While the number of implementations grows linearly in the number of facilities, this entails an exponential increase for the corresponding computation times (figure 6.6, middle). The reason for this are the steps needed for computation of a generalized shape-function, which takes into consideration all possible orientations when combining two blocks. In the current implementation, all combinations are generated, sorted, and the dominated implementations are identified and neglected to get the final shape-function for a meta-block.

The exponential increase of evaluation time for an individual can be avoided by clipping some implementations when adding two shape-functions, as described in section 5.3.2 (page 54). If a shape-function encodes more than 30 minimal implementations, then only every second one of these is considered during the computation of the shape-function for the next level of the tree. As it is shown in figure 6.9 (bottom), this results in an only linear increase of computation time.

During solution of the two-dimensional bin-packing problem generalized shape-functions are used, too, to encode all possible orientations for the blocks. Because the blocks have fixed shapes, the increase of the implementations to be stored in a shape-function should not be as extreme as in the facility layout problem. However, the average aspect ratio for a facility is 1.67 in comparison to the average ratio of 4.31 for a block in the instances of the bin-packing benchmark. This implies that the blocks in the bin-packing problems are more elongated in comparison to the shapes a facility can assume. Due to the larger number of blocks and their irregular shapes, without any clipping at all, in one single individual up to 297 different shapes are encoded (table 6.7, last column). In table 6.7 also data for two levels of clipping is presented. The middle columns represent values for the case of only clipping implementations that would lead to solutions with a high waste. Since the corresponding pattern is in one dimension larger than the width of the bin and would additionally lead to a packing height containing more than 50 % waste, it can be neglected. Figure 6.10 presents an example for this case. The left pattern consisting of five blocks will dominate the packing height of any pattern which contains it. Because its height is larger than the width of the bin, it has to be placed upright inside the bin. Thus, from its height and the area of the remaining blocks, the overall waste can be computed.

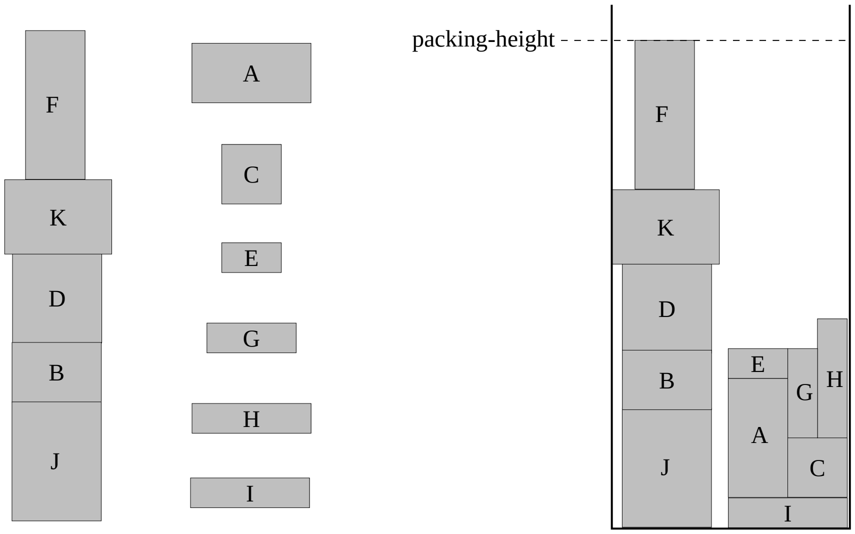


**Figure 6.10:** The influence of a particular subpattern on the overall packing height

The complexity may be further reduced by clipping every second implementation, if the number of encoded shapes is greater than 30, like it is done in the implementation of the hybrid genetic algorithm for the facility layout problem. This clipping is risky, since it might reduce the solution quality: after the initial bottom-up construction of the tree, that particular implementation encoded in the shape-function for the root node is chosen which has a width equal or less than the width of the bin and additionally has the minimal packing height of all feasible implementations. Clipping every second shape at any time during the computation of the tree can reject the optimal pattern for the particular tree structure.

## 6.7 Final Results and Comparison

The implementations of the hybrid genetic algorithms have been tested on a set of benchmarks for each of the three applications. Table 6.8 describes the number of generations and of overall evaluations needed to achieve a given solution quality. Note that the computation has been done in parallel on 16 processors. The average computation times to obtain the final solution quality are for example about 34 and 85 minutes for the layout generation of circuits *ami*33 and *ami*49, respectively. Table 6.9 presents timing data for this application. It is evident that the computation of the global routing consumes more than 60% of the total evaluation time for one individual. The runtimes for the facility layout problems are about 45 and 135 minutes for instances TL91-20 and TL91-30, respectively. In the current implementation, the computation of the generalized shape-functions is done quite straightforward, especially the sorting of all combinations for the shapes needed to

