Bibliography

[1] Stuart E. Dreyfuss, "An Appraisal of Some Shortest-Path Algorithms," *Journal of the Operations Research Society of America*, Vol. 17, pp. 395-412, 1969.

[2] Henrik Esbensen, "Computing Near-Optimal Solutions to the Steiner Problem in a Graph Using a Genetic Algorithm," Technical Report, DAIMI PB-348, Aarhus University, Feb. 1994.

[3] Henrik Esbensen, Pinaki Mazumder, "A Genetic Algorithm for the Steiner Problem in a Graph," *Proceedings of the European Design and Test Conference*, pp. 402-406, 1994.

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

[5] John H. Holland, *Adaption in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI, 1975.

[6] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976.

[7] Y. Nishizaki, M. Igusa, A. Sangiovani-Vincentelli, "Mercury: A New Approach to Macro-cell Global Routing," *Proceedings of the IFIP 10/WG10.5 International Conference on VLSI*, Munich, 1989.

[8] Carl Sechen, *VLSI Placement and Global Routing Using Simulated Annealing*, Kluwer Academic Publishers, Boston, 1988.

[9] G. F. Sullivan, "Approximation Agorithms for Steiner Tree Problems," Technical Report 249, Dept. of Computer Science, Yale University, 1982.

[10] R. Venkateswaran, P. Mazumder, "Routing Agorithms in Semiconductor Circuit Design," In preparation.

densities and updating channel densities dynamically would hence reduce runtime significantly. Another approach is to implement a parallel version of the router. Due to the inherent parallelism of any GA, a high speedup can be expected on any MIMD architecture [4].

D.5 Conclusion and Future Work

In this paper a novel approach to global routing of macro-cell layouts based on genetic algorithms has been presented. The performance of the router is compared to that of TimberWolfMC on MCNC benchmarks. Experimental results show that the quality of completed layouts improves when using the GA-based router instead of TimberWolfMC, assuring that the quality of the given placement is sufficiently high. The router is inferior to TimberWolfMC with respect to runtime, but major improvements are possible. Since the work presented here is a first approach to global routing based on genetic algorithms, future improvements of the layout quality obtainable are also very likely. We conclude that the genetic algorithm is well suited as the basic algorithm of a global router.

fined this assumption. The topology of the routing graph of ami33-2-M is unaltered throughout the process and the performance of the GA-based router is now superior to that of TimberWolfMC. Very similar results are observed for ami49-M. The routing graph topology is significantly altered during the layout process. The placement of ami49-2-M is obtained the same way as ami33-2-M, and the performance of the GA-based router improves significantly on this example.

The significant routing graph alterations for some problems are a consequence of rather poor initial placements. It is not clear how better placements would affect the relative performance of the two routers. As placement quality increases, the relative effect of eliminating a wire from the longest path in a polar graph increases, indicating a potential advantage for the GA-based router. On the other hand, a good placement contains less routing, suggesting that the performance gap would be narrowed.

For the test examples considered here, most routing channels on the longest paths are compacted to their minimum widths by the compactor, cf. the second assumption discussed in Section D.3.1. However, in most cases at least one channel on the longest paths are still wider than necessary. Hence, the area estimation performed tends to underestimate the final area. However, this assumption appears to be fairly reasonable.

D.4.4 Runtime

On average the router requires about 22, 12 and 130 minutes to route examples based on xerox, ami33 and ami49, respectively. TimberWolfMC spends about 30 seconds for examples based on xerox and ami33, and about 5 minutes for ami49-based examples. Hence, the GA-based router is clearly inferior to TimberWolfMC with respect to runtime. The total layout generation process performed by Mosaico (i.e. excluding placement) requires about 15 minutes for examples based on xerox and ami33, and about an hour for ami49-based examples, when TimberWolfMC is used. Hence, the use of the GA-based router increases the layout generation time by a factor of two or three.

However, the runtime of the current implementation can be improved significantly in a number of ways. The vast majority of the runtime is spend computing channel densities. When estimating the area of a solution, all densities are recomputed whether the routing in a channel is actually changed or not. Keeping track of the need to recompute channel

100(GAresult/TWresult − 1). Hence, a negative value indicates a reduction in percent obtained by the GA-based router, while a positive value indicates a percentage overhead as compared to TimberWolfMC. Despite the inherent problems of this kind of comparison discussed in Section D.4.2, it is clear that in general the GA-based global router obtains the best layout quality for the problem instances considered.

Problem	Solution	A_{tot}	A_{route}	WL
xerox-M	best	−1.9	−4.7	+0.0
	avg.	−1.4	−3.5	−0.8
ami33-M	best	−3.2	−5.1	−3.2
	avg.	+1.6	+2.5	−0.2
ami33-2-M	best	−3.0	−4.7	−1.5
	avg.	−1.1	−1.7	−0.2
ami49-M	best	−1.9	−3.3	−1.5
	avg.	−0.5	−0.8	−0.3
ami49-2-M	best	−4.2	−7.3	−4.0
	avg.	−3.7	−6.3	−2.9

Table D.2: *Relative improvements obtained by the GA-based router. best and avg. is best and average of the five runs performed.*

Inspection of the generated layouts reveals interesting information regarding the two major assumptions underlying the area estimation, discussed in Section D.3.1. The placement of xerox-M is adjusted only slightly during compaction, and the routing graph topology is unaltered. For this example, the GA-based router obtains an average reduction of 3.5 % of the routing area which comes at the price of a 0.8 % increase in total wirelength. However, for ami33-M the GA-based router on average obtains larger layouts than TimberWolfMC. In this case the placement, and hence the routing graph topology, is significantly modified by the compactor. As a consequence, the function minimized by the GA-based router in its second phase correlates very poorly to the actual layout generated, which inevitably leads to a poor result. To counteract this phenomenon, a new placement ami33-2-M was produced by ripping up all routing in the completed layout of ami33-M generated using TimberWolfMC. Since the placement thus obtained is the result of compaction and completion of all routing, it will probably only be subjected to minor adjustments when used itself as input to Mosaico. Experiments con-

in Ottools. ami33-2-M and ami49-2-M are other placements of ami33-M and ami49-M respectively. The generation of these placements are described in Section D4.3.

D4.2 Method

Two factors makes it difficult to device a sequence of experiments providing an absolute fair performance comparison of the two global routers. Firstly, global routing is just one of a sequence of heavily interacting steps needed to generate a complete layout. Hence, when considering a specific result, it may be influenced by a pattern of interactions with other tools, which accidentally favours one of the routers. Secondly, the optimization strategies used by the two routers are not identical. As described earlier, the GA-based router explicitly attempts to minimize area and secondarily wirelength. While TimberWolfMC also generates the shortest possible routes in phase one, area is not an explicit component of the optimization criterion used in the second phase. Instead, TimberWolfMC selects the shortest possible routes subject to channel capacity constraints.

The chosen strategy for experiments are as follows: For each of the placed examples listed in Table D.1, Mosaico was executed to generate a complete layout, using either TimberWolfMC or the GA-based router for the global routing task. Hence, all other steps of the layout process are performed by the same tools.

Mosaico was executed five times for each example using the GA-based global router in order to capture the variations caused by the stochastic nature of the applied algorithms. The same set of parameters are used for all program executions, i.e., no problem-specific tuning is performed. For each net, at most $R = 30$ alternative routes are generated. The parameters of the GA used in phase one are as given in [3]. The phase two GA is executed with population size $M = 40$, stop criteria $S = 100$, mutation probability $p_{mut} = 2.5 \times 10^{-4}$ and inversion probability $p_{inv} = 0.1$. There is no variation of results when applying TimberWolfMC.

D4.3 Layout Quality

Table D.2 summarizes the impact on the completed layouts of using the GA-based router instead of TimberWolfMC. A_{tot} denotes total area, A_{route} denotes routing area, i.e., the part of the total area not occupied by cells and W denotes total wirelength. Each entry is computed as

form a ring. A part of the ring is then selected at random and reversed. More specifically, two points $x, y \in \{0, 1, \ldots, N-1\}$, $x \neq y$, are selected at random. The operator then defines the new ordering π' as [2] $\pi'((x+i) \bmod N) = \pi((y-i) \bmod N)$ if $0 \leq i \leq (y-x) \bmod N$ and $\pi'((x+i) \bmod N) = \pi((x+i) \bmod N)$ otherwise, $i = 0, 1, \ldots, N-1$.

D.4 Experimental Results

The router has been implemented in the C programming language, and all experiments are performed on a Sun Sparc IPX workstation. The router is interfaced with the macro-cell layout system Mosaico, which is a part of the Octtools CAD framework developed at University of California, Berkeley. This integration allows for comparison of the routers performance to that of TimberWolfMC [8], a state of the art global router also interfaced to Mosaico.

D.4.1 Test Examples

Three of the MCNC macro-cell benchmarks, xerox, ami33 and ami49, were used for the experiments. However, due to a purely technical problem, it became necessary to remove all pads from these examples before using them [3]. The modified benchmarks are referenced using a '-M' suffix.

Problem	#cells	#nets	#terms
xerox-M	10	203	696
ami33-M	33	85	442
ami33-2-M	33	85	442
ami49-M	49	390	931
ami49-2-M	49	390	931

Table D.1: *Problem characteristics.*

Table D.1 lists the main characteristics of the test examples. The number of nets and the number of terminals listed are totals, i.e., they include the few trivial nets. xerox-M, ami33-M and ami49-M are placed by Puppy, a placement tool based on simulated annealing, also included

[2] The definition of π' relies on the mathematical definition of modulo, in which the remainder is always non-negative.

[3] In our version of Octtools (5.2) the channel definition program Atlas can not handle the pad placement generated by Padplace.

The population $P = \{ p_0, p_1, \ldots, p_{M-1} \}$ is then sorted lexicographically using area as most significant criterion and wirelength as a secondary criterion. Assume that P is sorted in decreasing order with respect to this ordering. The fitness F of p_i is then computed as $F(p_i) = 2i / (M-1)$ for $i = 0, 1, \ldots, M-1$. This scheme, called *ranking*, assures constant variance of fitness throughout the optimization process. Ranking is a good approach for controlling the speed of convergence, including the avoidance of premature convergence.

D.3.2.3 Crossover Operator

Given two parent individuals α and β, the crossover operator generates two offspring, ϕ and ψ. The parent individuals are not altered by the operator. In the following, a (second) subscript specifies which individual the marked property is a part of. Crossover consists of two steps:
1) One of the parents, say β, is chosen at random, and a copy γ of β is made. γ is then reordered so that it becomes homologous to α, that is, $\pi_\gamma = \pi_\alpha$.
2) The offspring are given the same ordering as their parents: $\pi_\phi = \pi_\psi = \pi_\alpha$. Standard 1-point crossover is then performed [5]: A crossover-point x is selected at random in $\{0, 1, \ldots, N-2\}$. The selected routes of ϕ is then defined by $q_{\pi(k),\phi} = q_{\pi(k),\alpha}$ if $k \leq x$ and $q_{\pi(k),\phi} = q_{\pi(k),\gamma}$ otherwise, where $\pi = \pi_\alpha$. Similarly, the selected routes of ψ is defined by $q_{\pi(k),\psi} = q_{\pi(k),\gamma}$ if $k \leq x$ and $q_{\pi(k),\psi} = q_{\pi(k),\alpha}$ otherwise.

D.3.2.4 Mutation and Inversion Operators

The mutation operator is very simple. It goes through the N tuples of the given individual and randomly selects another route for the k'th net with probability $p_{mut}(r_k - 1)$, where p_{mut} is a small user-defined probability. This scheme is called pointwise mutation.

As mentioned in Section D.3.2.1 a given global routing solution can be represented by several equivalent individuals because of the independence of the ordering π. However, the fitness of offspring produced by crossover depends on the specific orderings of the given parent individuals. The purpose of inversion is to optimize the performance of the crossover operator. With a given probability p_{inv}, the inversion operator alters the ordering π of a given individual. To obtain a uniform probability of movement of all tuples, we consider the set of tuples to

cess. Routine *evaluate* described in Section D3.2.2 computes the fitness of each of the given individuals, while *bestOf* finds the individual with the highest fitness. One execution of the outer "repeat" loop corresponds to the simulation of one generation. Throughout the simulation, M is kept constant. We keep track of the best individual s ever seen. Routine *stopCriteria* terminates the simulation when no improvement has been observed for S consecutive generations. Each generation is initiated by the formation of a set of offspring P_N of size M. The two mates p_1 and p_2 are selected independently of each other, and each mate is selected with a probability proportional to its fitness. The *crossover* routine described in Section D3.2.3 generates two offspring c_1 and c_2. Routine *reduce* returns the M fittest of the given individuals, thereby keeping the population size constant. The genetic operators for mutation and inversion are discussed in Section D3.2.4. Routine *optimize*(s) performs simple hill-climbing by executing a sequence of mutations on s, each of which improves the fitness of s. The output of the algorithm is then the solution s.

D3.2.1 Representation

A global routing solution is represented by specifying for each net which of its possible routes is used. More specifically, assume a fixed numbering $0, 1, \ldots, N-1$ of the nets, let $\pi : \{0, 1, \ldots, N-1\} \mapsto \{0, 1, \ldots, N-1\}$ be a bijection and denote by $r_k \leq R$ the number of routes generated in phase one for the k'th net. An individual is then a set of N tuples:

$$\{(\pi(0), q_{\pi(0)}), (\pi(1), q_{\pi(1)}), \ldots, (\pi(N-1), q_{\pi(N-1)})\}$$

where $1 \leq q_k \leq r_k$ for all $k = 0, 1, \ldots, N-1$. For example, the tuple (3,7) specifies that the 3rd net uses its 7'th route. The mapping π defines an ordering of the nets, the purpose of which is explained in Section D3.2.4. Note that the routing solution specified by an individual is independent of π.

D3.2.2 Definition of Fitness

Given a population P, the routine *evaluate* of Fig. D5 computes the fitness of each individual as follows. For each individual $p \in P$, its estimated area is computed as described in Section D3.1 and its estimated total wirelength is computed as the sum of the length of the routes specified by p.

The algorithm maintains a *population* of *individuals*, each of which corresponds to a specific solution. A measure of *fitness* defines the quality of an individual. Starting with some set of individuals, a process of evolution is simulated. The main components of this process are *crossover*, which mimics propagation, and *mutation*, which mimics the random changes occurring in nature. After a number of *generations*, highly fit individuals will emerge corresponding to good solutions to the given optimization problem. A good introduction to GAs is given in [4].

```
generate(P_C);
evaluate(P_C);
s = bestOf(P_C);
repeat until stopCriteria():
   P_N = ∅;
   repeat M/2 times:
      select p_1 ∈ P_C, p_2 ∈ P_C;
      crossover(p_1, p_2, c_1, c_2);
      P_N = P_N ∪ {c_1, c_2};
   end;
   evaluate(P_C ∪ P_N);
   P_C = reduce(P_C ∪ P_N);
   ∀ p ∈ P_C : possibly mutate(p);
   ∀ p ∈ P_C : possibly invert(p);
   evaluate(P_C);
   s = bestOf(P_C ∪ {s});
end;
optimize(s);
output s;
```

Figure D.5: *Outline of phase two.*

Fig. D.5 outlines the phase two algorithm. Initially, the current population P_C of size $M = |P_C|$ consists of $M - 1$ randomly generated individuals and a single individual consisting of the shortest route found for each net. Seeding the population with this relatively good solution does not lead to better final results, but merely speeds up the search pro-

of the longest path in HPG then estimates the horizontal length of the layout. By constructing VPG in a similar way, the area is estimated as the product of the longest path in HPG times the longest path in VPG.

In [7] the cost of an edge in the polar graphs is a rather simple function of the number of nets present in the corresponding routing channel. However, if m nets are present in a channel, the channel density can be any number between 0 and m, assuming that two metal layers are available for routing and that each layer is used exclusively for routing in a specific direction. Therefore, to obtain a more accurate area estimate, we compute the exact channel density for each edge in the routing graph. This is possible since the routing in phase one was performed using accurate positions for the terminals of each net, cf. Section D2. The cost of an edge in the polar graphs is then proportional to the density of the corresponding channel.

Several factors affects the accuracy of the area estimate. The two most important has to do with the subsequent compaction/spacing of the layout:

1) If the compactor alters the placement to the extent where the topology of the routing graph is changed, the polar graphs are also changed. Hence, the quality of the area estimate decreases significantly or may even become meaningless. In other words, a good initial placement is required so that the compactor will only perform minor adjustments of the cell positions. This situation reflects the well-known strong mutual dependency of the placement and global routing tasks.

2) It is implicitly assumed that the compactor generates a layout in which no routing channel on a longest path of a polar graph is wider than needed. Otherwise, the area will be underestimated.

The practical consequences of these assumptions are addressed in Section D4.3.

D3.2 Area and Wirelength Optimization

The concept of genetic algorithms, introduced by John Holland [5], utilizes the notion of the natural evolution process. In nature, the individuals constituting a population adapt to the environment in which they live. The fittest individuals have the highest probability of survival and tend to increase in numbers, while the less fit individuals tend to die out. This *survival-of-the-fittest* Darwinian principle is the basic idea behind the GA.

considers fewer distinct solutions and is slower than the GA. Therefore, the GA is used for every net with more than two terminals [1].

D.3 Phase Two of the Router

The area estimate is of course crucial to the phase two algorithm, and is discussed in Section D.3.1. A detailed description of the GA performing the optimization then follows in Section D.3.2.

D.3.1 Area Estimation

As in [7, 10] the area estimation is based on the formation of polar graphs as illustrated in Fig D.4. For a given placement and routing graph, two polar graphs are constructed, a horizontal (HPG) and a vertical (VPG). Let us start by considering HPG. The vertices of HPG consists of a vertex for each cell plus two additional vertices, a source and a sink. Each edge in HPG corresponds to a vertical edge in the routing graph and is directed from the source towards the sink.

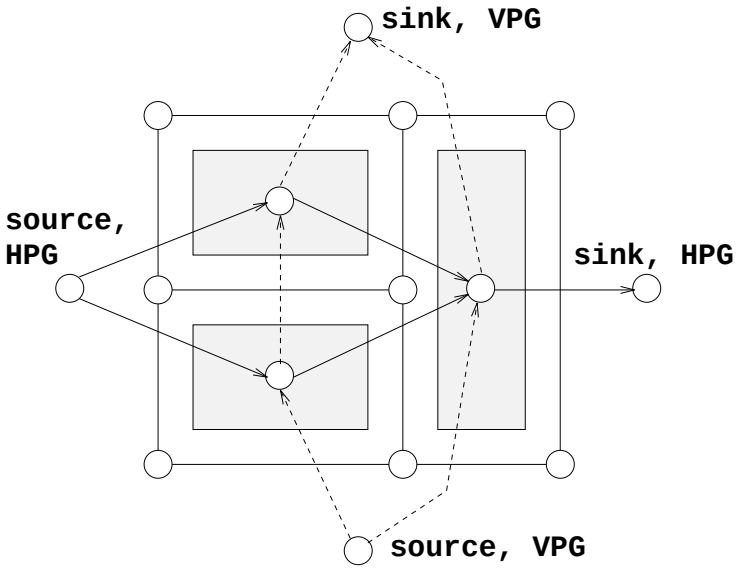

Figure D.4: *Polar graphs for area estimation.*

Assume that each edge (v, w) has a cost which corresponds to the spacing needed between cells v and w to perform the routing. Furthermore, assign to each path from source to sink a fixed cost which is the sum of the horizontal length of all cells visited on the path. The total cost

[1] To obtain as many distinct solutions as possible, the GA does not use the reduction of the search space described in [2, 3].

D2.1 Two-terminal nets

For each net with two terminals, an algorithm due to Lawler [6] is used to compute the shortest, second-shortest, third-shortest, etc. route until a maximum of R routes are found or no more routes exists. Lawlers algorithm is exact but also quite expensive, requiring time $O(Rn^3)$ for one net, where n is the number of vertices in the routing graph.

An earlier algorithm by Dreyfuss [1] may at first seem more attractive. It generates the R shortest routes from a designated vertex to each of the other vertices in time $O(Rn \log n)$. However, loops are allowed in a path, as opposed to Lawlers algorithm, and if two paths do not visit the same vertices in the same order they are considered distinct. One could then simply generate routes until R loopless routes were obtained, which were also distinct in the sense that their sets of edges are distinct. However, experiments have shown that this strategy is not feasible in practice due to the number of routes then required.

D2.2 Nets with at least three terminals

At most R distinct routes are generated for each net having three or more terminals using a GA for the SPG. For a detailed description of that algorithm the reader is referred to [2, 3]. There are two main advantages of using that algorithm in this context. Firstly, it generates high-quality solutions. In [2] the GA is tested on graphs with up to 2,500 vertices and is found to be within 1 % from the global optimum solution in more than 92 % of all runs. The routing graph of a macro-cell placement with C cells will have less than $3C$ vertices. It is therefore most likely that the GA will find the shortest existing route for every net in any reasonably sized macro-cell layout. The second advantage of the GA is that it provides a number of distinct solutions in a single run. The problem of Mercury and TimberWolfMC that only one route is generated for nets with many terminals is thus eliminated.