| | **quality above final average solution** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **5%** | | **2%** | | **1%** | | **0.5%** | | **0.0%** | |
| | **gen** | **eval** | **gen** | **eval** | **gen** | **eval** | **gen** | **eval** | **gen** | **eval** |
| *xerox* | 81 | 12960 | 189 | 30240 | 288 | 46080 | 396 | 63360 | 720 | 115200 |
| *hp* | 81 | 12960 | 189 | 30240 | 315 | 50400 | 360 | 57600 | 846 | 135360 |
| *ami*33 | 549 | 87840 | 828 | 132480 | 1053 | 168480 | 1188 | 190080 | 1701 | 272160 |
| *ami*49 | 468 | 74880 | 873 | 139680 | 1143 | 182880 | 1341 | 214560 | 1773 | 283680 |
| r25_1 | 27 | 4320 | 108 | 17280 | 171 | 27360 | 252 | 40320 | 405 | 64800 |
| r25_2 | 18 | 2880 | 72 | 11520 | 99 | 15840 | 171 | 27360 | 297 | 47520 |
| r25_3 | 18 | 2880 | 117 | 18720 | 198 | 31680 | 270 | 43200 | 1125 | 180000 |
| r35_1 | 18 | 2880 | 144 | 23040 | 225 | 36000 | 324 | 51840 | 693 | 110880 |
| r35_2 | 18 | 2880 | 45 | 7200 | 117 | 18720 | 198 | 31680 | 477 | 76320 |
| r35_3 | 45 | 7200 | 171 | 27360 | 243 | 38880 | 279 | 44640 | 405 | 64800 |
| r45_1 | 27 | 4320 | 108 | 17280 | 189 | 30240 | 261 | 41760 | 585 | 93600 |
| r45_2 | 36 | 5760 | 162 | 25920 | 261 | 41760 | 351 | 56160 | 396 | 63360 |
| r45_3 | 54 | 8640 | 144 | 23040 | 270 | 43200 | 324 | 51840 | 432 | 69120 |
| r60_1 | 63 | 10080 | 315 | 50400 | 495 | 79200 | 612 | 97920 | 999 | 159840 |
| r60_2 | 45 | 7200 | 279 | 44640 | 477 | 76320 | 621 | 99360 | 963 | 154080 |
| r60_3 | 54 | 8640 | 207 | 33120 | 279 | 44640 | 495 | 79200 | 846 | 135360 |
| TL91-5 | 8 | 1280 | 12 | 1920 | 16 | 2560 | 20 | 3200 | 28 | 4480 |
| TL91-6 | 12 | 1920 | 16 | 2560 | 20 | 3200 | 24 | 3840 | 48 | 7680 |
| TL91-7 | 8 | 1280 | 12 | 1920 | 16 | 2560 | 20 | 3200 | 80 | 12800 |
| TL91-8 | 24 | 3840 | 36 | 5760 | 60 | 9600 | 72 | 11520 | 160 | 25600 |
| TL91-12 | 56 | 8960 | 116 | 18560 | 152 | 24320 | 180 | 28800 | 200 | 32000 |
| TL91-15 | 76 | 12160 | 132 | 21120 | 144 | 23040 | 160 | 25600 | 200 | 32000 |
| TL91-20 | 100 | 16000 | 152 | 24320 | 172 | 27520 | 188 | 30080 | 200 | 32000 |
| TL91-30 | 112 | 17920 | 256 | 40960 | 308 | 49280 | 348 | 55680 | 400 | 64000 |

**Table 6.8:** The convergence velocity for the full versions of the hybrid genetic algorithms, shown are the number of generations (**gen**) and the total number of evaluations (**eval**) needed in the average case to obtain a particular quality with regard to the final average solution quality (avg. of 10 runs in each case)

| | time $[ms]$ | | |
|---|---|---|---|
| | **total** | **routing** | |
| $hp$ | 17.0 | 10.3 | 60.6% |
| $xerox$ | 22.0 | 13.7 | 62.3% |
| $ami33$ | 119.8 | 72.8 | 60.8% |
| $ami49$ | 290.1 | 191.8 | 66.1% |

**Table 6.9:** The costs of the global routing in the hybrid genetic algorithm for the layout generation problem

identify the dominated implementations is very inefficient and consumes most of the total runtime. Implementing more sophisticated routines for these operations would certainly result in a much better runtime.

The solution quality obtained by the implementations is given in tables 6.10 to 6.12. In these tables there are also previously reported results for the benchmarks presented. These are compared with the results achieved by the hybrid genetic algorithms, and it is efforted to give an explanation for the more or less strong deviation from the results of the other approaches.

| **instance** | **results presented in** [Krög93] | | | **this work** | |
|---|---|---|---|---|---|
| | **best** | **avg. seq. GA** | **avg. parallel GA** | **best** | **average** |
| r25_1 | 109 | 110.8 | 109.4 | 111 | 111.8 |
| r25_2 | 104 | 105.8 | 105.0 | 107 | 107.6 |
| r25_3 | 104 | 106.1 | 104.2 | 105 | 106.8 |
| r35_1 | 152 | 155.8 | 153.0 | 156 | 158.3 |
| r35_2 | 123 | 125.3 | 123.4 | 127 | 127.5 |
| r35_3 | 124 | 126.0 | 124.6 | 127 | 128.4 |
| r45_1 | 196 | 199.3 | 197.0 | 202 | 202.8 |
| r45_2 | 164 | 167.2 | 165.2 | 171 | 172.8 |
| r45_3 | 134 | 137.2 | 135.2 | 139 | 139.2 |
| r60_1 | 253 | 257.6 | 253.8 | 256 | 259.6 |
| r60_2 | 279 | 285.8 | 280.8 | 286 | 288.0 |
| r60_3 | 283 | 288.7 | 284.6 | 292 | 294.6 |

**Table 6.10:** The results for the two-dimensional binpacking-problem

Table 6.10 presents the results obtained for the two-dimensional bin-packing problem. Shown are the best and the average results for the 12 instances proposed by Kröger in his thesis [Krög93]. As it can be gathered from the table, all results are slightly poorer than those reported by him. In his work he uses a genetic algorithm and compares several parallel population models with up to 64 islands or processors and introduces the concept of meta-rectangles to increase the quality of the results. He uses only guillotinable patterns, his genotype encoding is a prefix representation of the corresponding slicing tree. His operators work on the prefix string, he extensively uses a hillclimber to repair and improve each individual after the application of a genetic operator. It is not clear yet why the hybrid algorithm proposed in this work is not competitive with his implementation. Unfortunately, in his work is no hint given on the total number of configurations being checked during optimization. In his implementation the hillclimber mutates each offspring until no improvement is obtained. Nevertheless, the best patterns generated by the hybrid genetic algorithm are presented in the appendix on pages 97 to 106. They already represent compact patterns with an overall waste of 3% to 4% for these difficult instances.

| **Tool** | **Reference** | *xerox* | *hp* | *ami*33 | *ami*49 |
|---|---|---|---|---|---|
| MBP | [UpSaSu90] | 25.79 | — | 2.42 | — |
| TimberWolf | [SwaSec90] | — | — | 2.57 | — |
| BB | [OnTaTa91] | 26.17 | 12.15 | 2.24 | 51.49 |
| SAGA | [EsbMaz94] | 28.52 | 11.81 | — | — |
| FRODO | [Pape95] | 31.13 | — | 3.37 | 60.02 |
| CAR | [Pape95] | 28.71 | — | 2.64 | 56.40 |
| **this work** | **best** | 27.77 | 12.96 | 2.81 | 55.65 |
| | **average** | 29.31 | 13.16 | 2.91 | 57.12 |

**Table 6.11:** The comparison of the results for VLSI layout generation

Table 6.11 demonstrates the results for the VLSI layout generation. The best results are reported by Onodera *et al.* [OnTaTa91]. They use a branch & bound method to place the cells. In addition to the rotation of cells during placement, they use a reflection operator to generate different implementations for a single cell. Their branch & bound approach scales up to placing 6 cells, for larger instances the layout has to be hierarchically composed. In particular, for circuit *ami*49 they need two levels of hierarchy. TimberWolf by Swartz and Sechen [SwaSec90] and MBP by Upton *et al.* [UpSaSu90] are based on simulated annealing. SAGA by Esbensen and Mazumder [EsbMaz94] is a mixture of a genetic algorithm and simulated annealing. It starts as a genetic algorithm and gradually switches to a simulated annealing process by reducing population size and increasing the mutation rate. Due to complexity constraints, their approach was not able to optimize larger circuits than *xerox* and *hp*. FRODO is a floorplanning tool which is based on the work of Müller [Müll90, LenMül93]. The reported results are from the thesis of Pape [Pape95], who uses the placements generated by FRODO as an input to his tool CAR, which refines and topologically compacts the layout.

Although the same set of benchmark circuits was treated in all previously mentioned approaches, the comparison of the results is more complicated. The result of TimberWolf have been achieved by combining several tools, BB and SAGA only determine a placement with estimating the routing space, the presented results have been produced by using other (commercial) tools for routing and compaction after placement. FRODO only does floorplanning, and CAR starts its work with a given placement. So there is always a strict distinction between placement and routing in all these approaches. The hybrid genetic algorithm proposed and implemented in this work fully integrates the computation of the global routing into the placement. The results have been exclusively obtained by the use of this algorithm. The routability of the layouts has been checked by a simple tool developed at the University of Osnabrück, which also produced the layouts presented on pages 107 to 109 in the appendix. Doing some compaction or using a special routing tool like the other approaches have done to obtain the results presented in table 6.11, would certainly produce better results. Alternatively, a sophisticated heuristic can be included to determine the demand of routing space inside the channels more accurately. Especially, in the layout for circuit *xerox* (page 107) exists much wasted space inside some channels due to the addition of too many tracks.

| | **previous work** | | **this work** | |
|---|---|---|---|---|
| | [TamLi91] | [KaRoCo95] | **best result** | **avg. result** |
| TL91–5 | 247 | 228 | 205 | 205 |
| TL91–6 | 514 | 362 | 317 | 317 |
| TL91–7 | 559 | 559 | 518 | 519 |
| TL91–8 | 839 | 839 | 705 | 709 |
| TL91–12 | 3162 | 3162 | 1954 | 1993 |
| TL91–15 | 5862 | 5862 | 3849 | 3923 |
| TL91–20 | — | 16535 | 7536 | 7777 |
| TL91–30 | — | 42814 | 16479 | 16876 |

**Table 6.12:** The results for the facility layout problem

In table 6.12 the obtained results for the instances of the facility layout problem are presented. For comparison the results of Tam and Li [TamLi91] and of Kado *et al.* [KaRoCo95] are shown. In the article of Tam and Li this benchmark set has been introduced. They describe a multi stage approach, where the facilities are approximated by circles, which are placed with regard to the flow costs. In a later step these circles are transformed into rectangular shaped facilities. Their approach is only scalable up to 15 facilities. They propose a hierarchical decomposition for larger instances, but they do not present final results for problems TL91-20 and TL91-30. Kado *et al.* implemented a set of genetic algorithms based on a slicing-tree representation of the layouts. Although they have been able to solve all instances of the benchmark set and already obtained better results for some smaller instances than Tam and Li, their results are not competitive with the results achieved by the implementation of the hybrid genetic algorithm introduced in this work.