For nets with few terminals, say 6-7 or less, exhaustive search for the shortest route will often be feasible. Using an algorithm by Sullivan [9] optimum can be found by exhausting a search space consisting of

$$\sum_{i=0}^{k} \binom{n}{i}$$

points, where $k = \min(t-2,\ n)$ and t is the number of terminals of the net. However, experiments has revealed that Sullivans algorithm often

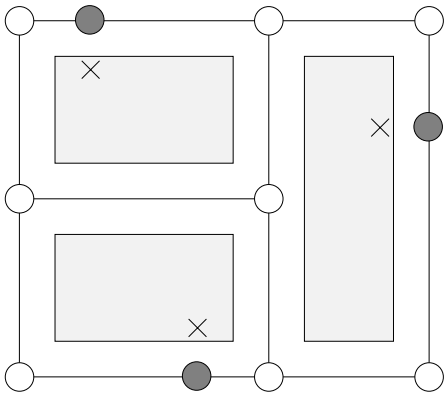

Figure D.2: *Addition of terminal vertices (shaded) for a net with three terminals (marked with crosses).*

Fig. D.3 outlines phase one. A net is *trivial* if all its terminals are projected onto the same edge of the routing graph. Although several routes can still be generated for a trivial net, it will rarely be advantageous. Hence, global routing is skipped for such nets.

```
generate routing graph
for each non-trivial net do:
    add vertices to graph
    if 2-terminal net:
        apply Lawlers algorithm
    else
        apply a GA for SPG
    end
    remove vertices from graph
end
```

Figure D.3: *Outline of phase one.*

The SPG is in general NP-complete. However, if only two vertices are to be connected, SPG reduces to a shortest path problem, which is handled by an algorithm of Lawler discussed in Section D.2.1. Nets with more than two terminals are handled by a GA discussed in Section D.2.2.

computing a corresponding path in the routing graph.

A quite detailed description of how to generate the routing graph for a given placement is given in [7]. Roughly speaking, each edge of the graph corresponds to a routing channel and each vertex corresponds to the intersection of two channels. An example is shown in Fig. D.1.

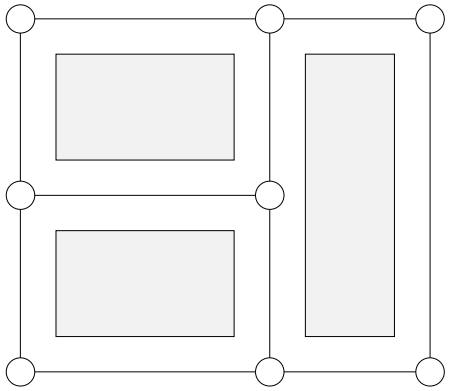

Figure D.1: *A placement and the corresponding routing graph.*

Before finding routes for a given net, vertices representing the terminals of the net are added to the routing graph at appropriate locations. Finding the shortest route for the net is then equivalent of finding a minimum cost subtree in the graph which spans all of the added terminal vertices, assuring that the cost of an edge is defined as its length. This problem is known as the Steiner Problem in a Graph (SPG). When a net has been treated, its terminal vertices are removed from the routing graph before considering the next net, thereby significantly reducing the size of the SPG instances to be solved.

For each terminal the location of the corresponding terminal vertex is determined by a perpendicular projection of the terminal onto the edge representing the appropriate routing channel, as illustrated in Fig. D.2. This is in contrast to the strategy used in e.g. [7]. Here vertices are added only at the center of routing channels and each terminal is then assigned to the closest vertex. This scheme may result in some nets having identical sets of terminal vertices, in which case some computations can be avoided. On the other hand, our scheme provides a more accurate estimate of the wirelength and also allows a more accurate area estimate as discussed in Section D.3.1.

D.1 Introduction

A well-known strategy for global routing of macro-cell layouts consists of two phases [10]. In the first phase, a number of alternative routes are generated for each net. The nets are treated independently one at a time, and the objective is to minimize the length of each net. In the second phase, a specific route is selected for each net, subject to channel capacity constraints, and so that some overall criterion, typically area or total interconnect length, is minimized. A main advantage of this routing strategy is its independence of a net ordering.

Mercury [7] and TimberWolfMC [8] are state of the art global routers for macro-cell layouts, and both are based on the two-phase strategy. For nets with a small number of terminals, these routers generate up to $10 - 20$ alternative routes for each net. However, due to the time complexity of the applied algorithms, only a single route is generated for nets having more than $5 - 11$ terminals. As noted in [8] this limits the overall quality obtainable.

In this paper a new global router is presented which minimizes area and secondarily, total interconnect length. While also being based on the two-phase strategy, this router differs significantly from previous approaches in two ways:

1) Each phase is based on a genetic algorithm (GA). The GA used in phase one provides several high-quality routes for each net independently of its number of terminals. In the second phase another GA minimizes the dual optimization criterion by appropriately selecting a specific route for each net.

2) The estimates of area and total interconnect length used throughout the optimization process are calculated very accurately. The area estimate is based on computation of channel densities and the wirelength estimate is based on exact pin locations.

Experimental results show that the layout quality obtained by the router compares favourably to that of TimberWolfMC.

D.2 Phase One of the Router

Before the global routing process itself is initiated a rectilinear *routing graph* is extracted from the given placement. Routing is then performed in terms of this graph, i.e., computing a global route for a net is done by

Appendix D

A Macro-Cell Global Router Based on Two Genetic Algorithms

This paper is published as H. Esbensen, "A Macro-Cell Global Router Based on Two Genetic Algorithms," *Proc. of The European Design Automation Conference*, pp. 428-433, Sept. 1994.

Abstract

This paper presents a novel approach to global routing of macro-cell layouts. A genetic algorithm generates several short routes for each net. Another genetic algorithm then selects a route for each net while minimizing area and secondarily interconnect length. Exact channel densities are used for area estimation. The layout quality obtained on MCNC benchmarks compares favourably to that of TimberWolfMC.

[27] H. Takahashi, A. Matsuyama, "An Approximate Solution for the Steiner Problem in Graphs", *Mathematica Japonica*, Vol. 24, No. 6, pp. 573-577, 1980.

[28] R. Venkateswaran, P. Mazumder, "Routing Algorithms in Semiconductor Circuit Design," In preparation, 1993.

[29] Darrell Whitley, "The Genitor Algorithm and Selection Pressure: Why Rank-Based Allocation of Reproductive Trials is Best," *Proceedings of the 3th International Conference on Genetic Algorithms*, pp. 116-121, 1989.

[30] Pawel Winter, "Steiner Problem in Networks: A Survey," *Networks*, Vol. 17, pp. 129-167, 1987.

[31] Pawel Winter, J. MacGregor Smith, "Path-Distance Heuristics for the Steiner Problem in Undirected Networks," *Algorithmica*, Vol. 7, pp. 309-327, 1992.

[13] S. L. Hakimi, "Steiner's problem in graphs and its implications," *Networks*, Vol. 1, pp. 113-133, 1971.

[14] M. Hanan, "On Steiner's Problem with Rectilinear Distance," *SIAM Journal of Applied Mathematics*, Vol. 14, No. 2, pp. 255-265, 1966.

[15] J. Hesser, R. Männer, O. Stucky, "Optimization of Steiner Trees using Genetic Algorithms," *Proceedings of the 3th International Conference on Genetic Algorithms*, pp. 231-236, 1989.

[16] John H. Holland, *Adaption in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, MI, 1975.

[17] Bryant A. Julstrom, "A Genetic Algorithm for the Rectilinear Steiner Problem," *Proceedings of the 5th International Conference on Genetic Algorithms*, pp. 474-480, 1993.

[18] A. Kapsalis, V. J. Rayward-Smith, G. D. Smith, "Solving the Graphical Steiner Tree Problem Using Genetic Algorithms," *Journal of the Operational Research Society*, Vol. 44, No. 4, pp. 397-406, 1993.

[19] R. M. Karp, "Reducibility among Combinatorial Problems," In R. E. Miller, J. W. Thatcher (Eds.), *Complexity of Computer Computations*, Plenum Press, New York, pp. 85-103, 1972.

[20] L. Kou, G. Markowsky, L. Berman, "A Fast Algorithm for Steiner Trees," *Acta Informatica*, Vol. 15, pp. 141-145, 1981.

[21] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976.

[22] J. van Leeuwen, "Graph Algorithms," in J. van Leeuwen, ed, *Handbook of Theoretical Computer Science*, Vol. A *Algorithms and Complexity*, Elsevier, Amsterdam, 1990.

[23] A. Lucena, J. E. Beasley, "A branch and cut algorithm for the Steiner problem in graphs," working paper, The Management School, Imperial College, England, July 1992.

[24] J. Plesnik, "A bound for the Steiner tree problem in graphs," *Math. Slovaca*, Vol. 31, pp. 155-163, 1981.

[25] V. J. Rayward-Smith, A. Clare, "On finding Steiner vertices," *Networks*, Vol. 16, pp. 283-294, 1986.

[26] M. L. Shore, L. R. Foulds, P. B. Gibbons, "An algorithm for the Steiner Problem in Graphs," *Networks*, Vol. 12, pp. 323-333, 1982.

Bibliography

[1] A. V. Aho, J. E. Hopcroft, J. D. Ullman, *Data Structures and Algorithms*, Addison-Wesley, Reading, Mass, 1983.

[2] Y. P. Aneja, "An Integer Linear Programming Approach to the Steiner Problem in Graphs," *Networks*, Vol. 10, pp. 167-178, 1980.

[3] J. E. Beasley, "An SST-Based Algorithm for the Steiner Problem in Graphs," *Networks*, Vol. 19, pp. 1-16, 1989.

[4] J. E. Beasley, "OR-Library: distributing test problems by electronic mail," *Journal of the Operational Research Society*, Vol. 41, pp. 1069-1072, 1990.

[5] Sunil Chopra, Edgar R. Gorres, M.R. Rao, "Solving the Steiner Tree Problem on a Graph Using Branch and Cut," *Operations Research Society of America Journal of Computing*, Vol. 4, No. 3, pp. 320-335, 1992.

[6] R. Dionne, M. Florian, "Exact and Approximate Algorithms for Optimal Network Design," *Networks*, Vol. 9, pp. 37-59, 1979.

[7] K. A. Dowsland, "Hill-climbing simulated annealing and the Steiner problem in graphs," *Eng. Opt.*, Vol. 17, pp. 91-107, 1991.

[8] S. E. Dreyfuss, R. A. Wagner, "The Steiner problem in graphs," *Networks*, Vol. 1, pp. 195-207, 1971.

[9] C. W. Duin, A. Volgenant, "Reduction Tests for the Steiner Problem in Graphs," *Networks*, Vol. 19, pp. 549-567, 1989.

[10] L. R. Foulds, V. J. Rayward-Smith, "Steiner problems in graphs: algorithms and applications," *Eng. Opt.* Vol. 7, pp. 7-16, 1983.

[11] M. R. Garey, D. S. Johnson, "The Rectilinear Steiner Tree Problem is NP-complete," *SIAM Journal of Applied Mathematics*, Vol. 32, No. 4, pp. 826-834, 1977.

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

Graph	Cost					CPU-time (secs)			
	C_{opt}	C_{sph}	ΔC_{sph}	C_{ga}	ΔC_{ga}	T_{bc1}	T_{bc2}	T_{sph}	T_{ga}
E-1	111	111	0	111	0	1,150	1,394	7,334	7,395
E-2	214	216	0.93	216	0.93	6,251	1,993	7,355	7,444
E-3	4,013	4,060	1.17	4,013	0	26,468	15,782	50,004	9,449
E-4	5,101	5,113	0.24	5,102	0.02	46,008	1,660	29,921	7,763
E-5	8,128	8,134	0.07	8,128	0	12,564	411	9,318	7,474
E-6	73	76	4.11	73	0	678	-	10,060	10,148
E-7	145	149	2.76	145	0	27,124	-	10,306	10,458
E-8	2,640	2,690	1.89	2,646	0.23	118,618	-	50,013	12,896
E-9	3,604	3,671	1.86	3,611	0.19	24,528	-	50,014	14,933
E-10	5,600	5,624	0.43	5,600	0	39,261	-	50,014	12,976
E-11	34	34	0	34	0	1,901	-	14,472	14,559
E-12	67	68	1.49	68	1.49	7,200	-	14,497	14,588
E-13	1,280	1,317	2.89	1,289	0.70	207,059	-	50,003	21,787
E-14	1,732	1,767	2.02	1,736	0.23	29,263	-	50,030	23,022
E-15	2,784	2,795	0.40	2,784	0	7,666	-	50,020	18,424
E-16	15	15	0	15	0	179	-	14,425	14,586
E-17	25	25	0	25	0	36,040	-	14,458	14,619
E-18	572	625	9.27	583	1.92	-	-	50,017	29,105
E-19	758	802	5.80	766	1.06	6,372	-	50,037	27,319
E-20	1,342	1,357	1.12	1,342	0	272	-	50,055	25,107

Table C.12: *Comparison of solution quality and CPU-time for the graphs in class E.*

Graph	T_{bc1}	T_{bc2}	T_{sph}	T_{avg}	T_{σ}
D-1	476	200	486	523	8
D-2	284	148	488	537	13
D-3	2,290	106	785	650	39
D-4	3,529	41	689	554	21
D-5	811	37	522	504	8
D-6	2,340	4,148	687	788	44
D-7	100	1,037	681	795	29
D-8	6,985	17,858	13,237	2,101	381
D-9	4,630	16,458	29,354	2,744	624
D-10	1,312	1,678	14,780	1,100	169
D-11	1,374	24,609	949	1,070	59
D-12	305	5,843	961	1,085	20
D-13	1,864	91,718	15,187	2,357	245
D-14	3,538	61,335	41,237	2,601	393
D-15	1,410	16,889	24,828	1,302	102
D-16	871	9,721	956	1,047	21
D-17	6,965	147,598	950	1,068	26
D-18	245,192	227,841	31,015	2,536	491
D-19	878	304,380	50,003	3,441	580
D-20	47	1,276	50,010	2,638	658

Table C10: *Comparison of CPU-time (seconds) for the graphs in class D.*

	Problem size			Reduced size		
Graph	n	m	$\|E\|$	n	m	$\|E\|$
E-1	2,500	5	3,125	680	5	1,286
E-2	2,500	10	3,125	710	9	1,328
E-3	2,500	417	3,125	637	199	1,233
E-4	2,500	625	3,125	435	164	964
E-5	2,500	1,250	3,125	222	108	649
E-6	2,500	5	5,000	1,845	5	4,318
E-7	2,500	10	5,000	1,891	10	4,372
E-8	2,500	417	5,000	1,723	286	4,193
E-9	2,500	625	5,000	1,608	358	4,069
E-10	2,500	1,250	5,000	1,046	366	3,388
E-11	2,500	5	12,500	2,498	5	12,093
E-12	2,500	10	12,500	2,500	10	12,123
E-13	2,500	417	12,500	2,341	321	11,760
E-14	2,500	625	12,500	2,139	388	11,325
E-15	2,500	1,250	12,500	1,461	443	8,514
E-16	2,500	5	62,500	2,500	5	29,332
E-17	2,500	10	62,500	2,500	10	29,090
E-18	2,500	417	62,500	2,429	355	28,437
E-19	2,500	625	62,500	2,351	485	27,779
E-20	2,500	1,250	62,500	1,988	758	24,423

Table C11: *The class E graphs before and after reductions.*

Graph	Problem size			Reduced size		
	n	m	$\|E\|$	n	m	$\|E\|$
D 1	1,000	5	1,250	274	5	510
D 2	1,000	10	1,250	285	10	523
D 3	1,000	167	1,250	224	67	441
D 4	1,000	250	1,250	159	66	339
D 5	1,000	500	1,250	97	48	246
D 6	1,000	5	2,000	761	5	1,741
D 7	1,000	10	2,000	754	10	1,735
D 8	1,000	167	2,000	731	124	1,708
D 9	1,000	250	2,000	654	155	1,613
D 10	1,000	500	2,000	418	146	1,317
D 11	1,000	5	5,000	993	5	4,674
D 12	1,000	10	5,000	1,000	10	4,671
D 13	1,000	167	5,000	922	122	4,433
D 14	1,000	250	5,000	853	160	4,173
D 15	1,000	500	5,000	550	157	2,925
D 16	1,000	5	25,000	1,000	5	10,595
D 17	1,000	10	25,000	999	9	10,531
D 18	1,000	167	25,000	978	145	10,140
D 19	1,000	250	25,000	938	193	9,676
D 20	1,000	500	25,000	814	324	8,907

Table C.8: *The class D graphs before and after reductions.*

Graph	C_{opt}	C_{sph}	ΔC_{sph}	C_{best}	C_{avg}	C_{worst}	C_σ	ΔC_{avg}	ΔC_{worst}	N_{ga}
D 1	106	106	0	106	106	106	0	0	0	0
D 2	220	220	0	220	220	220	0	0	0	0
D 3	1,565	1,570	0.32	1,565	1,565	1,565	0	0	0	0
D 4	1,935	1,940	0.26	1,935	1,935	1,935	0	0	0	0
D 5	3,250	3,254	0.12	3,250	3,250	3,250	0	0	0	0
D 6	67	71	5.97	67	67.1	68	0.3	0.15	1.49	1
D 7	103	103	0	103	103	103	0	0	0	0
D 8	1,072	1,095	2.15	1,072	1,072.7	1,074	0.6	0.07	0.19	6
D 9	1,448	1,471	1.59	1,448	1,448.4	1,450	0.7	0.03	0.14	3
D 10	2,110	2,120	0.47	2,110	2,110	2,110	0	0	0	0
D 11	29	29	0	29	29	29	0	0	0	0
D 12	42	42	0	42	42	42	0	0	0	0
D 13	500	514	2.80	500	500.6	502	0.7	0.12	0.40	5
D 14	667	675	1.20	668	669.7	671	0.9	0.40	0.60	10
D 15	1,116	1,121	0.45	1,116	1,116	1,116	0	0	0	0
D 16	13	13	0	13	13	13	0	0	0	0
D 17	23	23	0	23	23	23	0	0	0	0
D 18	223	239	7.17	226	227.7	230	1.2	2.11	3.14	10
D 19	310	335	8.06	312	313.3	315	0.9	1.06	1.61	10
D 20	537	539	0.37	537	537	537	0	0	0	0

Table C.9: *Comparison of solution quality for the graphs in class D.*

Graph	C_{opt}	C_{sph}	ΔC_{sph}	C_{best}	C_{avg}	C_{worst}	C_{σ}	ΔC_{avg}	ΔC_{worst}	N_{ga}
C 1	85	85	0	85	85	85	0	0	0	0
C 2	144	144	0	144	144	144	0	0	0	0
C 3	754	757	0.40	754	754.2	755	0.4	0.03	0.13	2
C 4	1,079	1,081	0.19	1,079	1,079.1	1,080	0.3	0.01	0.09	1
C 5	1,579	1,579	0	1,579	1,579	1,579	0	0	0	0
C 6	55	55	0	55	55	55	0	0	0	0
C 7	102	102	0	102	102	102	0	0	0	0
C 8	509	512	0.59	509	509	509	0	0	0	0
C 9	707	714	0.99	707	707.4	708	0.5	0.06	0.14	4
C 10	1,093	1,098	0.46	1,093	1,093	1,093	0	0	0	0
C 11	32	32	0	32	32	32	0	0	0	0
C 12	46	46	0	46	46	46	0	0	0	0
C 13	258	263	1.94	258	259.7	260	0.6	0.66	0.78	9
C 14	323	327	1.24	323	323.4	324	0.5	0.12	0.31	4
C 15	556	558	0.36	556	556	556	0	0	0	0
C 16	11	11	0	11	11.7	12	0.5	6.36	9.09	7
C 17	18	18	0	18	18	18	0	0	0	0
C 18	113	121	7.08	113	114.3	115	0.8	1.15	1.77	8
C 19	146	155	6.16	146	147	148	0.4	0.68	1.37	9
C 20	267	267	0	267	267	267	0	0	0	0

Table C6: *Comparison of solution quality for the graphs in class C.*

Graph	T_{bc1}	T_{bc2}	T_{sph}	T_{avg}	T_{σ}
C 1	27	25	61	79	6
C 2	812	45	61	79	3
C 3	543	25	72	104	19
C 4	510	23	75	83	10
C 5	474	5	61	63	0
C 6	49	561	83	130	11
C 7	83	522	86	153	24
C 8	674	1,106	260	263	39
C 9	1,866	5,813	966	425	93
C 10	246	32	544	181	49
C 11	333	2,769	119	187	20
C 12	120	1,175	119	224	19
C 13	9,170	9,895	646	544	91
C 14	212	1,150	1,316	547	130
C 15	211	913	1,544	262	56
C 16	10	877	119	180	22
C 17	98	14,557	119	203	26
C 18	45,848	20,726	873	563	102
C 19	117	1,689	3,050	601	136
C 20	15	225	11,374	334	57

Table C7: *Comparison of CPU-time (seconds) for the graphs in class C.*

Graph	T_{bc2}	T_{sph}	T_{avg}	T_{σ}
B-1	0.1	0.1	0.1	0.0
B-2	0.1	0.1	0.2	0.0
B-3	0.1	0.1	0.1	0.0
B-4	0.6	0.1	1.2	0.6
B-5	1.9	0.1	0.7	0.2
B-6	0.6	0.1	0.7	0.1
B-7	0.2	0.2	0.5	0.1
B-8	0.1	0.2	0.5	0.1
B-9	0.1	0.2	0.2	0.0
B-10	3.1	0.3	1.7	0.5
B-11	1.4	0.3	1.4	0.6
B-12	0.6	0.3	0.6	0.1
B-13	0.7	0.4	1.4	0.4
B-14	1.2	0.5	0.9	0.3
B-15	0.3	0.5	0.8	0.1
B-16	18.4	0.6	4.4	1.9
B-17	3.3	0.6	2.3	0.6
B-18	1.0	0.6	1.5	0.3