The major drawback of the other approaches is that they do not deal with sizing and placing of the facilities at the same time. In the hybrid genetic algorithm, several implementations for the facilities are taken into account during the creation of a tree due to the use of shape-functions. The flow between the facilities is considered by use of the iterated matching, which ensures that facilities with high flow are placed next to each other. The obtained accuracy of the overall solution quality depends on the number of implementations chosen for the facilities during discretization of the shapes, and on the clipping mode. Clipping of shape-functions has been proven to be very useful to keep the computation times tractable. The hybrid genetic algorithm for the facility layout problem is highly scalable. The layout area is used as an approximation for the flow costs during the optimization. The obtained layouts shown on pages 110 to 113 in the appendix are very compact in comparison to the layouts presented in [TamLi91] or in the thesis of Kado [Kado95]. Although only the simple facility layout problem has been treated, the approach can be easily extended to deal with constraints for the shape of the planar site the facilities have to be placed on, or the inclusion of dead-space ratios.

# Chapter 7

# Conclusion

In this work hybrid genetic algorithms for solving constrained packing and placement problems have been presented. The main feature of these algorithms is the genotype representation of a pattern by a binary tree. The genetic operators directly work on this tree structure. Mutation is done by exchanging or moving subtrees inside the tree of an individual. For recombination a multi-parent scheme called gene-pool recombination has been introduced. All subtrees contained in the better individuals of the current population are collected and are used to build a pool of the genetic information of these individuals. Disjunct subtrees are stochastically drawn out of this pool and are combined to complete trees representing the genotypes of the offspring.

The parallelization of the genetic algorithms is based on the stepping stone model. Each subpopulation located on a single processor is optimized by a sequential hybrid genetic algorithm. At fixed intervals individuals migrate between neighboured islands. All algorithms on the islands pursue different strategies, which are adapted during the search. The proposed adaptation scheme periodically ranks the parameter vectors of the strategies. This ranking is based on the average response to selection achieved during a fixed adaptation interval. The values of each parameter vector are adapted by assimilating them to the corresponding values of the next better strategy. The outstanding values of the best of all strategies are further extended.

All algorithms exploit hybrid techniques for implicit hillclimbing during the creation of new individuals. The main concept used are the shape-functions, a technique from VLSI physical design. In this work the notion of a shape-function has been generalized to encode orientations for blocks composed in a pattern. During the creation of an individual all feasible orientations for the blocks are taken into account by encoding them into the shape-function. Since the root node contains a shape-function, several patterns are encoded in a single individual. Out of this set the pattern with minimal area can be directly identified. This represents the optimal pattern according to the particular tree structure.

For two of the applications dealt with in this work the constraints do not directly influence the packing aspect of the problems. Another heuristic is used to take this into account. The iterated matching successively clusters blocks with regard to that constraint during creation of the trees. While the iterated matching turned out to be very beneficial for larger problem instances and during the initial stage of the optimization, it contains the risk of leading to premature convergence since it is a deterministic process. This is by-passed by adapting the use of the iterated matching during the search.

The constrained packing and placement problems which have been optimized by the implementations of the hybrid genetic algorithms are the two-dimensional bin-packing, the VLSI macro cell layout generation, and the facility layout problem. While the constraint in the bin-packing problem is the introduction of a maximal width of the packing pattern, the constraints in the other problems are based on quantified dependencies between the blocks. In the layout generation task a set of cells has to be placed, and interconnection wirings have to be routed on the space between them. While the objective is still a minimal area layout, the size of the layout is mainly influenced by the routing. Due to complexity constraints, usually only an estimated amount of routing space is provided around the cells during placement. The computation of the routes is done in a following phase. The hybrid genetic algorithm for the layout generation, which has been described in this work, fully integrates the determination of the global routes into the computation of the placement. This is possible due to the use of the binary slicing-tree representation of a placement, which keeps the positions of the cells flexible, until the computation of the global routing is completed. The results obtained for a set of benchmark circuits are competitive with results presented in literature. These results can be definitely improved by inclusion of a sophisticated detailed routing procedure.

The best results are obtained for the last application, the facility layout problem. This is similar to the layout generation task with regard to the constraint, but here the objective is not a minimal area layout. A sum of flows between the facilities has to be minimized. It has been shown that the layout area is still a good approximation for this objective. The main complexity of the facility layout problem is not the constraint added by the flow between facilities, it is the high degree of freedom introduced by their flexible shapes. In common approaches to this problem the rectangular facilities are approximated by circles, and a placement is computed with respect to the flows between their centres. Afterwards this placement is refined by choosing feasible rectangular shapes for all facilities. In the hybrid genetic algorithm the burden resulting from the flexibility of the facilities is handled by discretization and the use of the generalized shape-functions during the creation of the tree. This approach has been shown to be very scalable and produces vastly better results than all known techniques, especially for larger instances.

The problems considered in this work are either real-world optimization problems or represent variants of significant real-world optimization problems. While the proposed algorithm for the layout generation task adds too much routing space, since it uses one track for each net in a routing region, it can be certainly extended to become competitive to the best known tools. The strength of this approach, in comparison to others, is the fact that the global routing is completely integrated into the placement process, and that it deals with flexible cells by encoding their feasible implementations into the shape-functions during the construction of the tree.

Although the concepts of the implicit hillclimbing and gene-pool recombination are well matched with the genotype representation as a binary tree, they are applicable to all problems where a hierarchical representation of the candidate solutions is possible. The implicit hillclimbing eliminates useless combinations of partial solutions quite early during the construction of an individual, thus the final genotype only encodes a set of high quality neighboured solutions. In contrast to this, a conventional hillclimber blindly checks neighboured configurations.

It has been shown that it is quite useful to hybridize genetic algorithms. Although designing problem-specific encodings and operators is a hard job and needs specialized knowledge of the specific application, it is especially for combinatorial problems often the only way to get them tractable by a genetic algorithm. In theory it is a nice dream to think of genetic algorithms as a generally applicable optimization tool, which can solve every problem after encoding it in a bit-string. In practice a considerable amount of preliminary work has to be done to tailor the algorithm to a specific problem. The genetic algorithms do not really have magic power, but with some tricks you might enable them to give this impression.

# Bibliography


[ACSED] Automatic Control and Systems Engineering Department, University of Sheffield, `http://www.shef.ac.uk/uni/projects/gaipp/index.html`

[AnJoKi82] K. J. Antreich, F. M. Johannes, F. H. Kirsch, *A new approach for solving the placement problem using force models*, Proceedings IEEE Int. Symposium on Circuits and Systems, 1982, 481–486

[AS] Research Group Adaptive Systems, GMD, St. Augustin, H. Mühlenbein, `http://borneo.gmd.de/AS/pages/as.html`

[Bäck91] Th. Bäck, *Self-Adaptation in Genetic Algorithms*, Procs. of the 1st European Conference on Artificial Life, F.J. Varela, P. Bourgine (eds.), MIT Press, 1992, 263–271

[Bäck94] Th. Bäck, *Parallel Optimization of Evolutionary Algorithms*, Procs. 3rd Conf. on Parallel Problem Solving from Nature (PPSN III), Y. Davidor, H.-P. Schwefel, R. Männer (eds.), Springer LNCS 866, 1994, 418–427

[Bäck96] Th. Bäck, *Evolutionary Algorithms in Theory and Practice*, Oxford University Press, 1996

[BäHoSc91] Th. Bäck, F. Hoffmeister, H.-P. Schwefel, *A survey of Evolution Strategies*, Procs. of the 4th Int. Conf. on Genetic Algorithms (ICGA), R. K. Belew, L. B. Booker (eds), Morgan Kaufmann Publ., 1991, 2–9

[BäcSch93] Th. Bäck, H.-P. Schwefel, *An overview of evolutionary algorithms for parameter optimization*, Evolutionary Computation, Vol. 1, 1993, 1–23

[BerSer92] H. Bersini, G. Seront, *In Search of a good Evolution–Optimization Crossover*, Procs. 2nd Conf. on Parallel Problem Solving from Nature (PPSN II), R. Männer, B. Manderick (eds.), Elsevier Science Publishers, 1992, 479–488

[BIONIK] Institut für Prozeß und Anlagentechnik, Arbeitsgruppe Bionik und Evolutionstechnik, TU Berlin, `http://lautaro.fb10.tu-berlin.de`

[Breu77] M. A. Breuer, *Min-Cut Placement*, Journal of Design Automation and Fault-Tolerant Computing, Vol. 1, No. 4, 1977, 343–382

[CBL] CBL, *CAD Benchmarking Laboratory, North Carolina State University*, `http://www.cbl.ncsu.edu/www/CBL_Home.html`

[ChMaSh91] H. Chan, P. Mazumder, K. Shahookar, *Macro-cell and module placement by genetic adaptive search with bitmap-represented chromosome*, Integration, the VLSI journal, Vol. 12, No. 1, 1991, 49–77

[COGS] Evolutionary and Adaptive Systems (EASy) group, School of Cognitive and Computing Sciences (COGS) in the University of Sussex at Brighton, U. K., `http://www.cogs.susx.ac.uk/lab/adapt/index.html`

[CoHeMaRi91] J. P. Cohoon, S. U. Hegde, W. N. Martin, D. S. Richards, *Distributed Genetic Algorithms for the Floorplan Design Problem*, IEEE Trans. on CAD, Vol. 10, No. 4, April 1991, 483–492

[CohPar86] J. P. Cohoon, W. D. Paris, *Genetic Placement*, Procs. of IEEE Int. Conf. on CAD 1986, 422–425

[Darw59] C. Darwin, *The Origin of Species by Means of Natural Selection*, 1859

[Davi91] L. Davis, *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, New York, 1991

[DaAlDi95] V. P. Daza, A. G. de Alvarenga, J. de Diego, *Exact solutions for constrained two-dimensional cutting problems*, European Journal of Operational Research, Vol. 84, 1995, 633-644

[DeJo75] K. A. De Jong, *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*, PhD thesis, University of Michigan, 1975

[DunKer85] A. E. Dunlop, B. W. Kernighan, *A procedure for placement of standard cell VLSI layouts*, IEEE Trans. on CAD, Vol. 4, No. 1, Jan. 1985, 92–98

[Edinb] Evolutionary Computation Group, Department of Artificial Intelligence, University of Edinburgh, `http://www.dai.ed.ac.uk`