Table C.4: *Comparison of CPU-time (seconds) for the graphs in class B.*

	Problem size			Reduced size		
Graph	n	m	$\|E\|$	n	m	$\|E\|$
C-1	500	5	625	145	5	263
C-2	500	10	625	130	8	239
C-3	500	83	625	120	35	232
C-4	500	125	625	109	38	221
C-5	500	250	625	37	17	91
C-6	500	5	1,000	369	5	847
C-7	500	10	1,000	382	9	869
C-8	500	83	1,000	336	54	818
C-9	500	125	1,000	349	78	832
C-10	500	250	1,000	213	76	624
C-11	500	5	2,500	499	5	2,184
C-12	500	10	2,500	498	9	2,236
C-13	500	83	2,500	463	65	2,108
C-14	500	125	2,500	427	81	1,961
C-15	500	250	2,500	299	92	1,471
C-16	500	5	12,500	500	5	4,740
C-17	500	10	12,500	499	9	4,698
C-18	500	83	12,500	486	70	4,668
C-19	500	125	12,500	473	98	4,490
C-20	500	250	12,500	386	143	3,850

Table C.5: *The class C graphs before and after reductions.*

C.7 Computational Results

Graph	Problem size			Reduced size		
	n	m	$\|E\|$	n	m	$\|E\|$
B.1	50	9	63	1	1	0
B.2	50	13	63	7	4	12
B.3	50	25	63	1	1	0
B.4	50	9	100	34	7	72
B.5	50	13	100	35	10	76
B.6	50	25	100	25	10	60
B.7	75	13	94	16	6	26
B.8	75	19	94	16	7	25
B.9	75	38	94	1	1	0
B.10	75	13	150	50	10	115
B.11	75	19	150	47	8	108
B.12	75	38	150	31	11	74
B.13	100	17	125	28	9	47
B.14	100	25	125	22	8	42
B.15	100	50	125	16	9	28
B.16	100	17	200	63	9	148
B.17	100	25	200	51	12	113
B.18	100	50	200	35	12	84

Table C.2: *The class B graphs before and after reductions.*

Graph	C_{opt}	C_{sph}	ΔC_{sph}	C_{best}	C_{avg}	C_{worst}	C_{σ}	ΔC_{avg}	ΔC_{worst}	N_{ga}
B.1	82	82	0	82	82	82	0	0	0	0
B.2	83	83	0	83	83	83	0	0	0	0
B.3	138	138	0	138	138	138	0	0	0	0
B.4	59	59	0	59	59	59	0	0	0	0
B.5	61	61	0	61	61	61	0	0	0	0
B.6	122	122	0	122	122	122	0	0	0	0
B.7	111	111	0	111	111	111	0	0	0	0
B.8	104	104	0	104	104	104	0	0	0	0
B.9	220	220	0	220	220	220	0	0	0	0
B.10	86	86	0	86	86	86	0	0	0	0
B.11	88	88	0	88	88	88	0	0	0	0
B.12	174	174	0	174	174	174	0	0	0	0
B.13	165	168	1.82	165	165	165	0	0	0	0
B.14	235	235	0	235	235	235	0	0	0	0
B.15	318	318	0	318	318	318	0	0	0	0
B.16	127	127	0	127	127	127	0	0	0	0
B.17	131	131	0	131	131	131	0	0	0	0
B.18	218	218	0	218	218	218	0	0	0	0

Table C.3: *Comparison of solution quality for the graphs in class B.*

C.6 Conclusion

In this paper a new Genetic Algorithm (GA) for the Steiner Problem in a Graph (SPG) has been presented. The main idea behind the algorithm is the application of the Distance Network Heuristic for interpretation of bitstrings specifying selected Steiner vertices. This scheme ensures that every bitstring corresponds to a valid solution and eliminates the need for penalty terms in the cost measure, thereby avoiding potential problems of assigning a suitable cost value to an incomplete or invalid solution.

The performance of the algorithm has been tested on random graphs with up to 2,500 vertices and 62,500 edges. The experimental results show that in more than 92 % of all executions the GA finds a solution which is within 1 % from the global optimum. This performance compares favorably with one of the very best deterministic heuristics for SPG as well as with an earlier GA by Kapsalis et al. Performance is also compared to that of branch-and-cut algorithms by Lucena and Beasley and by Chopra et al. While the runtimes of these algorithms varies extremely and prevents the solution of some of the problem instances considered, the GA is capable of generating a near-optimal solution for all problems within a moderate amount of time.

We therefore conclude the following: In cases where a globally optimal solution is absolutely required, the size of the given problem is not too big and runtime is not important, one of the branch-and-cut algorithms are preferable. On the other hand, if a near-optimal solution is sufficient, or the problem is very large or a moderate runtime limit is needed, the GA presented here is the best choice of the possibilities considered.

Acknowledgement

The author wishes to thank Pranki Mazumder, University of Michigan, who supervised this work in its initial phase when the author was at University of Michigan. Also thanks to Jens Clausen and Pawel Winter, Copenhagen University, for pointing out possible improvements of an earlier version of the algorithm and for suggesting a suitable strategy for performance evaluation. Finally thanks to Peter Miller-Nielsen, Ole Caprani and Holger Arup, Aarhus University, for several useful discussions and suggestions concerning this work.

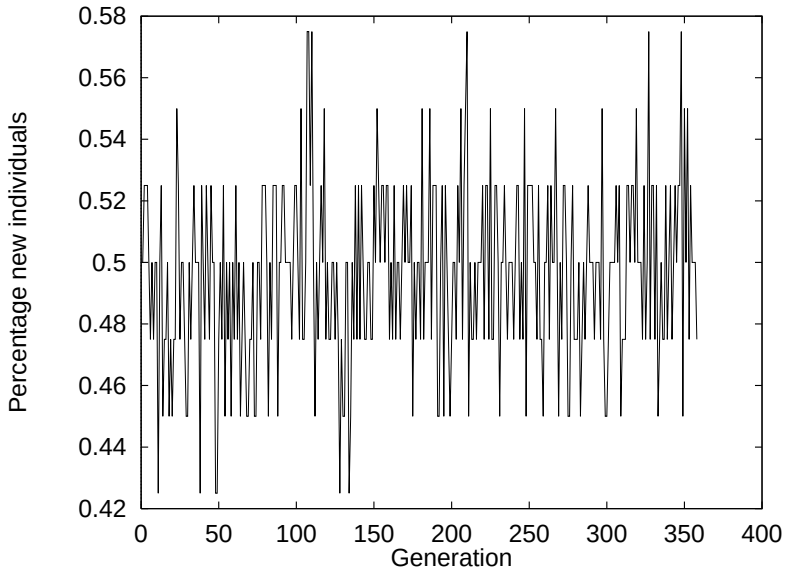

Figure C.10: *Percentage of new individuals in the population as a function of generation number.*

C.5 Future Work

The work presented here can be continued in at least three directions:

1) Performance improvement: As discussed in Section C.3 the main idea of the GA presented is the application of the DNF for interpretation of bitstrings. In contrast, the genetic operators for crossover, mutation and inversion are all standard. They are characterized by being very simple and blind in the sense that they do not utilize knowledge of the application domain in any way. The same is true for the hill-climber. One frequently used way of improving the performance of a GA is to apply more advanced genetic operators and/or operators exploiting application specific knowledge [12]. It is therefore likely that the performance of the GA presented here can be further improved by applying such techniques.

2) Other graph types: An obvious weakness of the test-suite used in this work is that all graphs are sparse and randomly generated. It remains to be seen how the GA performs on e.g. dense graphs, rectilinear graphs, non-random graphs arising in real-world applications, etc.

3) Contributions to performance: To obtain a detailed understanding of the reasons for the success of the algorithm it would be interesting to investigate how the various components of the algorithm contribute to the overall performance. What is the individual effect on solution quality and runtime caused by e.g. the decoding strategy, the inversion operator, the search space reduction or the initial graph reductions ?

However, the used stop criteria reflects a priority of solution quality as being more important than runtime.

Fig. C.9 shows for each generation the standard deviation of cost in the population. From a value of 19.2 in generation 0, the standard deviation decreases within 10 generations to about 2.0 and then stays at that level throughout the optimization process.

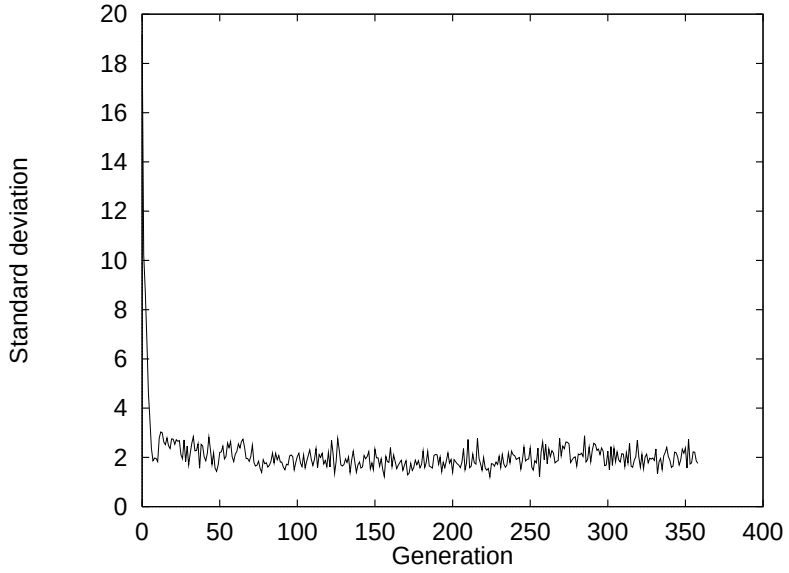

Figure C.9: *Standard deviation of cost as a function of generation number.*

As described in Section C.3.1 each generation is initiated by the generation of M offspring individuals. From the total of $2M$ individuals the best M individuals are then kept as members of the new population while the rest are discarded. Fig. C.10 shows for each generation the percentage of individuals in the newly created population which has just emerged as results of crossover. The percentage of newly generated individuals is very stable around 50. The important thing to note is that the fraction of new individuals do not decrease with time but is constant also into the late phase of the process. In other words, throughout the process half the individuals generated by the crossover operator are better than some other individual already in the population. This confirms the role of crossover as the most important of the genetic operators.

believe that the main reason for the performance gap between GARUS and the GA presented here is the different decoding strategy and consequently, the different cost evaluation strategy.

C.4.6 Typical Behavior

The progress of the typical optimization process is illustrated by Figures C.8, C.9 and C.10, which stems from a sample execution of the GA with graph D15 as input. It should be emphasized that although the graphs stems from a specific single run, the picture they give is very typical.

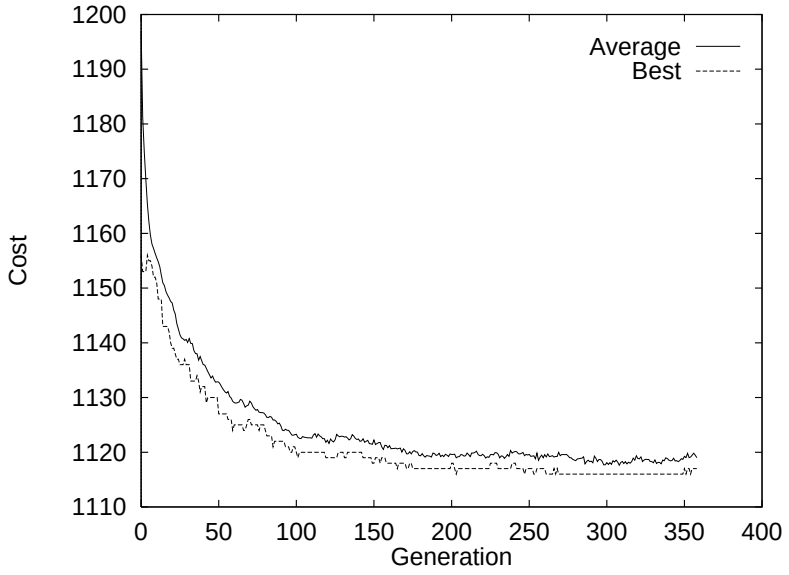

Figure C.8: *Cost of average and best individual as functions of generation number.*

For each generation, the top graph of Fig. C.8 indicates the average cost of the individuals in the population at that time, while the bottom graph indicates the cost of the current best individual. Initially, the average cost is 1,197 and the best is 1,156. The global optimum of 1,116 is obtained first time in generation 203, and the algorithm terminates after 358 generations. Note that improvement is very rapid during the first part of the process. Then it levels out and further improvement is obtained only slowly. As mentioned in Section C.3.1 the best as well as the average cost are parts of the stop criteria. If only the cost of the best solution were considered, the process would have terminated after generation 253, corresponding to a 29 % reduction of the runtime.

As problem size increases through the classes B, C, D and E the above observations become increasingly pronounced. If only class B graphs are considered, it is difficult to make any distinctions regarding performance of the algorithms. These examples appears to be too simple.

C.4.5 Comparison with Kapsalis' Algorithm

In this Section the GA by Kapsalis, Rayward-Smith and Smith [18] is denoted GAKRSS. As mentioned in Section C.1 GAKRSS differs from the GA presented here in a number of ways. Among other things, neither an inversion operator nor a hill-climber is applied in GAKRSS. However, the most significant differences concerns the decoder and the cost computation. In GAKRSS a genotype is a bitstring of length n in which the i'th bit indicates if the i'th vertex is part of the phenotype tree. To assure that every tree spans W each genotype is xor'ed with the fixed string specifying W. Hence, the encoding is very similar to our encoding. However, the interpretation of a genotype is very different. Assume a genotype specifies the vertex set Z, $W \subseteq Z \subseteq V$. The corresponding graph is then computed as the subgraph G_Z of G induced by Z. In general G_Z is not connected. Assume it consists of $k \geq 1$ components. The cost of a solution is defined as the sum of the cost of a minimum spanning tree for each component plus a penalty term which grows linearly with k.

Computational results are given only for the class B graphs from the OR-Library. The solution quality obtained for each graph is reported as the best result of five runs. For each graph *some* parameter setting of GAKRSS has been found with which the global optimum is found in five runs. However, the parameter setting varies with the problems given. When fixing the parameter setting for all graphs, GAKRSS finds the global optimum in approximately 70 % of all runs and the worst result generated is 7.3 % above the global optimum.

All experiments with GAKRSS are run on a Apple Mac IIfx. No total runtimes are given. Instead the time spend until the best solution found appears first time, referred to as Last Improvement Time (LIT), is measured. It is not clear exactly which stop criteria is used, i.e., how long the algorithm takes to terminate beyond LIT. For many of the graphs, the average LIT is in the range from 200 to 2,000 secs. There is a time limit of 4,000 secs. for a complete execution.

GAKRSS is clearly inferior to each of the other algorithms considered in this paper, both with respect to solution quality as well as runtime. We

For the class E examples, SPH-I finds optimum for 4 of the 20 graphs, while the GA finds the optimum for 11 of these graphs. $\Delta C_{worst} \leq \Delta C_{sph}$ holds for all but one graph in classes B, C and D, and in class E we have $\Delta C_{ga} \leq \Delta C_{sph}$ for all graphs. In other words, with a single exception even the worst results generated by the GA are equal to or better than the result generated by SPH-I. Furthermore, for the graphs where both SPH-I and the average execution of the GA fails to find the global optimum, the expected relative error ratio ΔC_{avg} of the GA is often an order of magnitude better than the error ratio ΔC_{sph} of SPH-I.

	Error Ratio		
Algorithm	= 0 %	< 0.5 %	< 1.0 %
SPH-I	48.7	66.7	70.5
GA	77.1	86.7	92.6

Table C.1: *Summary of solution qualities obtained by the GA and SPH-I.*

Table C.1 summarizes the solution qualities obtained by the GA and the SPH-I. These figures are based on the results of all 600 executions of the GA and all 78 executions of SPH-I performed in total. For each algorithm Table C.1 gives the accumulated percentage of runs which gave a result within the stated relative error from optimum. E.g., 66.7 % of all executions of the SPH-I gave a result which was less than 0.5 % from the optimum solution. When computing the values listed for the GA the results for the class E examples have been weighted by a factor of 10 to compensate for the imbalance in the number of executions for each graph.

The results regarding runtimes can be summarized in three main points:

- The GA is capable of finding a high-quality solution for *all* graphs considered in a moderate amount of time. This is not the case for any of the two branch-and-cut algorithms or for SPH-I.
- In most cases the runtime of the GA is very similar to that of SPH-I. In a few cases the GA is significantly faster than SPH-I, while the opposite is never the case.
- The variation of the runtime of the GA is very small compared to the variation observed for the branch-and-cut algorithms as well as SPH-I. As a consequence, the branch-and-cut algorithms are significantly faster than both the GA and SPH-I for some examples, while they are significantly slower on other examples.

Table C.12 lists the solution qualities obtained by the GA and the SPHI together with the runtimes of all algorithms considered. Due to the extensive runtimes required for the graphs in this class, the GA was executed only once for each example. C_{ga} denotes the cost obtained by the GA, ΔC_{ga} is the relative error of the solution found by the GA and the time spend by algorithm is denoted by T_{ga}. Hence, C_{ga} and T_{ga} can be considered estimates of C_{avg} and T_{avg}, respectively.

It should be noted that the listed value of C_{opt} for E18 may not be the global optimum, but according to the information in ORLibrary it is the best known solution as found by Beasley's algorithm [3]. The optimum for this graph was not found within a cpu-limit of 21,600 secs. on the Cray X-MP/48. Chopra et al. [5] also encountered problems with E18. No runtime is listed for this graph since the algorithm did not terminate within a cpu-limit of 10 days on the VAX 8700 [5]. Lucena and Beasley [23] does not report any results for graphs E16 through E20, and a reason is not given. However, considering the progression of runtimes for the graphs in classes C and D, it is reasonable to assume that the algorithm is unable to solve some of these problems in a reasonable amount of time.

SPHI exceeds the cpu-time limit of 50,000 secs. for graphs E13, E18, E19, E20, E14, E15, E18, E19 and E20. The estimated total time required by SPHI for these graphs varies from 81,000 secs. for E13 to 4.3×10^7 secs., or more than 16 months, for E20. Compared to the branch-and-cut algorithms and SPHI the runtimes of the GA are very moderate for all graphs with a maximum runtime of 2,105 secs. for E18. For most of the graphs for which SPHI terminates within the cpu-time limit the runtimes of the GA and SPHI are very similar. Regarding solution quality, SPHI finds the global optimum for 4 of the graphs and has a worst relative error ratio exceeding 9 % for E18. The GA finds optimum for 11 graphs and has a worst relative error ratio less than 2 %.

C.4.4.5 Summary of Results

This section summarizes the experimental results with respect to solution quality and runtime. When comparing the solution quality obtained by the GA to that obtained by SPHI for all graphs in classes B, C and D the following can be observed. Of a total of 58 graphs, SPHI finds the global optimal solution for 34 graphs, while the GA finds optimum 10 times out of 10 for 43 graphs and at least one time of 10 for 55 graphs.

vertex degree increases.

On the class D graphs SPHI finds optimum for 7 of the graphs, while the GA finds the optimum at least once for 17 graphs and every time for 13 graphs. SPHI has relative errors exceeding 2 % for 5 graphs while that only happens for the GA on graph D-18. For all graphs we have $C_{worst} \leq C_{sph}$ and $C_{worst} < C_{sph}$ holds for 13 graphs.

On this class of problems the runtimes for both branch-and-cut algorithms varies by three orders of magnitude and are as high as in the 20-30,000 secs. range corresponding to 2-3 days of computation. The runtimes of SPHI now also varies significantly. For practical reasons it became necessary to introduce a cpu-time limit of 50,000 secs. for this algorithm on graphs from classes D and E. When SPHI did not complete its computation within this limit, it was terminated and the best solution found so far was used. This happened for graphs D-19 and D-20. For these graphs the total time needed by SPHI is estimated to be 95,000 secs and 679,000 secs., respectively. These estimates can be considered to be quite accurate since they are based on measurements of the cpu-time spend for each pair of vertices x, $y \in V$, cf. Fig. C.7, which is then scaled with the relative number of vertex pairs not yet considered at the time the cpu-limit is exceeded. The average runtime of the GA varies from 504 secs. for D-5 to 3,441 secs. for D-19, i.e., by a factor of 7. This variation is small compared to the variation of the other algorithms considered. For graphs D-8, D-9, D-10, D-13, D-14, D-15, D-18, D-19 and D-20 the GA is on average an order of magnitude faster than SPHI while for the remaining graphs the runtimes of these algorithms are comparable.