[EiRaRu94] A. E. Eiben, P.-E. Raué, Z. Ruttkay, *Genetic algorithms with multi-parent recombination*, Procs. 3rd Conf. on Parallel Problem Solving from Nature (PPSN III), Y. Davidor, H.-P. Schwefel, R. Männer (eds.), Springer LNCS 866, 1994, 78–87

[Esbe92] H. Esbensen, *A Genetic Algorithm for Macro Cell Placement*, Procs. of the European Design Automation Conference, 1992, 52–57

[EsbMaz94] H. Esbensen, P. Mazumder, *SAGA: A Unification of the Genetic Algorithm with Simulated Annealing and its Application to Macro-Cell Placement*, Procs. of the 7th Int. Conf. on VLSI Design, 1994, 211–214

[Falc81] D. S. Falconer, *Introduction to Quantitative Genetics*, Longman, London, 1981

[Foga89] T. C. Fogarty, *Varying the Probability of Mutation in the Genetic Algorithm*, Procs. of the 3rd Int. Conf. on Genetic Algorithms (ICGA), J. D. Schaffer (ed.), Morgan Kaufmann Publ., 1989, 104–109

[FreHär93] B. Freisleben, M. Härtfelder, *Optimization of Genetic Algorithms by Genetic Algorithms*, Artificial Neural Nets and Genetic Algorithms, R. F. Albrecht, C. R. Reeves, N. C. Steele (eds.), Springer Verlag, 1993, 392–399

[FreMer96] B. Freisleben, P. Merz, *New Genetic Local Search Operators for the Traveling Salesman Problem*, Procs. 4th Conf. on Parallel Problem Solving from Nature (PPSN IV), H.-M. Voigt, W. Ebeling, I. Rechenberg, H.-P. Schwefel (eds.), Springer LNCS 1141, 1996, 890–899

[FriVor94] A. Fritsch, O. Vornberger, *Cutting Stock by Iterated Matching*, Operations Research Proceedings, Selected Papers of the Int. Conf. on OR 94, U. Derigs, A. Bachem, A. Drexl (eds.), Springer Verlag, 1995, 92–97

[GAG] Genetic Algorithms Group (GAG), George Mason University, Fairfax, Virginia, `http://www.cs.gmu.edu/research/gag/`

[GARAGe] Genetic Algorithms Research and Applications Group, Michigan State University, `http://isl.msu.edu/GA/`

[GarJoh79] M. R. Garey, D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, CA, 1979

[Gold89] D. E. Goldberg, *Genetic Algorithms in Search Optimization & Machine Learning*, Addison-Wesley, 1989

[GolDeb91] D. E. Goldberg, K. Deb, *A comparative analysis of selection schemes used in genetic algorithms*, Foundations of Genetic Algorithms, G. Rawlins (ed.), Morgan Kaufmann, 1991

[GoTeKu94] E. D. Goodman, A. Y. Tetelbaum, V. M. Kureichik, *A Genetic Algorithm Approach to Compaction, Bin Packing, and Nesting Problems*, Technical Report 940702, Case Center for Computer-Aided Engineering and Manufacturing, Michigan State University, 1994

[Gorg92] M. Gorges-Schleuter, *Comparison of Local Mating Strategies in Massively Parallel Genetic Algorithms*, Procs. 2nd Conf. on Parallel Problem Solving from Nature (PPSN II), R. Männer, B. Manderick (eds.), Elsevier Science Publishers, 1992, 479–488

[Gref86] J. J. Grefenstette, *Optimization of Control Parameters for Genetic Algorithms*, IEEE Trans. on Systems, Man, and Cybernetics, Vol. 16, No. 1, 1986, 122–128

[HiMiPe96] R. Hinterding, Z. Michalewicz, T. C. Peachey, *Self-Adaptive Genetic Algorithm for Numeric Functions*, Procs. 4th Conf. on Parallel Problem Solving from Nature (PPSN IV), H.-M. Voigt, W. Ebeling, I. Rechenberg, H.-P. Schwefel (eds.), Springer LNCS 1141, 1996, 420–429

[Holl92] J. H. Holland, *Adaptation in Natural and Artificial Systems*, 1992 Edition, MIT Press, 1992

[IlliGAL] Illinois Genetic Algorithm Laboratory, *IlliGAL*, University of Illinois at Urbana-Champaign, `http://gal4.ge.uiuc.edu`

[Juls95] B. A. Julstrom, *What Have You Done for Me Lately? Adapting Operator Probabilities in a Steady-State Genetic Algorithm*, Procs. of the 6th Int. Conf. on Genetic Algorithms (ICGA), L. J. Eshelman (ed.), Morgan Kaufmann Publ., 1995, 81–87

[Kado95] K. Kado, *An Investigation of Genetic Algorithms for Facility Layout Problems*, Master Thesis, University of Edinburgh, Department of Artificial Intelligence, 1995

[KaRoCo95] K. Kado, P. Ross, D. Corne, *A Study of Genetic Algorithm Hybrids for Facility Layout Problems*, Procs. of the 6th Int. Conf. on Genetic Algorithms (ICGA), L. J. Eshelman (ed.), Morgan Kaufmann Publ., 1995, 498–505

[Koza92] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*, MIT Press, 1992

[Krög93] B. Kröger, *Parallele Genetische Algorithmen zur Lösung eines zweidimensionalen Bin Packing Problems*, PhD thesis, University of Osnabrück, Department of Mathematics/Computer Science, 1993

[Krög95] B. Kröger, *Guillotineable binpacking: A genetic approach*, European Journal of Operational Research, Vol. 84, No. 3, 1995, 645–661

[KröVor90] B. Kröger, O. Vornberger, *A Parallel Branch & Bound Approach for Solving a two-dimensional Cutting Stock Problem*, Report No. 25/1990, University of Osnabrück, Department of Mathematics/Computer Science, 1990

[Lauxt96] F. Lauxtermann, *Traveling Salesman Problem mit Iteriertem Matching*, Diploma Thesis, University of Osnabrück, Department of Mathematics/Computer Science, 1996

[Leng90] Th. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*, John Wiley & Sons, 1990

[LenMül87] Th. Lengauer, R. Müller, *The Complexity of Floorplanning based on Binary Circuit Partitions*, Technical Report No. 46, University of Paderborn, Department of Mathematics and Computer Science, 1987

[LenMül93] Th. Lengauer, R. Müller, *Robust and Accurate Hierarchical Floorplanning with Integrated Global Wiring*, IEEE Trans. on CAD, Vol. 12, No. 6, June 1993, 802–809

[Mitc96] M. Mitchell, *An Introduction to Genetic Algorithms*, MIT Press, 1996

[MoArAr92] R. N. Morabito, M. N. Arenales, V. F. Arcaro, *An and–or-graph approach for two-dimensional cutting problems*, European Journal of Operations Research, Vol. 58, 1992, 263–271

[MelGau96] R. D. Meller, K.-Y. Gau, *The Facility Layout Problem: Recent Trends and Perspectives*, Journal of Manufacturing Systems, Vol. 15, No. 5, 1996, 351–366

[Mühl89] H. Mühlenbein, *Parallel Genetic Algorithms, Population Genetics and Combinatorial Optimization*, Procs. of the 3rd Int. Conf. on Genetic Algorithms (ICGA), J. D. Schaffer (ed.), Morgan Kaufmann Publ., 1989, 416–421

[Mühl91] H. Mühlenbein, *Evolution in Time and Space — The Parallel Genetic Algorithm*, Foundations of Genetic Algorithms, G. J. E. Rawlins (ed.), Morgan Kaufmann Publ., 1991, 316–337

[MühSch94] H. Mühlenbein, D. Schlierkamp-Voosen, *The science of breeding and its application to the breeder genetic algorithm BGA*, Evolutionary Computation, Vol. 1, No. 4, 1994, 335–360

[MühVoi95] H. Mühlenbein, H.-M. Voigt, *Gene Pool Recombination in Genetic Algorithms*, Procs. of the Metaheuristics Int. Conf., I. H. Osman, J. P. Kelly (eds.), Kluwer Academic Publishers, Norwell, 1995

[Müll90] R. Müller, *Hierarchisches Floorplanning mit integrierter globaler Verdrahtung*, PhD Thesis, Department of Mathematics and Computer Science, University of Paderborn, 1990

[NCARAI] Navy Center for Applied Research in Artificial Intelligence, Washington, D. C., `http://www.aic.nrl.navy.mil`

[MCNC] North Carolina's Microelectronics, Computing and Networking Center, North Carolina, `http://www.mcnc.org`

[OliFer90] J. F. Oliveira, J. S. Ferreira, *An improved version of Wang's algorithm for two-dimensional cutting problems*, European Journal of Operational Research, Vol. 44, 1990, 256–266

[OnTaTa91] H. Onodera, Y. Taniguchi, K. Tamaru, *Branch-and-Bound Placement for Building Block Layout*, Procs. 28th ACM/IEEE Design Automation Conference, 1991, 433–439

[Otte83] R. Otten, *Efficient Floorplan Optimization*, Procs. of Int. Conf. on Comp. Design 1983, 499–502

[PC2] Paderborn Center for Parallel Computing ($PC^2$), University of Paderborn, Germany, `http://www.uni-paderborn.de/pcpc/pcpc.html`

[Pape95] M. Pape, *Chip Assembly mit Topologischer Kompaktierung*, PhD Thesis, Department of Mathematics and Computer Science, University of Paderborn, 1995

[PaReWo94] P. M. Pardalos, F. Rendl, H. Wolkowics, *The quadratic assignment problem: A survey and recent developments*, DIMACS Series Discr. Math. Theor. Comp. Science 16, 1994, 1-42

[Rech94] I. Rechenberg, *Evolutionsstrategie '94*, Frommann-Holzboog, Stuttgart, 1994

[SaiYou95] S. M. Sait, H. Youssef, *VLSI Physical Design Automation: Theory and Practice*, McGraw-Hill, 1995

[SantaFe] Santa Fe Institute, New Mexico, `http://www.santafe.edu`

[SchMüh94] D. Schlierkamp-Voosen, H. Mühlenbein, *Strategy Adaptation by Competing Subpopulations*, Procs. 3rd Conf. on Parallel Problem Solving from Nature (PPSN III), Y. Davidor, H.-P. Schwefel, R. Männer (eds.), Springer LNCS 866, 1994, 199–208

[SchMüh96] D. Schlierkamp-Voosen, H. Mühlenbein, *Adaptation of Population Sizes by Competing Subpopulations*, Procs. IEEE Int. Conf. on Evolutionary Computation (ICEC '96), IEEE Press, 1996, 330–335