C.4.4.4 The E Graphs

For the graphs from class E the effect of graph reductions follows a pattern which coincides perfectly with the patterns observed for classes C and D. Even after reductions the search space sizes for the class E graphs are enormous. Using the bound

$$\mathcal{S}(n, m) > \binom{n-m}{k} \geq \left(\frac{n-m-k+1}{k}\right)^k$$

where $k = \min(m-2, n-m)$ reveals that a number of graphs in this class has search spaces exceeding 10^{100} points. Especially, the search space for E-13 exceeds 10^{231} points and for E-18 it exceeds 10^{242} points. These bounds are computed after graph reductions have been performed.

However, as the average vertex degree increases, the effect of reductions of types a and b (see Section C.3.2) decreases significantly. When m is small, the effect of reductions of type d is also very limited, as can be seen by the results for C11, C12, C16 and C17. The obtained reduction in search space sizes for these problems are negligible. The effect of reductions of type c increases with the number of edges. For C16 through C20 about two thirds of all edges are eliminated by graph reductions, mainly of type c. However, since the CA operates in terms of shortest paths, minimum spanning trees, etc., the number of edges are not that important for the performance of the algorithm.

Table C.6 shows that the CA finds the global optimum at least once for all examples and every time for 12 of the graphs, while SPH-I finds the optimum for 10 of the graphs. When neither the average CA run nor the SPH-I finds the global optimum, ΔC_{avg} is often an order of magnitude better than ΔC_{sph}. This is the case for C3, C4, C9, C14, C18 and C19. For C18 and C19 the solutions produced by SPH-I are very poor with errors in the 6 - 7 % range. The results for C16 are in direct contrast to all other results. While the SPH-I finds optimum, the CA encounters severe problems. In 7 of 10 runs it misses the global optimum value of 11 and outputs a tree of cost 12. This corresponds to a huge relative error ΔC_{worst} of 9.09 %.

In Table C.7 and subsequent tables T_{bc1} denotes the runtime of the branch-and-cut algorithm by Chopra et al. [5]. Depending on the problem, the runtime for both branch-and-cut algorithms varies extremely. Chopra's algorithm spans from 10 secs. for C16 to more than 45,000 secs. for C18, while Lucena's algorithm varies from 5 secs. for C5 to more than 20,000 secs. for C18. As a consequence, the branch-and-cut algorithms are significantly faster than both the CA and SPH-I for some graphs and significantly slower for others. The runtimes of the CA and the SPH-I are similar for most graphs, although the CA is significantly faster for graphs C15, C19 and C20. The time variation T_{σ} of the CA is relatively small.

C.4.4.3 The D Graphs

The effect of graph reductions on the class D graphs shows a pattern similar to that observed for the C graphs, although now the pattern is even clearer. Most graphs are reduced significantly, note especially D5. The effect of reductions decreases as m decreases and as the average

reduced to the degenerate graph consisting of a single vertex only, which means that the optimal solution is found solely by performing graph reductions.

Table C.3 compares the solution quality obtained by the GA to the globally optimal solutions as well as to the solutions found by SPHI. C_{opt} is the global optimum and C_{sph} is the solution found by SPHI. C_{best}, C_{avg} and C_{worst} is the best, average and worst result produced by the GA in the 10 runs, while C_{σ} denotes the standard deviation of the 10 cost values. $\Delta C_{sph} = 100(C_{sph} / C_{opt} - 1)$ is the relative error in percent of the solution found by SPHI compared to the optimum solution. Similarly, $\Delta C_{avg} = 100(C_{avg} / C_{opt} - 1)$ denotes the average error of the solutions found by the GA, and $\Delta C_{worst} = 100(C_{worst} / C_{opt} - 1)$ is the worst error produced by the GA. Finally, N_{ga} denotes the number of the 10 runs which did not find the global optimum. This notation is also used in the following sections.

As can be seen, the GA finds the global optimum for all examples in every execution. SPHI performs similarly for all graphs except B13, for which it has a 1.82 % overhead.

Table C.4 compares the runtimes of the GA with that of SPHI and the branch-and-cut algorithm by Lucena and Beasley [23]. T_{bc2} denotes the runtimes of the latter algorithm and T_{sph} is the time of SPHI. The average time spent by the GA is denoted T_{avg} while T_{σ} denotes the standard deviation of the time for the 10 runs. Chopra et al. [5] gives no computational results for these graphs. It can be seen that all runtimes are very small and within the accuracy of these measurements it is difficult to draw any conclusions regarding differences in speed for the different algorithms.

The fact that all three algorithms finds optimal solutions (except for SPHI on B13) in a very short time suggests that these examples are simply too small to facilitate any distinction of performance of the algorithms. For several of the graphs the search spaces after graph reductions are indeed very small and the largest search space is that of B17 with less than 10^9 points, which is not that much for a combinatorial optimization problem.

C.4.4.2 The C Graphs

From Table C.5 it can be seen that the graph reductions are also very effective on most graphs in the C class. Note especially graph C-5 which after reductions has a search space size of only approximately 10^6 points.

The GA has been executed 10 times for each example in the B, C and D classes. Solution quality is then evaluated in terms of best, average and worst results produced. However, due to runtime requirements the GA was only executed once for each of the examples in class E. The parameter settings are $M = 40$, $S = 50$, $p_{mut} = 0.005$ and $p_{inv} = 0.1$. These values are used for all executions, i.e., no problem specific tuning has been made. As mentioned in Section C.1 fixed parameter values are of major importance from a practical point of view.

The GA as well as SPH-I are implemented in the C programming language. For both algorithms, examples from classes B, C and D are executed on a Sun Sparc IPX workstation having 32 MB RAM. These examples require at most 10 MB of memory. For the class E examples, the memory requirement is about 58 MB. Therefore, for these examples the GA as well as SPH-I are executed on a DEC VAXstation 5000-240 workstation having 128 MB RAM.

The branch-and-cut algorithm by Lucena and Beasley [23] is a further development of the algorithm presented in [3], but instead of using a Cray, it is now executed on a Sun Sparc 2 workstation. This machine is roughly as fast as the Sun Sparc IPX, but probably somewhat slower than the DEC VAXstation 5000-240. Chopra et al.'s algorithm [5] is executed on a VAX 8700 which is at most as fast as the other machines. When comparing absolute runtimes in Section C.4.4 the reader should keep these differences regarding the used hardware in mind. However, the runtime variations caused by the different machines are insignificant compared to the variations caused by different problem instances when considering a specific algorithm.

C.4.4 Results

In the following sections the detailed experimental results for all four problem classes are commented. The tables referenced can be found in Section C.7. A summary and conclusion of the results are given in Section C.4.4.5.

C.4.4.1 The B Graphs

Table C.2 lists the characteristics of the problems in class B before and after the graph reductions of Section C.3.2 are performed. The reductions significantly impacts all graphs. Especially, graphs B1, B3 and B9 are

due to the fact that all distances have been precomputed. Since $O(m^2)$ candidate solution trees T are computed, the total runtime of SPH-I becomes $O(n^3 + m^4 n)$.

```
graphReductions();
c(T_SPH-I) = ∞;
∀ x, y ∈ W, x ≠ y do
    T = G_xy;
    Q = W ∩ V_xy;
    while W \ Q ≠ ∅ do
        find a vertex z ∈ W \ Q closest to a vertex in T;
        add to T a shortest path from T to z;
        Q = Q ∪ { z };
    if c(T) < c(T_SPH) then T_SPH = T;
output T_SPH;
```

Figure C.7: *Outline of SPH-I.*

C.4.3 Experimental Method

The GA is evaluated by four kinds of comparisons:

- The solution quality obtained is compared to the global optimum.
- The absolute runtime is compared to that of two distinct branch-and-cut algorithms by Lucena and Beasley [23] and Chopra, Gorres and Rao [5].
- Solution quality and absolute runtime is compared to that of SPH-I.
- Comparison with the GA by Kapsalis et al. [18].

The branch-and-cut algorithms are guaranteed to find the global optimum. However, runtime may be unacceptable for some problem instances or may even prevent some problems from being solved. It is therefore of interest to investigate if a near-optimal solution can be found for *all* problems by using a moderate amount of time.

For a given graph, the size of the search space $\mathcal{S}(n, m)$ to be explored by the GA is

$$\mathcal{S}(n, m) = \sum_{i=0}^{k} \binom{n-m}{i}$$

where $k = \min(m-2, n-m)$, since this is the number of possible distinct choices of the Steiner vertices. Some of the problem instances considered represents extremely large search spaces, as will be seen in Section C.4.4.4. However, as mentioned in Section C.3.7, the corresponding phenotype spaces are smaller.

C.4.2 Iterated Shortest Path Heuristic (SPHI)

As mentioned in Section C.3.4 a comparative study of the deterministic heuristics SPH, DNH and ADH has been made by Winter and Smith [31]. Several variants of these heuristics, especially a number of repetitive variants of SPH, are also considered in the study. The ADH is in general considered to be one of the best deterministic heuristics, which is also confirmed by the investigation in [31]. However, the results also reveals that some of the repetitive variants of SPH consistently outperform ADH with respect to result quality. Furthermore, by applying initial graph reductions the runtime of the repetitive SPH variants can be made comparable to that of the other heuristics. One of the specific conclusions in [31] is that on the largest random graphs considered, the repetitive SPH variant denoted SPHZZ outperforms all other heuristics. Therefore, this heuristic has been chosen for comparison with the GA.

Fig. C.7 outlines our implementation of SPHZZ, denoted by SPHI. It starts by computing $D(G)$ and performing graph reductions as described in Section C.3.2. For given vertices x and y, $G_{xy} = (V_{xy}, E_{xy})$ denotes the subgraph of G corresponding to the shortest path between x and y. In each iteration of the outer loop a tree T is build which spans all vertices in W. T is initialized with a shortest path between two of the vertices to be spanned, and T is then extended by repeated addition of a shortest path to a closest, not yet connected vertex. This scheme is tried for all possible initializations of T, and the algorithm outputs the best such tree obtained.

As described in Section C.3.2 routine $graphReductions$ requires time $O(n^3)$. The construction of each candidate solution T takes time $O(m^2 n)$ since the "while" loop is iterated $O(m)$ times and it takes time $O(mn)$ to find each z vertex and extend T with a shortest path to it. This is

in Step 3 of the decoding process is almost always a tree, and as a consequence, Step 4 is rarely executed. Therefore, the true bottleneck of the algorithm is the MST computation performed in Step 2 of the decoding, which requires time $O(m \quad ^2)$.

C.4 Experiments

This section describes the experimental method applied and the results obtained. Characteristics of the test examples used are given in Section C.4.1. The deterministic heuristic SPH I used for comparison is described in Section C.4.2 and Section C.4.3 describes the chosen method for performing the comparative experiments. The results are reported and discussed in Section C.4.4. As mentioned in Section C.1 an earlier GA for SPG has been developed by Kapsalis et al. and a comparison to that algorithm is presented in Section C.4.5. Finally, Section C.4.6 describes the typical behavior of the GA during an optimization process.

C.4.1 Test Examples

The algorithm is tested on all 78 SPG instances from the OR-Library [4]. According to their size, these graphs are divided into four classes denoted by B, C, D and E. All graphs are generated at random subject only to the connectivity constraint, that is, the topology is random and the vertices to be spanned are selected at random. Every edge cost is a random integer in the interval [1, 10]. In class B each graph has n equal to 50, 75 or 100. The value of m is either $n / 6$, $n / 4$ or $n / 2$ and the average vertex degree is either 2.5 or 4. Since all combinations exists, class B consists of 18 graphs. Classes C, D and E consists of graphs with n equal to 500, 1,000 and 2,500 respectively. m equals 5, 10, $n / 6$, $n / 4$ or $n / 2$ and the average vertex degree is 2.5, 4, 10 or 50. Thus, each of the classes C, D and E consists of 20 graphs.

One of the main advantages of using this test-suite is that it facilitates comparison with the global optimal solutions. The global optima were first computed by J. E. Beasley who developed a branch-and-cut algorithm which was executed on a Cray X-MP/48 supercomputer [3].

A part of the ring is then selected at random and reversed. More specifically, two points x, $y \in \{0, 1, \ldots, r-1\}$, $x \neq y$, are selected at random. The operator then defines the reordering π' of g as [2]

$$\pi'((x+i) \bmod r) = \begin{cases} \pi((y-i) \bmod r) & \text{if } 0 \leq i \leq (y-x) \bmod r \\ \pi((x+i) \bmod r) & \text{otherwise} \end{cases}$$

for all $i = 0, 1, \ldots, r-1$. The inversion operator is illustrated in Fig. C.6.

Before inversion:

α : { (2, 0), (3, 1), (0, 1), (4, 0), (1, 0)}

After inversion with $x = 2$, $y = 0$:

α : { (0, 1), (3, 1), (2, 0), (1, 0), (4, 0)}

Figure C.6: *Illustration of the inversion operator with* $r = 5$.

C.3.8 Time Complexity

The filter routine described in Section C.3.4, the generation of each of the initial individuals, and the genetic operators crossover, mutate and invert each requires time $O(r) = O(n-m)$. The repeated decodings using DNH is the most expensive operation of the GA. Since knowledge of shortest paths is also required when performing some of the initial graph reductions, $D(G)$ is precomputed once and for all as mentioned in Section C.3.2. This reduces the time of Step 1 of DNH to $O(1)$ and as a consequence, one decoding can now be performed in time $O(mn \log(n\,m))$. Fitness computation requires $O(M \log M)$ to sort the individuals. In total, the GA's setup time is $O(n^3)$, and each generation requires time $O(M[n\,m \log(n\,m) + \log M])$.

Measurements reveals that the vast majority of the total runtime is spend on decodings. It also turns out that in practice the graph formed

[2] The definition of π' relies on the mathematical definition of modulo, in which the remainder is always non-negative.

Given individuals:

α : { (2, 1), (0, 1), (1, 0), (4, 0), (3, 0)}
β : { (1, 0), (2, 1), (4, 1), (3, 1), (0, 0)}

Step 1. Reordering β :

γ : { (2, 1), (0, 0), (1, 0), (4, 1), (3, 1)}

Step 2. Crossover with $x = 2$:

ϕ : { (2, 1), (0, 1), (1, 0), (4, 1), (3, 1)}
ψ : { (2, 1), (0, 0), (1, 0), (4, 0), (3, 0)}

Step 3. Only ϕ is subjected to the filter :

ϕ : { (2, 0), (0, 1), (1, 0), (4, 1), (3, 1)}

Figure C.5: *Illustration of the crossover operator with* $m = r = 5$.

C.3.7 Mutation and Inversion Operators

The mutation operator is extremely simple. Given a genotype g, the operator inverts each of the r bits in g with a small given probability p_{mut}. This scheme is called pointwise mutation. If necessary, g is then passed through the filter routine.

For a given phenotype, several equivalent genotypes usually exist. Since crossover is performed in terms of genotypes, the fitness of produced offspring depends on which of the possible genotypes are used as codings of the given phenotypes. The purpose of inversion is to optimize the performance of the crossover operator by rearranging the components within a given genotype, as explained in detail in [12, 16].

With a given probability p_{inv}, the inversion operator reorders the tuples of a given genotype g by altering its ordering π. This does not change the phenotype corresponding to g. To obtain a uniform probability of movement of all tuples, we consider the genotype to form a ring

C.3.5 Fitness Measure

Given a population $P = \{p_0, p_1, \ldots, p_{M-1}\}$, the routine *evaluate* of Fig. C.2 computes the fitness of each individual as follows. Let $C(p)$ be the cost of individual p, i.e. the cost of the Steiner tree represented by p, and assume that P is sorted so that $C(p_0) \geq C(p_1) \geq \ldots \geq C(p_{M-1})$. The fitness F of p_i is then computed as

$$F(p_i) = \frac{2i}{M-1} \qquad i = 0, 1, \ldots, M-1.$$

This fitness computation scheme is called *ranking* and is discussed in [29]. Controlling the variance of the fitness values is one of the frequent problems of GAs [12]. Ranking assures that the variance is constant throughout the optimization process. The specific scheme chosen here constantly gives the best individual twice the probability of the median individual of being selected for crossover.

C.3.6 Crossover Operator

Given two parent genotypes α and β, the crossover operator generates two offspring, ϕ and ψ. The parent genotypes are not altered by the operator. An example of crossover is shown in Fig. C.5. In this section, a superscript specifies which individual the marked property is a part of. Crossover consists of three steps:

1) One of the parents, say β, is chosen at random, and a copy γ of β is made. γ is then reordered so that it becomes homologous to α, that is, $\pi^\gamma = \pi^\alpha$.

2) Both offspring are given the same ordering as their parents, i.e., $\pi^\phi = \pi^\psi = \pi^\alpha$. Standard 1-point crossover is then performed [12, 16]: A crossover-point x is selected at random in $\{0, 1, \ldots, r-2\}$. The selection of Steiner vertices in ϕ and ψ is then defined by

$$i^\phi_{\pi(k)} = \begin{cases} i^\alpha_{\pi(k)} & \text{if } k \leq x \\ i^\gamma_{\pi(k)} & \text{if } k > x \end{cases}$$

and

$$i^\psi_{\pi(k)} = \begin{cases} i^\gamma_{\pi(k)} & \text{if } k \leq x \\ i^\alpha_{\pi(k)} & \text{if } k > x \end{cases}$$

where $\pi = \pi^\alpha$.

3) Finally, both ϕ and ψ are subjected to the filter routine, if necessary.

For a given instance of SPG, assume a fixed numbering $0, 1, \ldots, r-1$ of the vertices in $V \setminus W$. Let $\pi : \{0, 1, \ldots, r-1\} \mapsto \{0, 1, \ldots, r-1\}$ be a bijective mapping. A genotype is then a set of r tuples:

$$\{ (\pi(0), i_{\pi(0)}), (\pi(1), i_{\pi(1)}), \ldots, (\pi(r-1), i_{\pi(r-1)}) \}$$

where $i_k \in \{0, 1\}$, $k = 0, 1, \ldots, r-1$. The Steiner vertices $S \subseteq V \setminus W$ specified by the genotype is $S = \{ v_k \in V \mid i_k = 1 \}$. The Steiner tree in G corresponding to the genotype is the tree computed by DNH using the set $S \cup W$ as the vertices to be connected. In Step 5 of DNH every vertex $v \notin W$ of degree 1 is deleted. Note that the Steiner tree is independent of π. In other words, the Steiner tree constituting the phenotype of an individual does not change if the tuples in its genotype are reordered.

Any set of values of the i_k's in a genotype correspond to a valid phenotype. However, Lawler [21] has shown that a MST in $D(G)$ exists, which has at most $m-2$ Steiner vertices. This result relies on the fact that regardless of the edge cost function c, the edge costs in $D(G)$ always satisfy the triangle inequality. Hence, it is sufficient to consider only the subset of genotypes which satisfies $|S| \leq \min(m-2, r)$. To take advantage of this reduction of the search space, a routine *filter* has been defined, which given any genotype g enforces the satisfaction of $|S| \leq \min(m-2, r)$ by randomly selecting and clearing the necessary number of set bits.

When the initial random population has been generated, the filter is applied to each of the individuals. From then on, the search is limited to the restricted region by applying the filter to every new individual generated by one of the genetic operators.

It is important to note that the DNH is *not* chosen for use as decoder because it is a especially good heuristic in terms of result quality. In [31] the performance of DNH is compared to that of two other well-known polynomial time heuristics for the SPG: The Shortest Path Heuristic (SPH) by Takahashi and Matsuyama [27] and the Average Distance Heuristic (ADH) by Rayward-Smith and Clare [25]. With respect to result quality the DNH is clearly outperformed by both these heuristics. The reason to use DNH for decoding is first of all that it provides a way to interpret *any* set of selected vertices as a *valid* Steiner tree, and secondly, that it is relatively fast. The important advantage of considering valid Steiner trees only is that it eliminates the need for penalty terms in the cost measure, and thus avoids potential problems of assigning a suitable cost value to an invalid or incomplete solution.