[SchVor95] V. Schnecke, O. Vornberger, *Genetic Design of VLSI-Layouts*, Procs. First IEE/IEEE Int. Conf. on GAs in Engineering Systems: Innovations and Applications, GALESIA '95, IEE Conference Publication No. 414, 1995, 430–435

[SchVor96a] V. Schnecke, O. Vornberger, *A Genetic Algorithm for VLSI Physical Design Automation*, Procs. 2nd. Int. Conf. on Adaptive Computing in Engineering Design and Control, ACEDC '96, University of Plymouth, I. C. Parmee (ed.), 1996, 53–58

[SchVor96b] V. Schnecke, O. Vornberger, *An Adaptive Parallel Genetic Algorithm for VLSI-Layout Optimization*, Procs. 4th Conf. on Parallel Problem Solving from Nature (PPSN IV), H.-M. Voigt, W. Ebeling, I. Rechenberg, H.-P. Schwefel (eds.), Springer LNCS 1141, 1996, 859–868

[Schw95] H.-P. Schwefel, *Evolution and Optimum Seeking*, John Wiley & Sons, New York, 1995

[Sech85] C. Sechen, A. Sangiovanni-Vincentelli, *The timberwolf placement and routing package*, IEEE Journal of Solid-State Circuits, Vol. SC-20, 1985, 510–522

[ShaMaz91] K. Shahookar, P. Mazumder, *VLSI Cell Placement Techniques*, ACM Computing Surveys, Vol. 23., No. 2, June 1991, 143–220

[Sher93] N. Sherwani, *Algorithms for VLSI Physical Design Automation*, Kluwer Academic Publishers, 1993

[SmiFog96a] J. Smith, T. C. Fogarty, *Self Adaptation of Mutation Rates in a Steady State Genetic Algorithm*, Procs. IEEE Int. Conf. on Evolutionary Computation (ICEC '96), IEEE Press, 1996, 318–323

[SmiFog96b] J. Smith, T. C. Fogarty, *Adaptively Parameterised Evolutionary Systems: Self Adaptive Recombinations and Mutation in a Genetic Algorithm*, Procs. 4th Conf. on Parallel Problem Solving from Nature (PPSN IV), H.-M. Voigt, W. Ebeling, I. Rechenberg, H.-P. Schwefel (eds.), Springer LNCS 1141, 1996, 859–868

[Stoc83] L. Stockmeyer, *Optimal Orientations of Cells in Slicing Floorplan Designs*, Information and Control, Vol. 57, 1983, 91–101

[SwaSec90] W. Swartz, C. Sechen, *New Algorithms for the Placement and Routing of Macro Cells*, Procs. IEEE Int. Conf. on Computer Aided Design, 1990, 336–339

[SYS] Chair of Systems Analysis, Department of Computer Science, University of Dortmund, `http://ls11-www.informatik.uni-dortmund.de`

[Tam92a] K. Y. Tam, *Genetic algorithms, function optimization, and facility layout design*, European Journal of Operational Research, North-Holland, Vol. 63, 1992, 322–346

[Tam92b] K. Y. Tam, *A simulated annealing algorithm for allocating space to manufacturing cells*, Int. Journal of Production Research, Vol. 30, No. 1, 1992, 63–87

[TamLi91] K. Y. Tam, S. G. Li, *A hierarchical approach to the facility layout problem*, Int. Journal of Production Research, Vol. 29, No. 1, 1991, 165–184

[UNM] S. Forrest, University of New Mexico, Dept. of Computer Science, `http://www.cs.unm.edu/~forrest`

[UpSaSu90] M. Upton, K. Samii, S. Sugiyama, *Integrated Placement for Mixed Macro Cell and Standard Cell Designs*, Procs. 27th ACM/IEEE Design Automation Conference, 1990, 32–35

[UWE] Evolutionary Computing Group, University of the West of England, Bristol, U. K., `http://www.btc.uwe.ac.uk/evol/index.html`

[vCCaVa91] D. J. van Camp, M. W. Carter, A. Vannelli, *A nonlinear optimization approach for solving facility layout problems*, European Journal of Operational Research, North-Holland, Vol. 57, 1991, 174–189

[Vask89] F. J. Vasko, *A Computational Improvement to Wang's Two-Dimensional Cutting Stock Algorithm*, Computers and Industrial Engineering, Vol. 16, No. 1, 1989, 109–115

[Wang83] P. Y. Wang, *Two Algorithms for Constrained Two-Dimensional Cutting Stock Patterns*, Operations Research, Vol. 31, No. 3, 1983, 573–586

[WaGoPu96] G. Wang, E. D. Goodman, W. F. Punch, *Simultaneous Multi-Level Evolution*, Technical Report 96-03-01, MSU GARAGe, Michigan State University, 1996

[WhiChr77] C. Whitlock, N. Christofides, *An Algorithm for Two-Dimensional Cutting Problems*, Operartions Research, Vol. 25, No. 1, 1977, 30–44

[WolMac95] D. H. Wolpert, W. G. Macready, *No Free Lunch Theorems for Search*, Technical Report SFI–TR–95–02–010, Santa Fe Institute, New Mexico, 1995

[WoLeLi88] D. F. Wong, H. W. Leong, C. L. Liu, *Simulated Annealing for VLSI Design*, Kluwer Academic Publishers, 1988

[Zimm88] G. Zimmermann, *A new area and shape function estimation technique for VLSI layouts*, IEEE/ACM Procs. 25th Design Automation Conference, 1988, 60–65