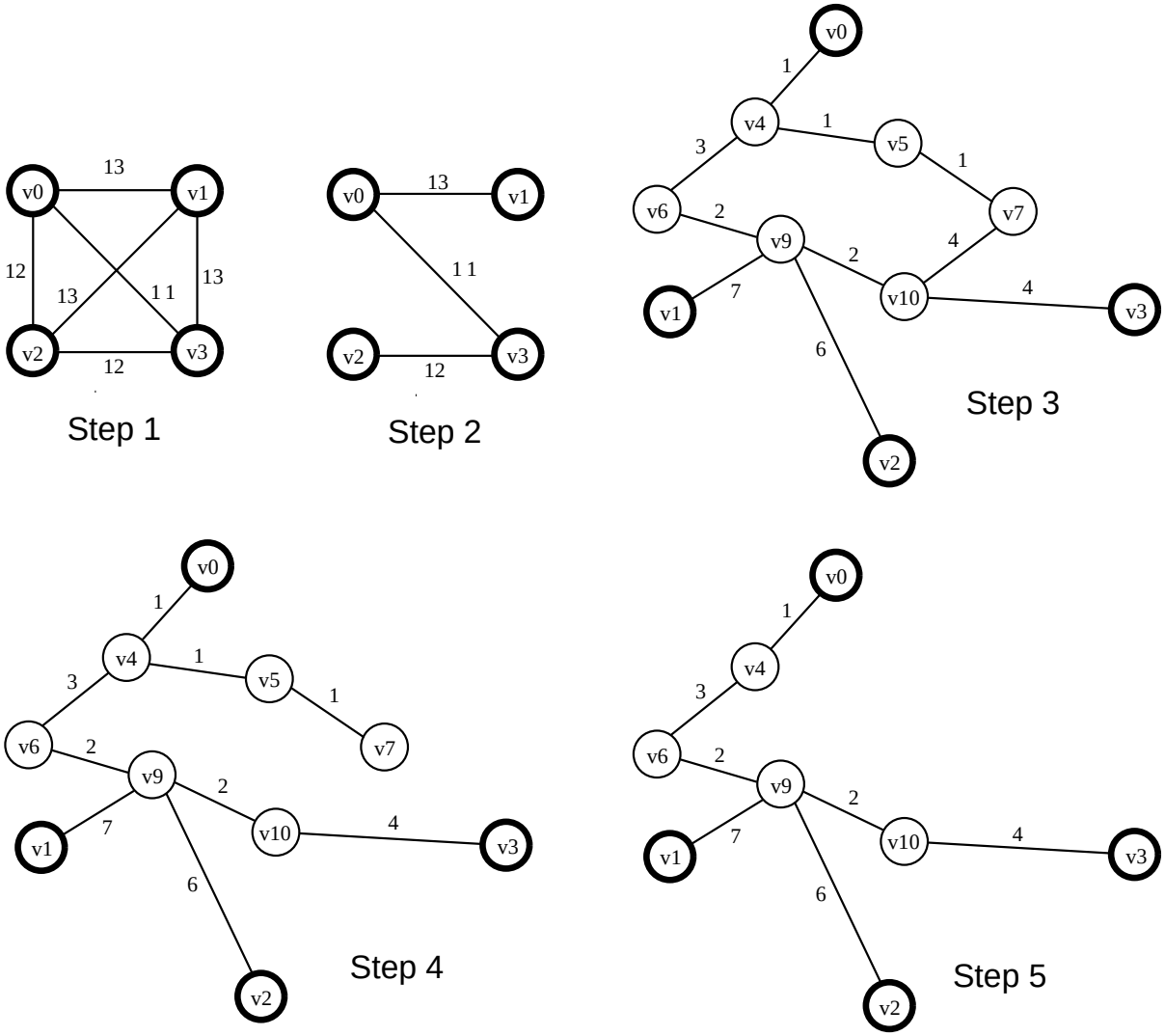

Figure C.4: *The steps of DNH given the input graph from Fig. C.1.*

C.3.4 Genotype and Decoder

The basic idea of the genotype and the associated decoder is the following. The genotype specifies a set of selected Steiner vertices. The decoder computes the corresponding phenotype by executing the DNH using the union of the selected Steiner vertices and W as the set of vertices to be spanned. The selected Steiner vertices are specified by a bitstring in which each bit corresponds to a specific vertex. If the bit is set, the vertex is selected. For reasons to be discussed in Section C.3.7, we need the genotype to be independent of the ordering of the bits in the string. This is obtained by associating with each bit a tag which identifies the vertex specified by that bit.

Specifically, the genotype and the decoder can be described as follows.

C.3.3 Distance Network Heuristic (DNH)

The key point in designing any GA is the design of a suitable genotype of an individual together with its interpretation, the decoder. The genetic encoding developed here is based on use of the Distance Network Heuristic (DNH), a deterministic heuristic for the SPG developed by Kou et al. [20]. Therefore, before proceeding by presenting the genotype and the decoder, the DNH is described.

Given a graph $G = (V, E)$, a cost function c and a subset of vertices W in accordance with the definition of SPG in Section C.2, the DNH computes an approximation T_{DNH} to the MST for W in G in five steps:

1. Construct the subgraph G_1 of $D(G)$ induced by W.
2. Compute a MST T_1 of G_1.
3. Construct from T_1 the subgraph G_2 of G by substituting each edge in T_1 by the corresponding shortest path in G.
4. Compute a MST T_2 of G_2.
5. Compute T_{DNH} from T_2 by repeatedly deleting all vertices $v \in V \setminus W$ having $\deg(v) = 1$.

Any ties in Steps 2, 3 or 4 are broken arbitrarily. An example of how the DNH works is shown in Fig. C.4, given as input the graph G of Fig. C.1 and the subset $W = \{v_0, v_1, v_2, v_3\}$.

If $D(G)$ is not known, Step 1 of DNH requires time $O(mn^2)$ to compute shortest paths from each of the m vertices. Since G_1 is complete the MST in Step 2 is computed using Prim's algorithm requiring time $O(m^2)$. Each of the $m-1$ edges of T_1 may correspond to a path in G of up to $n-1$ edges. Hence, Step 3 requires time $O(mn)$ and Step 4 requires time $O(mn \log(n m))$ using Kruskal's algorithm [1]. The final step is done in time $O(n)$. Hence, if $D(G)$ is not known, Step 1 is the most expensive and gives the DNH a time complexity of $O(mn^2)$.

Routine *graphReductions* terminates when no reduction of any type succeeded for a complete iteration, i.e., when no reduction can reduce G further.

```
compute D(G);
repeat
    reductions(c);
    reductions(b);
    reductions(d);
    reductions(a);
until no improvement in one iteration;
```

Figure C.3: *Outline of routine graphReductions.*

To deduce the worst case time complexity of *graphReductions*, start by considering the maximum total time spend on reductions of type d. Due to the required update of $D(G)$ a single reduction requires time $O(n^2)$. Since vertices can be added to W when performing reductions of type a, $O(n)$ type d reductions are possible. Hence, the total time spend on type d reductions is $O(n^3)$. One execution of *reductions(x)* require at most time $O(n^2)$ when either $x \neq d$ or $x = d$ but no contraction is performed. Since each of the reductions a, b and d decreases the number of vertices by one, and since type c reductions are performed exhaustively in the sense that after executing *reductions(c)* no edge exist which can be removed by a type c reduction, at least one vertex must be removed in every second iteration of the "repeat" loop in *graphReductions*. Hence, there can be no more than $O(n)$ iterations. In total this gives routine *graphReductions* the time complexity $O(n^3)$.

Although it is not difficult to construct a graph for which none of the reductions performed by *graphReductions* applies, the routine has been observed to be very effective on many graphs, as will be seen in Section C.4.4. When applied to the graph of Fig. C.1, the result is the degenerate graph consisting of one vertex only, implying that a MST has been found. In general, especially reductions of type d has been observed to be very powerful when m is relatively large, which coincides with the results reported in [31].

d) Assume that $v \in W$ and denote the closest neighbor to v by $u \in V$, and the second-closest neighbor by $w \in V$. Since G is connected, u always exists. If w does not exist, assume $c(e_{vw}) = \infty$. Let z be a vertex in $W \setminus \{v\}$ which is closest to u. If $c(e_{vu}) + c(sp(u, z)) \leq c(e_{vw})$ then any MST must include e_{vu}. Therefore, G can be contracted along this edge. Note that $u \in W \Rightarrow z = u \Rightarrow c(sp(u, z)) = 0$ i.e., contraction can always be performed in this case.

To obtain the largest possible overall reduction of G, the above reductions are performed repeatedly as described below. Knowledge of the cost of a shortest path is required whenever a reduction of type c or d is performed. Shortest paths are also repeatedly needed by the GA as will become apparent in Section C.3.4. Therefore, the distance graph $D(G)$ is computed initially using Floyd's algorithm [1] which requires time $O(n^3)$. Whenever one of the above reductions are performed, $D(G)$ has to be dynamically updated. When representing $D(G)$ as an adjacency matrix the update is trivial for reductions of type a or b. It simply consists of deleting the row and column corresponding to the deleted vertex. Reductions of type c leaves $D(G)$ unchanged. However, for reductions of type d the update is slightly more involved. Whenever a contraction is performed, $D(G)$ is updated using an $O(n^2)$ algorithm by Dionne and Florian [6].

In [30, 31] the following reduction is also suggested along with the reductions described above: If $\max\{c(sp(v, u)), c(sp(v, w))\} < c(e_{uv})$, $e_{uv} \in E$ and $v \in W$, then no MST can include e_{uv}, which therefore can be deleted. However, in this case the required update of $D(G)$ has a worst case complexity of $O(n^3)$ using Dionne and Florian's algorithm [6]. I.e., the update could be as expensive as recomputing the entire distance graph, and for this reason this reduction is omitted.

When performing a sequence of reductions of the same type, the overall result depends on the chosen traversal of the graph, that is, the order in which reductions are tried out. Furthermore, reductions of distinct types are mutually dependent in the sense that performing all possible reductions of some type may allow new subsequent reductions of another type. It is not clear in which order reductions should be performed to obtain the overall best reduction of a given graph [31]. The arbitrarily chosen scheme for performing reductions in routine $graphReductions$ is shown in Fig. C.3. Routine $reductions(x)$ performs a single traversal of all vertices (or edges in the case of type c reductions) of G in an unspecified order and carries out a reduction of type x wherever possible.

c_1 and c_2. Routine *reduce* returns the M fittest of the given individuals, thereby keeping the population size constant. With a small probability p_{mut}, the *mutation* operator randomly changes each of the components, or *genes*, of its argument, as described in Section C.3.7. The genetic operator *invert*(p) alters the genotype of an individual p without altering the corresponding phenotype. As described in [12], the purpose of this operator is to optimize the relative positions of the genes of p with respect to the crossover operator. The inversion operator will be described in Section C.3.7. Routine *optimize*(s) performs simple local hill-climbing by executing a sequence of mutations on s, each of which improves the fitness of s. An exhaustive strategy is used so that when the routine has been executed, no single mutation exists, which can improve s further. The output of the algorithm is then the solution s.

C.3.2 Graph Reductions

Before the GA itself is executed an attempt to reduce the size of the given problem is performed using standard graph reduction techniques. Routine *graphReductions* of Fig. C.2 performs four kinds of rather simple reductions all of which are described in [30, 31]. More elaborate reductions as well as proofs of the correctness of the reductions used here can be found in [9]. Let e_{vw} denote the edge between vertices v and w, and let $sp(v, w) \subseteq E$ denote the shortest path between v and w. The four reductions used are:

a) Assume $deg(v) = 1$ and $e_{vw} \in E$. If $v \in W$ any MST must include e_{vw}. Hence, v and e_{vw} can be removed from G and w is added to W if it is not already there. If $v \in V \setminus W$, no MST can include e_{vw}, i.e. in this case v and e_{vw} can also be deleted.

b) If $v \in V \setminus W$, $deg(v) = 2$ and $e_{uv}, e_{vw} \in E$, then v, e_{uv} and e_{vw} can be deleted from G and replaced by a new edge between u and w of equivalent cost. More specifically, if $e_{uw} \notin E$ then $E = E \cup \{e_{uw}\}$ and $c(e_{uw}) = c(e_{uv}) + c(e_{vw})$. If there is an edge from u to w already, i.e., $e_{uw} \in E$, then $c(e_{uw}) = \min\{c(e_{uw}), c(e_{uv}) + c(e_{vw})\}$.

c) If $e_{vw} \in E$ and $c(e_{vw}) > c(sp(v, w))$ then no MST can include e_{vw}, which therefore can be deleted.

```
graphReductions();
generate(P_C);
evaluate(P_C);
s = bestOf(P_C);
repeat until stopCriteria():
    P_N = ∅;
    repeat M/2 times:
        select p_1 ∈ P_C, p_2 ∈ P_C;
        {c_1, c_2} = crossover(p_1, p_2);
        P_N = P_N ∪ {c_1, c_2};
    end;
    evaluate(P_C ∪ P_N);
    P_C = reduce(P_C ∪ P_N);
    ∀ p ∈ P_C: possiblyMutate(p);
    ∀ p ∈ P_C: possiblyInvert(p);
    evaluate(P_C);
    s = bestOf(P_C ∪ {s});
end;
optimize(s);
output s;
```

Figure C.2 *Outline of the algorithm*

putes the fitness of each of the given individuals, while *bestOf* finds the individual with the highest fitness. One execution of the outer "repeat" loop corresponds to the simulation of one generation. Throughout the simulation the number of individuals $M = |P_C|$ is kept constant. We keep track of the best individual s ever seen. Routine *stopCriteria* terminates the simulation when no improvement of the best or the average fitness has been observed for S consecutive generations, or when the algorithm has converged so that all individuals have the same fitness. Each generation is initiated by the formation of a set of offspring P_N of size M. The two mates p_1 and p_2 are selected from P_C independently of each other, and each mate is selected with a probability proportional to its fitness. The *crossover* routine described in Section C.3.6 generates two offspring

C.3 Description of the Algorithm

In this section the developed algorithm is described in detail. First an overview of the algorithm is given in Section C.3.1. Initially an attempt to reduce the size of a given problem is made by applying some graph reduction techniques described in Section C.3.2. The main idea of the GA is the application of the Distance Network Heuristic for interpretation of the representation manipulated by the genetic operators. This is discussed in Sections C.3.3 and C.3.4. Other components of the algorithm is described in Sections C.3.5, C.3.6 and C.3.7. Finally, the time complexity of the algorithm is discussed in Section C.3.8.

C.3.1 Overview

The concept of genetic algorithms, introduced by John Holland [16], is based on natural evolution. In nature, the individuals constituting a population adapt to the environment in which they live. The fittest individuals have the highest probability of survival and tend to increase in numbers, while the less fit individuals tend to die out. This *survival-of-the-fittest* Darwinian principle is the basic idea behind the GA.

The algorithm maintains a *population* of *individuals*, each of which corresponds to a specific solution to the optimization problem at hand. A measure of *fitness* defines the quality of an individual. Starting with a set of random individuals, a process of evolution is simulated. The main components of this process are *crossover*, which mimics propagation, and *mutation*, which mimics the random changes occurring in nature. After a number of *generations*, highly fit individuals will emerge corresponding to good solutions to the given optimization problem.

A *phenotype* is the physical appearance of an individual, while a *genotype* is the corresponding representation or genetic encoding of the individual. Crossover and mutation are performed in terms of genotypes, while fitness is defined in terms of phenotypes. For a given genotype, the corresponding phenotype is computed by a *decoder*. A good introduction to genetic algorithms is given in [12].

Fig. C.2 shows a template for the GA considered here. Before the GA itself is executed, routine *graphReductions* tries to reduce the size of the given problem as described in Section C.3.2. Then the initial current population P_C is constructed from randomly generated individuals by routine *generate*. Routine *evaluate* described in Section C.3.5 com-

The Steiner Problem in a Graph (SPG): Given a connected, undirected graph $G = (V, E)$, a positive edge cost function $c : E \mapsto \Re^+$, and a subset $W \subseteq V$, compute a connected subgraph $G' = (V', E')$ of G, such that $W \subseteq V'$ and such that $c(G')$ is minimal.

Any acyclic subgraph G' of G such that $W \subseteq V'$ is called a *Steiner Tree for W in G*. A solution G' with minimal cost is called a *Minimal Steiner Tree (MStT) for W in G*. The set $S \subseteq V \setminus W$ such that $V' = W \cup S$ is called the *Steiner vertices* of G'. Note the generality of this problem formulation. We do not require G to be planar, and we do not require c to satisfy the triangle inequality.

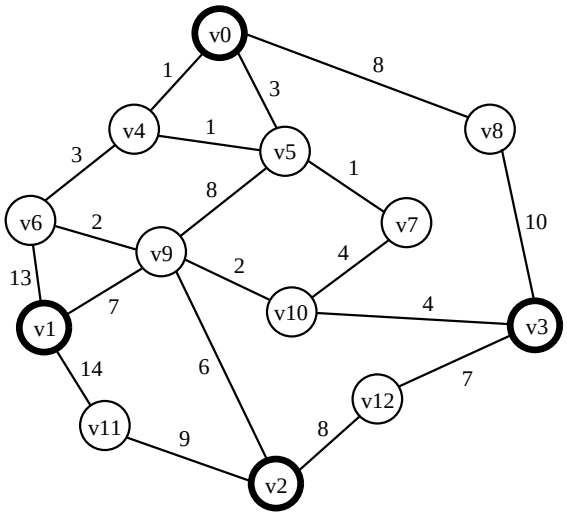

Figure C.1: *An example instance of the SPG. The highlighted vertices constitutes W.*

Throughout this paper, let $n = |V|$, $m = |W|$ and $r = n - m$. If $m = 2$, SPG reduces to the shortest path problem, which can be solved by e.g. Dijkstra's algorithm [22] in time $O(|E| \log n)$. If $m = n$, SPG is the Minimum Spanning Tree problem (MST), which can be solved in $O(n^2)$ time by e.g. Prim's algorithm [1]. However, if $2 < m < n$, SPG is in general NP-complete [1] [19].

[1] Some special graph topologies do exist, for which SPG can still be solved in polynomial time [30].

experimental results shows the following:

- The GA presented here clearly outperforms the GA in [18] with respect to solution quality as well as runtime.
- The solution quality obtained by our GA is always at least as good as that obtained by SPH, and often the error ratio is an order of magnitude better. Depending on the problem, the two algorithms either require similar amounts of runtime, or the GA is significantly faster.
- As opposed to the branch-and-cut algorithms, the GA is not guaranteed to find a global optimal solution. However, the experiments reveals that the GA do find the global optimum in more than 77 % of all runs and is within 1 % from optimum in more than 92 % of all runs. While the GA is capable of finding near-optimal solutions for *all* test examples in a moderate amount of time, the runtime of the branch-and-cut algorithms varies extremely and even prevent some of the largest problem instances from being solved.

The paper is organized as follows. A precise problem definition is given in Section C.2. Section C.3 presents a detailed description of the developed algorithm and discusses some of the main design decisions taken. The experimental method as well as detailed experimental results are given in Section C.4, and in Section C.5 possible directions for future work are suggested. Finally, Section C.6 concludes the paper.

C.2 Problem Definition

The graph terminology used in this paper is as in [1]. For a given graph $G = (V, E)$ and a subset $V' \subseteq V$, the *subgraph of G induced by V'* is a graph $G' = (V', E')$, such that 1) $E' \subseteq E$, 2) $(v_i, v_j) \in E' \Rightarrow v_i, v_j \in V'$, and 3) $[v_i, v_j \in V' \wedge (v_i, v_j) \in E] \Rightarrow (v_i, v_j) \in E'$. A graph is *complete* if it has an edge between every pair of vertices. The *distance graph* of G, denoted $D(G)$, is the complete graph having the same set V of vertices, in which the cost of each edge (v_i, v_j) equals the cost of the shortest path in G from v_i to v_j. For a given edge cost function $c : E \mapsto \Re$, the *cost of a graph G* is the sum of the cost of all edges of G, and is denoted by $c(G)$.
The problem considered can now be defined:

C.1 Introduction

The Steiner Problem in a Graph (SPG) is one of the classic problems of combinatorial optimization. Given a graph and a designated subset of the vertices, the task is to find a minimum-cost subgraph spanning the designated vertices. The SPG arises in a large variety of diverse optimization problems such as network design, multiprocessor scheduling and integrated circuit design [10, 28].

Numerous algorithms of various kinds have been developed for the SPG. Exact algorithms can be found in e.g. [2, 3, 5, 8, 13, 23, 26]. However, since the SPG is NP-complete [19] these algorithms have exponential worst case time complexities. Therefore, a significant research effort has been directed towards polynomial time heuristics, cf. e.g. [2, 20, 24, 25, 27, 31]. Simulated annealing has also been applied to SPG [7].

The Rectilinear Steiner Problem (RSP) is an important special case of SPG [14], which is still NP-complete [11]. While at least two genetic algorithms for RSP have been published [15, 17], we are aware of only one previous genetic algorithm (GA) for the SPG, developed by Kapsalis, Rayward-Smith and Smith [18].

The contribution of this paper is a new GA for the SPG which differs significantly from the approach of Kapsalis et al. [18] in a number of ways. While invalid solutions are allowed but penalized in [18], our approach is to enforce constraint satisfaction at all times, thereby eliminating the need for penalty terms in the cost function. Another major difference is our use of an inversion operator.

The performance evaluation strategies also differs significantly. While the parameter settings used in [18] varies from problem to problem, a fixed set of parameter values has been used for all results reported in this paper. From a practitioners point of view a stochastic algorithm is of limited use if it requires its parameters to be tuned every time a new problem instance is presented. Therefore we consider a fixed parameter setting to be of major importance.

The presented algorithm is tested on all SPG instances from the ORLibrary [4]. This test suite consists of randomly generated graphs with up to 2,500 vertices and 62,500 edges. The obtained performance is compared to that of the GA by Kapsalis et al. [18], an iterated version of the Shortest Path Heuristic called SPH-I, which is one of the very best deterministic heuristics [31], and two recent branch-and-cut algorithms by Lucena and Beasley [23] and Chopra, Gorres and Rao [5]. The

Abstract

A new Genetic Algorithm (GA) for the Steiner Problem in a Graph (SPG) is presented. The algorithm is based on a bitstring encoding. A bitstring specifies selected Steiner vertices and the corresponding Steiner tree is computed using the Distance Network Heuristic. This scheme ensures that every bitstring corresponds to a valid Steiner tree and thus eliminates the need for penalty terms in the cost function.

The GA is tested on all SPG instances from the ORLibrary of which the largest graphs have 2,500 vertices and 62,500 edges. When executed 10 times on each of 58 graph examples, the GA finds the global optimum at least once for 55 graphs and every time for 43 graphs. In total the GA finds the global optimum in 77 % of all program executions and is within 1 % from the global optimum in more than 92 % of all executions.

The performance is compared to that of two branch-and-cut algorithms and one of the very best deterministic heuristics, an iterated version of the Shortest Path Heuristic (SPHI). For all test examples but one, even the worst result ever found by the GA is equal to or better than the result of SPHI and in many cases the average error ratio of the GA is an order of magnitude better than that of SPHI. The runtime of the GA is moderate for all test examples. This is in contrast to SPHI as well as the branch-and-cut algorithms, for which the runtime in some cases are extremely high.

Appendix C

Computing Near-Optimal Solutions to the Steiner Problem in a Graph Using a Genetic Algorithm

This paper is available as technical report DAIMI PB-348, Computer Science Department, Aarhus University, February 1994. An earlier version of the algorithm was presented in H. Esbensen, P. Mazumder, "A Genetic Algorithm for the Steiner Problem in a Graph," *Proc. of The European Design and Test Conference*, pp. 402-406, 1994.

[10] David J. Strag, Paul T. Wasser, "Towards a Unified Thermodynamic Genetic Operator," *Proceedings of the 2nd International Conference on Genetic Algorithms*, pp. 116-122, July 1987.

[11] M. Upton, K. Sanii, S. Sugiyama, "Simulated Annealing Placement for Mixed Macro Cell and Standard Cell Layouts", *Proceedings of The International Workshop on Layout Synthesis*, Vol. 1, May 1990.

Bibliography

[1] Emile Aarts, Jan Korst, "Simulated Annealing and Boltzmann Machines," *John Wiley & Sons*, Chichester, 1989.

[2] M Beardslee, J. Burns, A Casotto, M Igusa, F. Romeo, A Sangiovanni-Vincentelli, "MISMAC: User's manual", Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, March 1990.

[3] Torsten Bosenick, Werner Ebeling, "Boltzmann-, Darwin-, Haeckel-Strategies in Optimization Problems," *Proceedings of the 1st Workshop on Parallel Problem Solving from Nature*, pp. 430-444, FRG, Oct. 1990, Springer-Verlag, 1991.

[4] W.M Dai, B. Eschermann, E. S Kuh, M Pedram, "Hierarchical Placement and Floorplanning in BEAR," *IEEE Transactions on Computer-Aided Design*, pp. 1335-1349, Vol. 8, No. 12, 1989.

[5] Henrik Esbensen, "A Genetic Algorithm for Macro Cell Placement," *Proceedings of The European Design Automation Conference*, pp. 52-57, FRG, Sept. 1992.

[6] D. E. Goldberg, "Genetic Algorithms in Search, Optimization, and Machine Learning," *Addison-Wesley*, 1989.

[7] J. H Holland, "Adaption in Natural and Artificial Systems," *University of Michigan Press*, Ann Arbor, MI, 1975.

[8] Hidetoshi Onodera, Y. Taniguchi, Keikichi Tamaru, "Branch-and-Bound Placement for Building Block Layout", *Proceedings of The 28th Design Automation Conference*, pp. 433-439, 1991.

[9] K. Shahookar, P. Mazumder, "VLSI Cell Placement Techniques," *ACM Computing Surveys*, Vol. 23, No. 2, 1991.

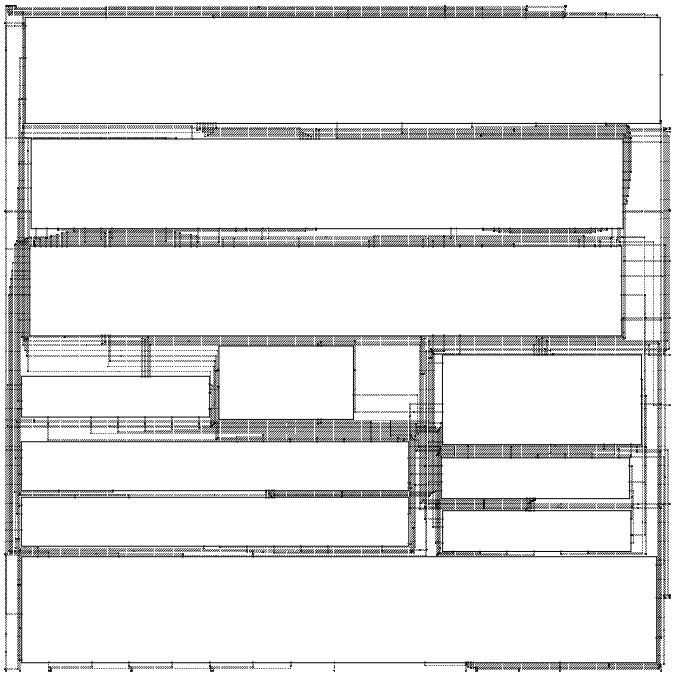

Figure B.19: *The result obtained for the Hp benchmark.*

B.7 Conclusion

This paper has presented a stochastic optimization algorithm, called SAGA, that combines the genetic algorithm with simulated annealing. The approach is application independent and adaptive. The performance of the unified algorithm on the macro-cell placement problem has been investigated. It is empirically shown that a mixture of GA and SA performs better than a pure GA on this problem. Furthermore, on MCNC placement benchmarks, we obtain layouts better than or comparable to previously published results by using a GA/SA mixture. The current implementation is not runtime competitive but significant improvements can be made. We therefore conclude that the approach presented is a very promising approach to macro-cell placement.

quality of the best completed layouts are compared to the best published results. The absolute area is core area in mm^2. To ease comparison, relative areas have also been computed by assigning the best result for each benchmark the relative area 1. The total interconnect length in mm and the total number of vias is also given.

Benchmark	System	Area		Route	
		relative	absolute	length	Vias
Apte	SAGA	1.000	53.58	489	647
	BBL [8]	1.009	54.05	460	-
	Seattle Silicon [11]	1.022	54.77	350	-
Xerox	Seattle Silicon [11]	1.000	25.79	601	1104
	BBL [8]	1.015	26.17	628	-
	SAGA	1.053	27.15	679	1379
	BEAR [4]	1.104	28.47	633	897
	MOSAICO [2]	1.125	29.01	650	1173
	VITAL [2]	1.230	31.17	866	1029
Hp	SAGA	1.000	11.81	261	675
	Seattle Silicon [11]	1.003	11.85	200	-
	BBL [8]	1.029	12.15	278	-

Table B.3: *Comparison of quality with other systems. All SAGA results listed are obtained using the mixed GA/SA strategy. A hyphen indicates that the value is not available.*

While SAGA is highly competitive with respect to solution quality, the current implementation of the algorithm requires significantly more runtime than the other systems listed in Table B.3. The BBL approach [8] is about 2-3 times faster and the Seattle Silicon approach [11] is about 10-20 times faster. However, there is a number of reasons why we expect that runtime can be improved significantly. First of all, in the current implementation the majority of the runtime is spend computing channel densities in a very inefficient manner. All density computations are done from scratch. Instead, by using a suitable datastructure, most channel densities could be computed much faster by a dynamic update from a previous, almost identical computation. Furthermore, due to the inherent parallelism of this kind of algorithm, a high speedup of a parallel version of the unified algorithm can be expected on any MIMD architecture.

[2] Referenced here as found in [4, 11].

are quite close to the optimum, suggesting that the room for improvement over the GA is small. However, for the Apte and Xerox benchmarks, the values of A_{best}, A_{avg} and A_{σ} are significantly improved by the mixed strategy, while for the Hp benchmark, no significant improvement can be observed. The improvement of estimated area obtained by the mixed strategy on the Xerox benchmark is illustrated in more detail in Fig. B.18. The results of each of the 40 runs of SAGA were grouped so that the i'th group (left to right on Fig. B.18) corresponds to areas (mm^2) in the interval $[27.5+0.3i, 27.5+0.3(i+1)[$. The height of a bar indicates the number of the 40 runs belonging to the group.

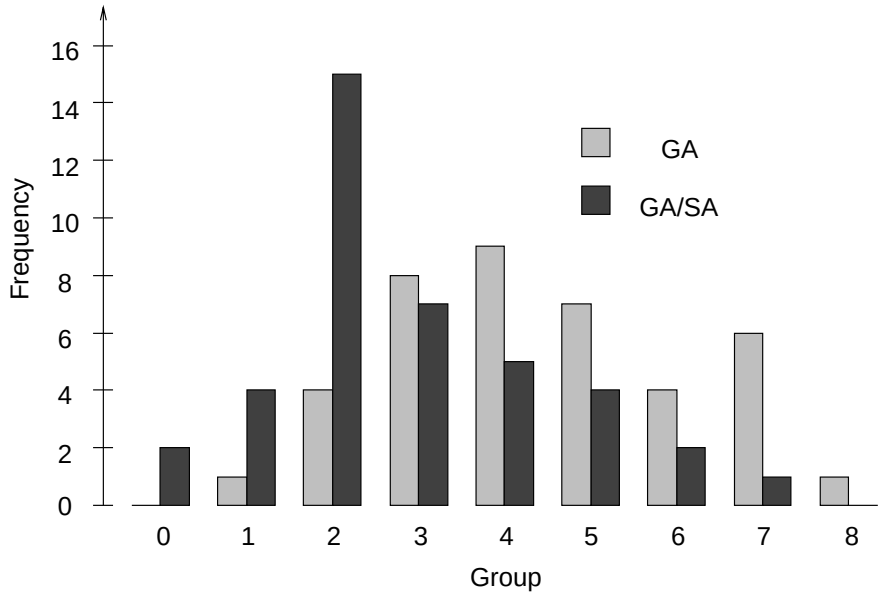

Figure B.18: *Performance comparison on Xerox.*

On average the GA/SA is about 40 % slower than the GA for the Xerox benchmark while for Apte and Hp the differences are less than 8 %. The relatively large standard deviations of runtime are caused by the conservative stop criterion $S = 200$. If an improvement is seen after e.g. 195 consecutive generations without any other improvements, the search is continued for at least another 200 generations, no matter how insignificant the improvement might have been.

B6.3 Comparison with other Systems

Ten of the 40 placements generated for each benchmark by SAGA, using the mixed GA/SA strategy, has been routed and compacted using Mosaico [2], which is part of the Octtools CAD framework. In Table B.3, the

B6.2 Comparing the GA with a Mixed Strategy

Using the benchmarks, the performance of a mixed GA/SA strategy has been compared to that of a pure GA. Consistent with the results of Section B6.1, the parameters $M_0 = 25$, $S = 200$, $p_{mut}^0 = 0.025$, and $p_{inv} = 0.05$ were used for both strategies. For the GA, $\mathcal{P} = 1 - 10^{-7}$, while for the GA/SA, $\beta = 0.7$, $\gamma = 1.4$, $R = 80$, $\mathcal{P} = 0.99$, $\alpha = 0.6$ and $\lambda = 1$.

The same parameter settings have been used for all three benchmarks, i.e., no problem-specific tuning has been made. For each benchmark and each set of parameters, SACA was executed 40 times. Table B.2 summarizes the results. A_{best} and A_{avg} denote the best and the average estimated area in mm^2, respectively. A_σ denotes the standard deviation. Since SACA minimizes estimated area as opposed to area after routing and compaction, the best comparison of the two optimization approaches is obtained by comparing estimated areas. T_{avg} denotes the average CPU-time in seconds and T_σ is the standard deviation of the CPU-time.

Benchmark	Quantity	GA	GA/SA
Apte	A_{best}	57.599	57.477
	A_{avg}	58.648	58.250
	A_σ	0.816	0.727
	T_{avg}	3,134	3,328
	T_σ	1,563	2,259
Xerox	A_{best}	27.891	27.554
	A_{avg}	28.961	28.521
	A_σ	0.533	0.486
	T_{avg}	9,354	13,192
	T_σ	3,908	2,568
Hp	A_{best}	12.805	12.803
	A_{avg}	13.310	13.294
	A_σ	0.357	0.369
	T_{avg}	3,311	3,070
	T_σ	1,353	1,499

Table B.2 *Comparison of the pure GA with a GA/SA mixture.*

The results reported in [5] approximately corresponds to what is here called the pure GA, and they are comparable to the best results published. It is therefore likely that the areas shown in Table B.2 obtained by the GA

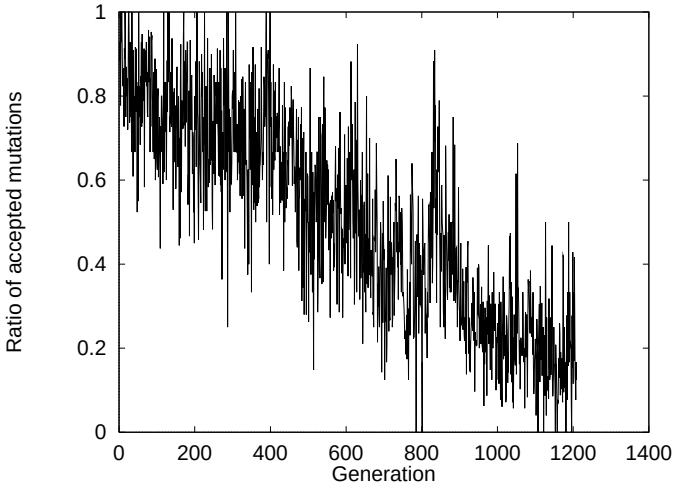

Figure B.16: *Ratio of attempted mutations which are actually accepted.*

$\alpha = \gamma = 1.0$, and the remaining parameters as before. In other words, it is the SA-controlled mutations that allows the population size to be reduced while at the same time maintaining convergence. On the other hand, fixing the population size ($\beta = 1.0$) while performing an increasing number of SA-controlled mutations leads to an ineffective process. In the late phase, many mutations will be attempted on a large number of individuals, but only few mutations will actually be accepted and performed.

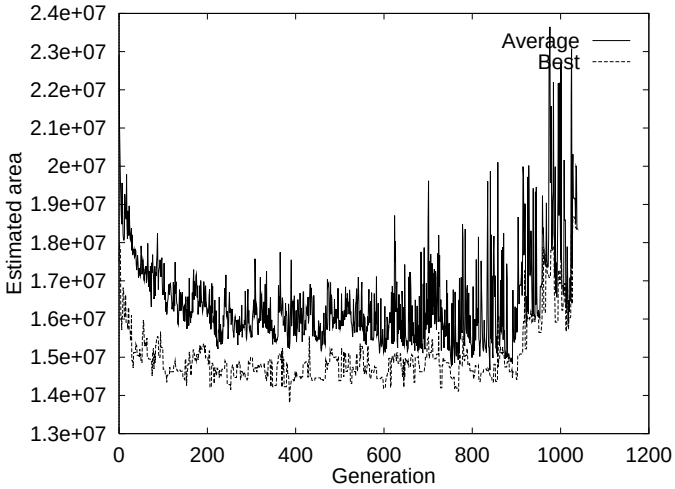

Figure B.17: *Effect of reducing the population size in a otherwise pure GA process.*

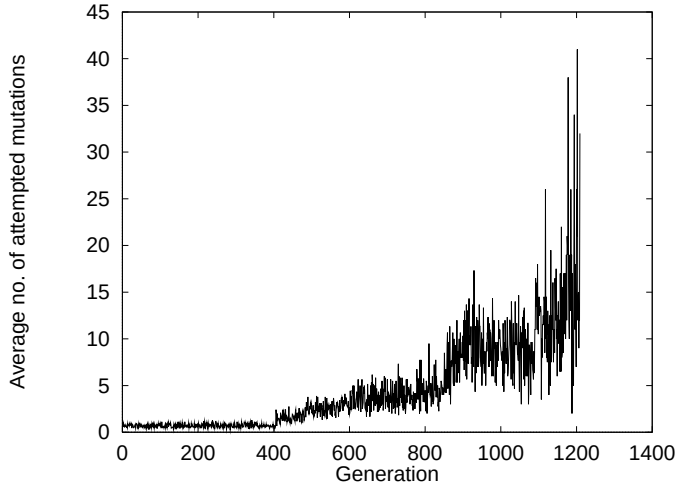

Figure B.14: *Average number of attempted mutations on each individual as a function of generation number.*

In the first phase of the optimization process only few mutations are attempted, and almost all of them are accepted. This resembles a pure GA process. In the final phase of the optimization, the probability of accepting cost-increasing mutations becomes small and the ratio of accepted mutations decreases to about 20 %.

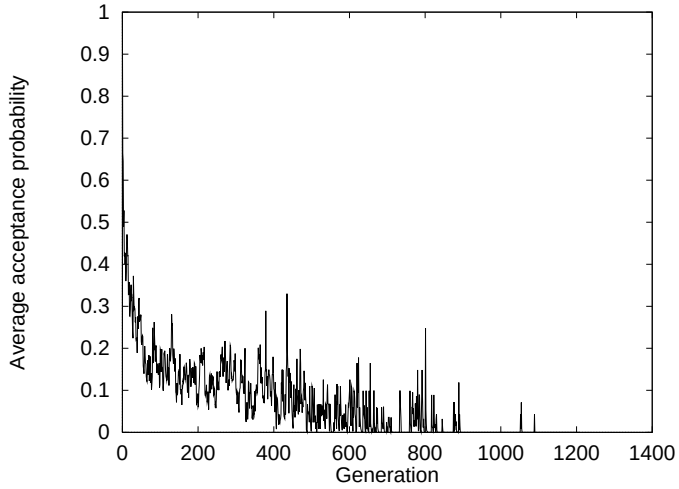

Figure B.15: *Average value of the acceptance probabilities P in the population.*

The decrease of the population size and the increase of the number of SA controlled mutations are both important components of the optimization process. Reducing the population size in an otherwise pure GA process will cause divergence, as illustrated in Fig. B.17. These graphs stem from a sample execution of SACA with the parameters $\mathcal{P} = 1 - 10^{-7}$,

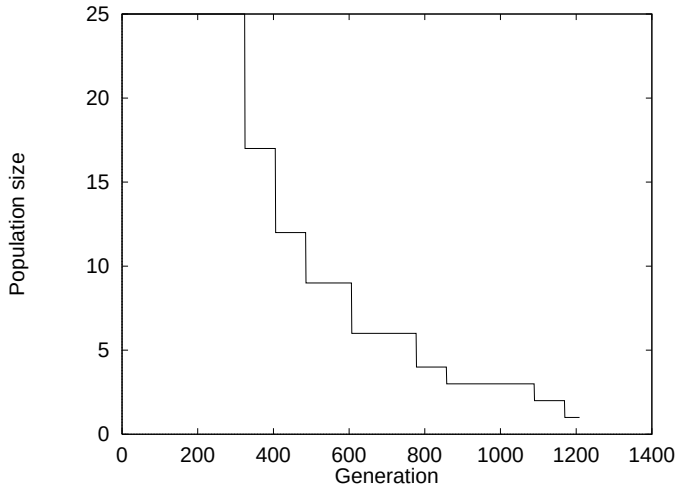

Figure B.12: *Population size as a function of generation number.*

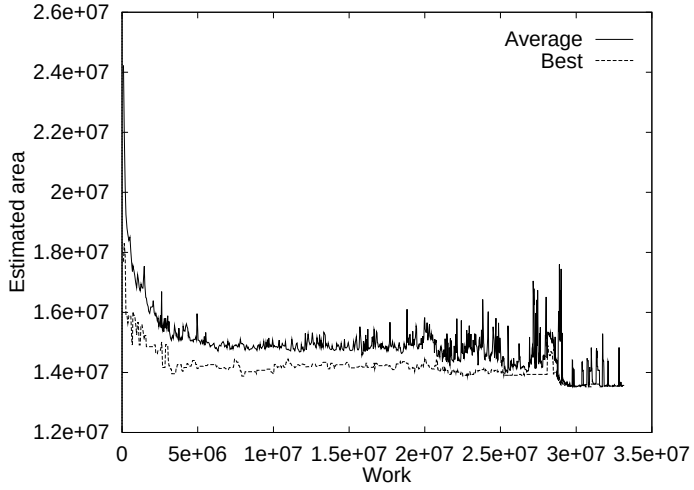

Figure B.13: *Estimated areas of the average and best individuals as functions of computational work.*

tion B.5.2). This quantity is approximately proportional to the actual CPU-time required.

For each generation, the average number of attempted mutations per individual is shown in Fig. B.14. Offspring generated later in the optimization process are subjected to more mutations. The probability of accepting a cost-increasing mutation of an individual is decreased according to the number of mutations performed on it, as described in Section B.4.2. Therefore, the average value of the acceptance probabilities P_s (as defined in Section B.4.2) in the population decreases with time, as illustrated in Fig. B.15. The ratio of attempted mutations, which are accepted and performed, is shown in Fig. B.16 as a function of generation number.

mixed mode. These graphs are extracted from the same sample execution of the algorithm using the parameter values $M_0 = 25$, $S = 200$, $p^0_{mut} = 0.025$, $p_{inv} = 0.05$, $\beta = 0.7$, $\gamma = 1.4$, $R = 80$, $\mathcal{P} = 0.99$, $\alpha = 0.6$ and $\lambda = 1$.

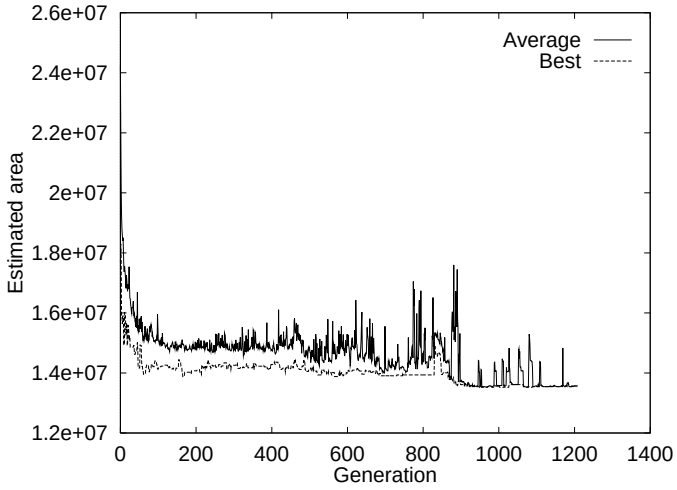

Figure B.11: *Estimated areas of the average and best individuals as functions of generation number.*

Fig. B.11 shows for each generation the average of the estimated area of all individuals and the estimated area of the best individual. Both quantities improves very rapidly within the first 100 generations. From then and until about generation 800, the best individual improves only very slowly. Up to this point, the observed behaviour is typical for a pure GA, in case of which no further improvement should be expected. However, due to the SA component of this algorithm, new significant improvements are obtained from generation 800 to 1,000. The very best individual emerges in generation 1,009 and the process terminates after 1,209 generations.

The population size M decreases as shown in Fig. B.12. From generation 1,170 it equals 1 and the process becomes pure SA. Of course this does not always happen. In many executions, the final population size is greater than 1.

Since the population size as well as the expected number of attempted mutations on each individual per generation varies, the number of generations simulated is not proportional to the actual amount of computations performed. The graphs of Fig. B.13 give the obtained areas as functions of computational work. For practical reasons, work is measured here as the number of channel densities measured during decodings (see Sec-

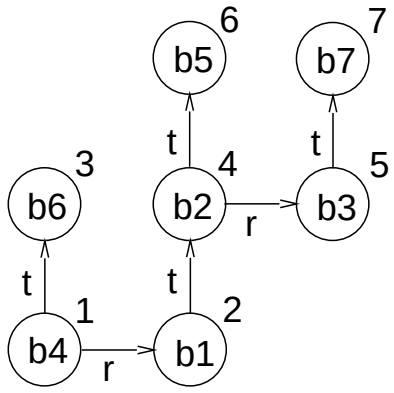

Figure B.10: *Another genotype for the phenotype in Fig. B.7.*

B.6 Experimental Results

In this section experimental results obtained with an implementation of the application of SACA to macro-cell placement is reported. In Section B.6.1, the behaviour of SACA when executed in mixed GA/SA mode is investigated. Section B.6.2 compares the performance of the pure GA mode with the mixed GA/SA mode, and in Section B.6.3, the performance of SACA is compared to the best results obtained by other placement algorithms found in the literature.

Benchmark	Cells	Nets	Terminals
Avqsmall	21854	22124	64
Xerox	10203	20908	10
Hp	11682	33054	45

Table B.1: *Benchmark characteristics.*

The implementation is written in the C programming language and consists of about 14,000 lines of source code. All experiments are performed on a DECstation 5000-240 workstation. Performance is measured using three benchmarks from the 1992 MCNC International Workshop on Placement and Routing. Table B.1 lists the main characteristics of these examples.

B.6.1 Behaviour in Mixed Mode

The most interesting execution mode of SACA is the mixed GA/SA mode, which also causes the most complex behaviour. Figures B.11 through B.16 illustrate the typical optimization process obtained in the

and updated as described in Section B4.2 using the corresponding standard deviations of area and interconnect length $\hat{\sigma}_a$ and $\hat{\sigma}_w$, respectively. The same parameter P_s (and P_s) is used for both temperature computations. The probability $P_{acc}(s, r)$ of accepting the mutation $r = mutate(s)$ is then calculated as

$$P_{acc}(s, r) = \begin{cases} \exp\left(\frac{C_a(r) - C_a(s)}{T_s^a}\right) & \text{if } C_a(r) > C_a(s) \\ \exp\left(\frac{C_w(r) - C_w(s)}{T_s^w}\right) & \text{if } C_a(r) = C_a(s) \wedge C_w(r) > C_w(s) \\ 1 & \text{otherwise} \end{cases}$$

The implementation of the local hill climber, routine $optimize(t)$ of Fig. B5 is very simple. It performs a sequence of mutations, each of which improves the fitness of t. An exhaustive strategy is used so that when $optimize(t)$ has been executed, no single mutation exists, that can improve t further.

B5.6 Inversion Operator

For a given phenotype, several equivalent genotypes usually exist. Since crossover is performed in terms of genotypes, the fitness of produced offspring depends on which of the possible genotypes are used as codings of the given phenotypes. As mentioned in Section B2, the purpose of inversion is to optimize the performance of the crossover operator by rearranging the components within a given genotype.

The inversion operator selects a subtree at random and moves it to another free position in such a way that no constraints are violated and so that the corresponding phenotype is still the same. An example of this is shown in Fig. B10. This genotype tree is generated by moving the subtree rooted at b_2 in the genotype shown in Fig. B7.

2. Alter the set of edges E by exchanging b_i and b_j. The priorities of the cells are exchanged simultaneously so that no pair of cells are prevented a priori from being exchanged due to the constraint that any node always has a higher priority than its predecessor. An example is shown in Fig. B.9.

3. Alter π by exchanging $\pi(b_i)$ and $\pi(b_j)$.

4. Change the transformation of a cell by altering the value of $o(b_i)$.

When performing each of these mutations, a part of the genotype has to be decoded to check if the mutated individual satisfies all constraints. Mutations 1 and 4 require that all cells having priority $\pi(b_i)$ or higher are decoded, while mutations 2 and 3 require decoding from priority $\min(\pi(b_i), \pi(b))$. A mutation is only performed if it does not cause any constraint violations.

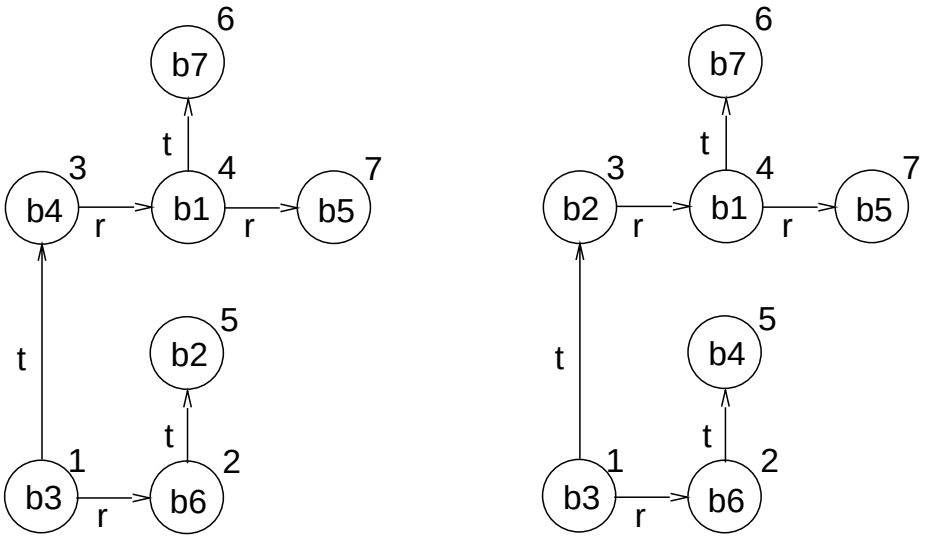

Figure B.9: *A mutation of type 2: Cells b_2 and b_4 are exchanged, while the priorities are still attached to the same positions in the tree.*

Because of the dual optimization criterion described in Section B.5.3, the cost of an individual s cannot be suitably expressed in a single number $C(s)$. Therefore, the acceptance criterion for mutations shown in Fig. B.6 has to be modified slightly for this application. Let $C_a(s)$ and $C_w(s)$ denote the estimated area and total interconnect length of s, respectively. Each individual has two separate temperatures, T_s^a corresponding to area and T_s^w corresponding to interconnect length. These are defined

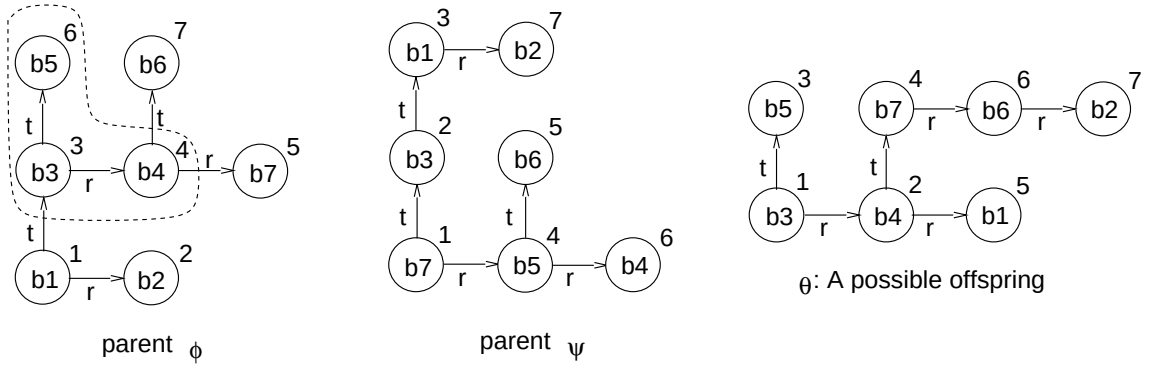

Figure B8: *Combining ϕ and ψ.*

E_θ is constructed as follows. From the cell tree of ϕ, a connected subset $T' = (V', E')$, $V' \subset V$, $E' \subset E_\phi$ is chosen. T' is chosen at random but subject to the constraint that decoding T' in the order defined by π_ϕ, i.e., using $b \in V' \mid \forall b' \in V' \setminus \{b\} : \pi_\phi(b) < \pi_\phi(b')$ as root, causes no constraint violations. The size of V' is determined by a normal distributed stochastic variable having mean $n/2$ and standard deviation 1. In Fig. B8, the chosen T' is indicated by the dashed line. Initially E_θ is defined to be E'. Hence, θ has inherited all cells in V' from ϕ. The remaining cells $V - V'$ are then inherited from ψ by extension of E_θ. The cell tree of ψ is traversed in ascending order according to π_ψ. At any node it is checked if the corresponding cell b belongs to V', that is, whether it has been placed in θ already. If so, the cell is skipped. Otherwise, b is added to the cell tree of θ by extending E_θ. The position at which to add b is randomly chosen among all free and feasible positions. The transformation of any cell is inherited unaltered together with the cell itself. π_θ is uniquely defined so that it corresponds to the order in which the cells were placed when creating E_θ.

B.5.5 Mutation Operator and Hill Climber

The implementation of the operator *mutate* of Fig. B.3 performs four different types of random changes on the given genotype. Let b_i and b_j denote two randomly chosen cells, $i \neq j$. The four types of mutation are:

1. Alter the set of edges E by moving a leaf b_i to another free and randomly chosen position. The type of the edge going into the leaf may be changed as part of the move.

of each node. The transformation of each cell is defined by the function $o : V \rightarrow \{0,1,2,\ldots,7\}$.

When all cells are placed, the decoder computes the rectangle B. This is done by extending the smallest rectangle enclosing all cells, until the routing area estimate is satisfied along all edges of B. At any point in time of the optimization process, each individual satisfies all constraints.

B.5.3 Fitness Measure

The fitness of an individual is relative to the fitness of the rest of the population. Therefore, fitness values are always computed for a population of individuals at a time. Let Φ be the set of all possible individuals for a given instance of the problem having n cells $b_1, \ldots, b_n$, and let $\mathcal{F} : \Phi \rightarrow \Re^+$ denote the fitness measure. Since the objective is to minimize layout area, initially $\mathcal{F}$ is defined as

$$\mathcal{F}(s) = \frac{1}{\mathcal{A}(B_s) - \sum_{i=1}^{n} \mathcal{A}(b_i)}$$

where B_s is the outer rectangle of the individual s and $\mathcal{A}(x)$ is the area of rectangle x. That is, $\mathcal{F}(s)$ is the inverse of the total estimated routing area in s. All individuals having equal area will now have equal fitness. But when fixing the total area of a placement, the probability of a 100% routing completion within the estimated area is likely to increase as the total interconnect length decreases. The minimization of the total interconnect length is therefore introduced as a secondary optimization criterion. All individuals having the same area will have their fitness values adjusted so that fitness increases as the estimated interconnect length decreases. This adjustment assures that area is still the primary optimization criterion, i.e., smaller area will always mean higher fitness. Finally, the adjusted fitness values are scaled linearly as described in [6] in order to control the variance in the population. For a detailed description of the fitness computation the reader is referred to [5].

B.5.4 Crossover Operator

Given two individuals ϕ and ψ, the crossover operator generates a feasible offspring θ. This operation is illustrated in Fig. B.8. Throughout this section, a subscript specifies which individual the marked property is a part of.

The genotype of an individual having n cells $b_1, \ldots, b_n$ is now described. An example genotype with $n = 7$ cells is shown in Fig. B.7 together with the corresponding phenotype. The absolute positions of all cells are represented by a binary tree (V, E), $V = \{b_1, \ldots, b_n\}$, in which the i'th node corresponds to cell i. Two kinds of edges exist: top-edges and right-edges, so that $E = E_t \cup E_r$, $E_t \cap E_r = \emptyset$. All edges are directed and are oriented away from the root of the tree. Each node has at most one outgoing top-edge and at most one outgoing right-edge.

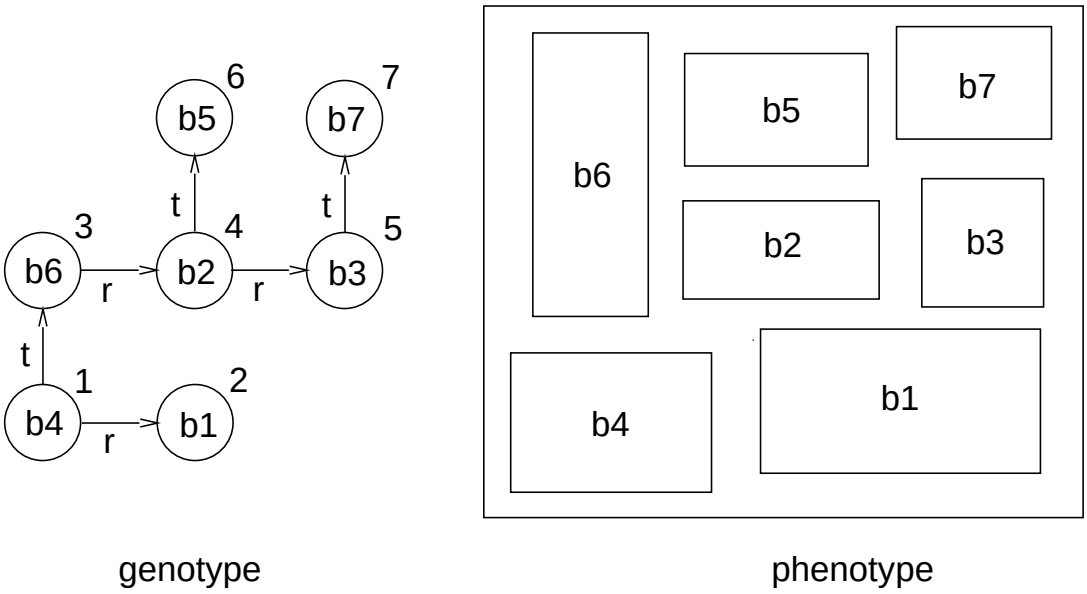

Figure B.7: *An example genotype and the corresponding phenotype.*

Let $e_{ij} \in E$ denote an edge from b_i to b_j, and let (b_i^{xl}, b_i^{yl}) and (b_i^{xu}, b_i^{yu}) denote the coordinates of the lower left and upper right corners of b_i, respectively. $e_{ij} \in E_t$ (E_r) means that cell b_j is placed above (to the right of) b_i in the phenotype. That is,

$$\forall\, e_{ij} \in E : e_{ij} \in E_t \Rightarrow b_j^{yl} \geq b_i^{yu}\ , \quad e_{ij} \in E_r \Rightarrow b_j^{xl} \geq b_i^{xu}\ .$$

The tree is decoded as follows. The cells are placed one at a time in a rectangular area having horizontal length W and infinite vertical length. Each cell is moved as far down and then as far left as possible without violating the routing area estimate, which is computed as each cell is placed. The estimate is based on the computation of channel densities, and is described in detail in [5]. The cells are placed in ascending order according to their *priorities*, which are defined by the one-to-one mapping $\pi : V \rightarrow \{1, \ldots, n\}$. Any node has higher priority than its predecessor in the tree. In Fig. B.7, the priorities are indicated at the top right hand side

B.5.1 Problem Definition

The macro-cell placement problem can be defined as follows: Given

- A set of rectangular $cells$, each with a number of $terminals$ at fixed positions along the edges of the cell.
- A netlist specifying the interconnections of all terminals.
- An approximate horizontal length W of the chip under construction.

Compute

- The absolute position of each cell.
- The orientation and possible reflection(s) of each cell.
- A rectangle B defining the shape of the chip.

The objective is to minimize the area of B subject to the following constraints:

- No pair of cells overlap each other.
- The rectangle B encloses all cells and has approximate horizontal length W.
- The area within B, which is not occupied by cells, is sufficiently large to contain all routing needed to implement the required interconnections.

To meet the last constraint, the necessary routing area is estimated during the placement. The estimate is based on the assumptions that two metal layers are used for routing, the area occupied by cells and the area used for routing are disjoint, and all nets are treated as signal nets.

B.5.2 Genetic Encoding

The genetic encoding of a macro-cell placement is based on a generalization of the two-dimensional bin packing problem. The standard bin packing algorithm places the blocks in the bin one at a time at the downmost and then at the leftmost position. For a given instance of the placement problem, let a *BL-placement* (bottom-left) denote a solution, in which no cell can be moved further down or to the left without causing a violation of a routing area estimate. The solution space considered by the algorithm is restricted to the set of all possible BL-placements.

```
r = mutate(s);
if T_s = ⊥ do:
    P_s = P;
    T_s = σ̂ / ln(P_s);
    c_s = 0;
end;
with prob min(exp((C(s) - C(r)) / T_s), 1.0) do:
    s = r;
    c_s = c_s + 1;
    if c_s = λ do:
        P_s = α P_s;
        T_s = σ̂ / ln(P_s);
        c_s = 0;
    end;
end;
```

Figure B.6: *Structure of the routine SAmutate(s).*

B.4.3 GA and SA as Special Cases

SACA reduces to a pure GA when $M_0 > 1$, $R = S$, $\alpha = 1.0$, and $\mathcal{P}$ is close to 1.0. Pure SA is obtained whenever $M_0 = 1$. In this case, the reproduction step in general is equivalent to a mutation which is accepted if and only if it improves cost. Standard crossover operators as found in [6, 7] have the property that crossover(x, x) always yields the offspring x, in which case the reproduction step becomes equivalent to the empty statement [1].

B.5 Application to Macro-Cell Placement

The specific genetic encoding and corresponding operators developed for the macro-cell placement problem is briefly described in this section. For a detailed description, the reader is referred to [5].

[1] Alternatively, the generation of Π_n can be conditioned by $M > 1$, as can the invocation of the inversion operator, if desired.

The population size M after c_R reductions is

$$M = \max(\text{round}(\beta^{c_R} M_0),\ 1.0)$$

where M_0 is the initial population size, $0 \leq \beta \leq 1$ is a real valued parameter, and round(x) performs rounding to the nearest integer value of x. Ultimately we may have $M = 1$, corresponding to a pure SA process. When M is decreased, the M fittest individuals are kept, while the rest are discarded. Furthermore, the mutation rate p_{mut} is increased so that after c_R increases, it is given by

$$p_{mut} = \min(\gamma^{c_R} p^0_{mut},\ 1.0)$$

where p^0_{mut} is the initial mutation rate, and $\gamma \geq 1$ is a real valued parameter. Finally, notice that mutations are now performed by the routine *SAmutate*, which will be discussed in the following section.

B.4.2 SA-Controlled Mutations

Mutation of individual s is performed as illustrated in Fig. B.6. Routine *mutate* of Fig. B.3 is used to generate a random change of s. If this is the first mutation of s, its variables P_s, T_s and c_s controlling its cooling schedule are defined. Then the mutation is performed with a temperature-dependent probability, as in SA. It may be noted how this scheme resembles the SA outline of Fig. B.4. However, the temperature reduction is now computed in a slightly different way which will be explained below.

The absolute values of a suitable temperature schedule are problem-dependent. To circumvent this problem, we define a schedule for reducing the probability of accepting a cost-increasing mutation. The temperature decrease is then computed so that the specified probability is obtained. More specifically, let P_s be the probability of accepting a mutation on s, which increases the cost of s by σ, the standard deviation of the cost of all solutions in the search space. From an initial value $\mathcal{P}$, $0 < \mathcal{P} < 1$, P_s is then reduced by a factor α, $0 < \alpha \leq 1$, whenever quasi-equilibrium has been obtained. For a given value of P_s, the corresponding temperature T_s is computed as

$$T_s = \frac{-\hat{\sigma}}{\ln(P_s)}$$

where $\hat{\sigma}$ is an estimate of σ computed during generation of the initial population.

```
generate(Π_c);
∀ s ∈ Π_c: T_s = ⊥;
evaluate(Π_c);
q = best(Π_c);
c_R = 0;
repeat until stopCriterion():
    if no improvement for R generations do:
        c_R = c_R + 1;
        M = max(round(β^{c_R} M), 1);
        Π_c = reduce(Π_c, M);
        p_mut = min(γ p_mut, 1.0);
    end;
    Π_n = ∅;
    repeat M times:
        select s ∈ Π_c, t ∈ Π_c;
        v = crossover(s, t);
        T_v = ⊥;
        Π_n = Π_n ∪ {v};
    end;
    evaluate(Π_c ∪ Π_n);
    Π_c = reduce(Π_c ∪ Π_n, M);
    ∀ s ∈ Π_c: s = mutate(s);
    ∀ s ∈ Π_c: with prob. p_inv do:
        s = invert(s);
    evaluate(Π_c);
    q = best(Π_c ∪ {q})
end;
∀ t ∈ Π_c ∪ {q}: t = optimize(t);
r = best(Π_c ∪ {q});
```

Figure B.5: *Outline of SAGA.*

B.4 The Unified Algorithm

The unified algorithm SACA can now be presented. It can be viewed as a GA which has been modified in two major ways, each of which will be discussed in detail in the following sections:

1. The mutations performed on an individual are accepted with a certain probability as in SA. Each individual has its own temperature, and during its lifetime, its temperature is decreased according to its own cooling schedule.

2. Initially, SACA executes as a pure GA. But as the GA stagnates, as illustrated in Fig. B.1, SACA gradually switches over to SA. The speed of this switch is adaptive, since it is determined by the progress of the optimization itself.

SACA has two important properties:

- It is application-independent, in the sense that it can potentially be applied to any optimization problem for which GA and SA are well-suited.

- It unifies the GA and SA in such a way that it can be executed exclusively in GA or SA mode by selecting appropriate values of its control parameters.

B.4.1 The Switch Towards SA

Fig. B.5 gives an overview of SACA. By comparing it to Fig. B.2, it can be seen that only few things have changed. The temperature of individual s is denoted T_s, and $\perp$ denotes the undefined value. Thus, every new individual, generated in the initial population or as a result of crossover, has an undefined temperature.

The switch towards SA is handled by the *if*-statement which initiates each generation. A step towards SA is taken whenever no improvement has been observed for R generations, $0 \leq R \leq S$. A step towards SA consists of reducing the population size M, and increasing the mutation rate p_{mut}. In other words, more SA-controlled mutations will be performed on a smaller number of individuals.

B.3 The Simulated Annealing Algorithm

The idea of SA is to perform optimization by simulating the thermal process of cooling down a solid in such a way that it obtains a state of minimal energy. A good presentation of SA is given in [1].

```
generate(s);
T = T;
repeat until stopCriterion():
    c_s = 0;
    while c_s < λ do:
        q = mutate(s);
        with prob. min(exp(-(C(s)-C(q))/T), 1.0) do:
            s = q;
            c_s = c_s + 1;
        end;
    end;
    T = αT;
end;
```

Figure B.4: *Outline of the SA.*

Fig. B.4 outlines a simple SA implementation. It starts with a randomly generated solution s. As the algorithm progresses, a sequence of random changes are performed on s. Routine *mutate* of Fig. B.3 is used for this, assuring that p_{mut} is sufficiently high. Each change is accepted and carried out with a probability which depends on the temperature T. The temperature is regularly decreased according to the parameter $0 < \alpha \leq 1$, and it starts from an initial temperature T. At each fixed temperature, a sequence of changes are performed on s until a quasi equilibrium state is obtained. In Fig. B.4, the temperature is reduced each time λ changes on s have been accepted. $C(x)$ denotes the cost of the solution x. Note that if a random change decreases the cost of s, it is always accepted. If the cost is increased, it is accepted with probability $\exp(-\frac{C(s)-C(q)}{T})$, which decreases with T.

in terms of genotypes, while fitness is defined in terms of phenotypes. For a given genotype, the corresponding phenotype is computed by a *decoder*. A good introduction to genetic algorithms is given in [6].

Fig. B.2 shows a template for the GA considered here. Initially, the current population Π_c is constructed from randomly generated individuals. Routine *evaluate* computes the fitness of each of the given individuals, while *bestOf* finds the individual with the highest fitness. One execution of the outer "repeat" loop corresponds to the simulation of one generation. Throughout the simulation, $M = |\Pi_c|$ is kept constant. We keep track of the best individual q ever seen. Routine *stopCriterion* terminates the simulation when no improvement has been observed for S generations. Each generation is initiated by the formation of a set of offspring Π_n of size M. The two mates s and t are selected independently of each other, and each mate is selected with a probability proportional to its fitness. Routine *reduce*(Π, k) returns the k fittest individuals from Π, thereby keeping the population size constant.

```
∀ components g_1, g_2, ..., g_k of tg:
    with prob p_mut do:
        alter g_k randomly
```

Figure B.3: *Structure of the routine mutate(t).*

As illustrated in Fig. B.3, the mutation operator performs pointwise mutation with a given probability on each of the components, or *genes*, of its argument. The genetic operator *invert*(t) alters the genotype of t without altering the corresponding phenotype. As described in [6], the purpose of this operator is to optimize the relative positions of the genes of t with respect to the crossover operator. Finally, local hill climbing is performed on all existing individuals by routine *optimize*(t). It is common practice to apply a hill climber in a GA in an attempt to slightly improve the final solution [6]. The solution r is the output of the algorithm.

less fit individuals tend to die out. This *survival-of-the-fittest* Darwinian principle is the basic idea behind the GA.

```
generate(Π_c);
evaluate(Π_c);
q = best(Π_c);
repeat until stopCriterion():
    Π_n = ∅;
    repeat M times:
        select s ∈ Π_c, t ∈ Π_c;
        v = crossover(s, t);
        Π_n = Π_n ∪ {v};
    end;
    evaluate(Π_c ∪ Π_n);
    Π_c = reduce(Π_c ∪ Π_n, M);
    ∀ t ∈ Π_c: t = mutate(t);
    ∀ t ∈ Π_c: with prob p_inv do:
        t = invert(t);
    evaluate(Π_c);
    q = best(Π_c ∪ {q})
end;
∀ t ∈ Π_c ∪ {q}: t = optimize(t);
r = best(Π_c ∪ {q});
```

Figure B.2 *Outline of the GA.*

The algorithm maintains a *population* of *individuals*, each of which corresponds to a specific solution. A measure of *fitness* defines the quality of an individual. Starting with a set of random individuals, a process of evolution is simulated. The main components of this process are *crossover*, which mimics propagation, and *mutation*, which mimics the random changes occurring in nature. After a number of *generations*, highly fit individuals will emerge corresponding to good solutions to the given optimization problem. A *phenotype* is the physical appearance of an individual, while a *genotype* is the corresponding genetic encoding or representation of the individual. Crossover and mutation are performed

have developed and compared various mixed strategies for the TSP. One of the strategies is called life-cycle. A population of individuals coexist. Mutations are accepted with a certain probability as in SA. Each individual goes through a life-cycle. As it gets older, its probability of being mutated decreases while the probability of mating increases. Boseniuk and Ebeling report that the life-cycle strategy is superior to pure SA on the TSP.

The approach presented here is inspired by Boseniuk and Ebeling's ideas, although many significant refinements have been made to improve the performance of the algorithm. In our approach, the way GA and SA are mixed is dynamically changed during the optimization process, while it is static in [3]. The contributions of this paper are:

- To provide a new algorithmic model which unifies the GA and the SA into one algorithm. The resulting algorithm is application independent and highly adaptive.
- To demonstrate the performance of the approach on the macro-cell placement problem. It is experimentally shown that a mixed strategy performs better than a pure GA. Furthermore, using the mixed strategy on MCNC macro-cell placement benchmarks, we obtain results comparable to, or better than previously published results.

The rest of this paper is organized as follows. In Sections B.2 and B.3 the concepts of GA and SA are briefly introduced. The unified algorithm is then discussed in detail in Section B.4. The discussion in Sections B.2, B.3 and B.4 is application independent. Section B.5 describes the application of the unified algorithm to the macro-cell placement problem. This includes a brief description of the application-specific genetic encoding and corresponding operators. Finally, experimental results are described in Section B.6, and a conclusion is given in Section B.7.

B.2 The Genetic Algorithm

The concept of genetic algorithms, introduced by John Holland [7] of the University of Michigan, utilizes the notion of the natural evolution process. In nature, the individuals constituting a population adapt to the environment in which they live. The fittest individuals have the highest probability of survival and tend to increase in numbers, while the

B.1 Introduction

The genetic algorithm (GA) is a general-purpose stochastic optimization technique, frequently used to solve NP-hard optimization problems. It has been successfully applied to a wide variety of problems in various fields, including VLSI layout generation [9].

The typical GA convergence curve is illustrated in Fig. B.1. Initially, the cost of the solutions improves very rapidly. But then it becomes very difficult to obtain further improvement. The majority of the runtime is spent in the later phase of the process in which small improvements are obtained very slowly. The work presented here is motivated by the need to overcome this shortcoming of the GA. Our approach is to unify the GA with the simulated annealing algorithm (SA), another well-known, high-performance optimization technique. While SA in general is able to obtain improvements also into the late phase of the process, it does not converge as fast as the GA in the initial phase. The unified algorithm, called SACA, is designed in such a way that the advantages of the GA as well as the SA are utilized.

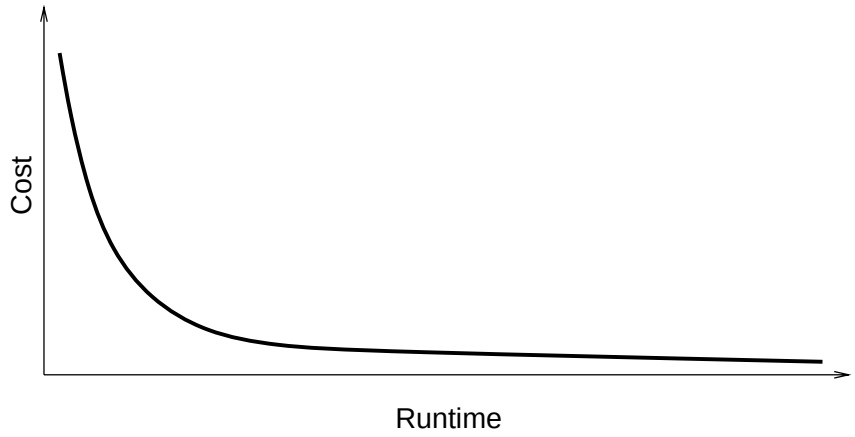

Figure B.1: *The typical convergence of a GA.*

Earlier attempts to combine GA and SA have been made. With the GA as the starting point, Strag and Wasser have incorporated elements from SA with the objective to improve control of population variance [10]. This is obtained by a so-called thermodynamic operator, in which the number of destroyed/preserved schemata are controlled by a temperature-dependent stochastic variable. The degree of schema disruption decreases as a global temperature is decreased. However, this approach is limited to ordering problems like the Traveling Salesman Problem (TSP). A much more general approach is presented in [3]. Here Boseniuk and Ebeling

Appendix B

SAGA: Macro-Cell Placement by a Unification of the Genetic Algorithm with Simulated Annealing

This paper is coauthored by P. Mazumder and is an extended version of H. Esbensen, P. Mazumder, "SAGA: A Unification of the Genetic Algorithm with Simulated Annealing and its Application to Macro-Cell Placement," *Proc. of The 7th International Conference on VLSI Design*, pp. 211-214, 1994.

Abstract

In this paper a stochastic optimization algorithm called SAGA is presented, which is a generalization of the genetic algorithm and the simulated annealing algorithm. Depending on the settings of its control parameters, SAGA executes as a genetic algorithm, a simulated annealing algorithm, or a controllable mixture of these. SAGA represents an application-independent approach to optimization, and the resulting search process is highly adaptive. The performance of the approach on the macro-cell placement problem is examined. It is experimentally shown that a mixture of the genetic algorithm with simulated annealing yields higher layout quality than a pure genetic algorithm. Furthermore, layout qualities obtained by SAGA on MCNC benchmarks have been observed to be comparable to or better than previously published results.

[10] Donald McFarlane, Ian East, "An Investigation of Several Parallel Genetic Algorithms", *Tools and Techniques for Transputer Applications*, pp. 60-67, IOS Press, 1990.

[11] S. Mohan, P. Mazumder, "VLSI Wires: Standard Cell Placement on a Network of Workstations", *IEEE Transactions on Computer-Aided Design*, Vol. 12, No. 9, pp. 1312-1326, Sept. 1993.

[12] H. Mühlenbein, M. Georges-Schleuter, O. Krämer, "Evolution algorithms in combinatorial optimization," *Parallel Computing*, Vol. 7, pp. 65-85, 1988.

[13] Hidetoshi Onodera, Yo Taniguchi, Keikichi Tamaru, "Branch-and-Bound Placement for Building Block Layout", *Proceedings of The 28th Design Automation Conference*, pp. 433-439, 1991.

[14] S. Sahni, A. Bhatt, "The Complexity of Design Automation Problems", *Proceedings of The 17th Design Automation Conference*, pp. 402-411, 1980.

[15] K. Shahookar, P. Mazumder, "A Genetic Approach to Standard Cell Placement using Meta-Genetic Parameter Optimization," *IEEE Transactions on Computer-Aided Design*, Vol. 9, No. 5, pp. 500-511, May 1990.

[16] K. Shahookar, P. Mazumder, "VLSI Cell Placement Techniques", *ACM Computing Surveys*, Vol. 23, No. 2, 1991.

[17] K. Shahookar, P. Mazumder, "GASP - A Genetic Algorithm for Standard Cell Placement", *Proceedings of The European Design Automation Conference*, pp. 660-664, March 1990.

[18] M. Upton, K. Sakii, S. Sugiyama, "Simulated Annealing Placement for Mixed Macro Cell and Standard Cell Layouts", *Proceedings of The International Workshop on Layout Synthesis*, Vol. 1, May 1990.

Bibliography

[1] M. Beardslee, J. Burns, A. Casotto, M. Iqusa, F. Romeo, A. Sangiovanni-Vincentelli, "MISMAC: User's manual", Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, March 1990.

[2] H. Chan, P. Mazumder, K. Shahookar, "Macro-cell and module placement by genetic adaptive search with bit-map-represented chromosome", *Integration, the VLSI Journal*, Vol. 12, No. 1, pp. 49-77, Nov. 1991.

[3] J. P. Cohoon, S. U. Hedge, W. N. Martin, D. Richards, "Distributed Genetic Algorithms for the Floorplan Design Problem," *IEEE Transactions on Computer-Aided Design*, Vol. 10, pp. 483-492, April 1991.

[4] J. P. Cohoon, W. D. Paris, "Genetic Placement", *Proceedings of the IEEE International Conference on Computer-Aided Design*, pp. 422-425, 1986.

[5] W. M. Dai, B. Eschermann, E. S. Kuh, M. Pedram, "Hierarchical Placement and Floorplanning in BEAR", *IEEE Transactions on Computer-Aided Design*, pp. 1335-1349, Vol. 8, No. 12, 1989.

[6] D. E. Goldberg, "Genetic Algorithms in Search, Optimization, and Machine Learning", *Addison-Wesley*, 1989.

[7] A. Herrigel, W. Fichtner, "An Analytic Optimization Technique for Placement of Macro-Cells", *Proceedings of The 26th Design Automation Conference*, pp. 376-381, 1989.

[8] J. H. Holland, "Adaption in Natural and Artificial Systems", *University of Michigan Press*, Ann Arbor, MI, 1975.

[9] B. Krüger, P. Schwerdtfeger, O. Vornberger, *Genetic Packing of Rectangles on Transputers*, Transputing '91, Vol. 2, IOS Press, 1991.

MIMD architecture, due to the inherent parallelism in this kind of algorithm. This has been demonstrated numerous times in the literature, cf. e.g. [9, 10, 11, 12].

A.5 Conclusion

In this paper a genetic algorithm for the macro-cell placement problem has been presented. The algorithm is based on a generalization of the two-dimensional bin-packing problem. By using the notion of bin-packing, a genetic encoding has been developed in which most constraints of the problem are implicitly represented. As a consequence, each individual always satisfies every constraint. This design decision is in direct contrast with the more frequent approach of allowing constraint violations throughout the optimization process and controlling the degree of violation by introducing penalty terms in the quality measure. The advantage of the proposed strategy is that it allows a more accurate estimate of the layout quality, since the use of penalty terms have been avoided.

The layout quality obtained by the algorithm is comparable to the best published results. Since this work is a novel approach to macro-cell placement, further improvements are likely. The current runtime is not competitive, but can be improved significantly. Therefore it is concluded that the genetic algorithm is a promising approach to the macro-cell placement problem.

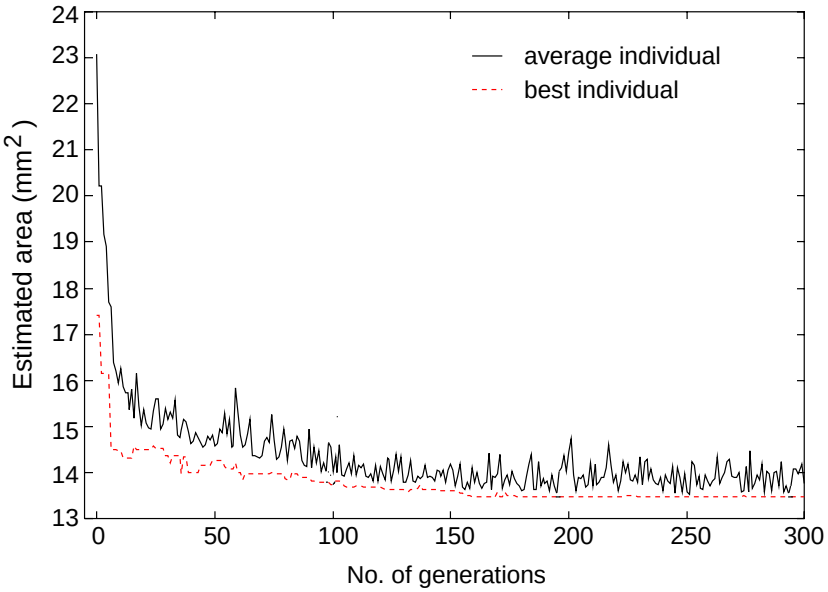

Figure A8: *The typical evolution of the solutions during time. For each generation the estimated area of the best and average individual is shown.*

A4.5 Computation Time

The main factor setting the limits of the applicability of the algorithm is the time consumption. On average, the current implementation requires 52 CPU-minutes for the apte benchmark, 55 CPU-minutes for hp and 156 CPU-minutes for xerox. The computation time requirements currently prevents experiments with larger benchmarks like ami33 and ami49. Furthermore, in order to make the runtime competitive to that of other systems, it needs to be reduced by a factor of 3 to 10. However, there are two main reasons why it is believed that such a significant reduction is indeed obtainable.

Firstly, in the current experimental implementation of the algorithm, the vast majority of the total runtime is spent measuring channel densities during decoding. For an example such as the hp benchmark, the algorithm computes between 10 and 30 million channel densities. Whenever a new density is needed, it is computed without reusing any information, no matter how similar the new positioning of the involved cells are to the previous situation. Therefore, a data structure which allows a channel density to be dynamically updated as a cell is being moved slightly, should be developed.

Secondly, the runtime can also be significantly improved by implementing a parallel version of the algorithm. One of the characteristics of GAs in general is that a high speedup can be expected on any

Benchmark	System	Area	Wirelength	Vias
apte	This work	53.99	563	720
	BBL [13]	54.05	460	-
	Seattle Silicon [18]	54.77	350	-
hp	Seattle Silicon [18]	11.85	200	-
	This work	11.95	262	697
	BBL [13]	12.15	278	-
xerox	Seattle Silicon [18]	25.79	601	1104
	BBL [13]	26.17	628	-
	This work	26.58	556	1377
	BEAR [5]	28.47	633	897
	MOSAICO [1]	29.01	650	1173
	VITAL [1]	31.17	866	1029

Table A.4: *Comparison of quality with other systems. A hyphen indicates that the value is not available.*

A.4.4 Convergence Rate

Fig. A.8 shows the estimated areas of the best and average individuals as a function of time for the typical optimization process. During the first few generations the best as well as the average individual improves drastically and very fast from the initial random solutions. Then, when a large number of individuals in the population presumably are relatively close to the optimum solution, further progress becomes very slow. This is the typical behaviour of any GA. Here it has the advantage that if the designer is willing to settle for a solution which is relatively far, say 10 % from the best obtainable, then that solution can be produced much faster.

[1] References of these tools can be found in [5, 18].

A4.3 Layout Quality

Since the algorithm is stochastic the layouts generated by consecutive program executions will not be exactly identical. For each of the three benchmarks the results of executing the algorithm ten times using a random initialization of the random number generator, are shown in Table A3.

Benchmark	A_{best}	A_{avg}	A_{σ}
apte	53.99	55.91	1.20
xerox	26.58	29.11	1.51
hp	11.95	12.81	0.49

Table A3: *Variation in result quality*

A_{best} and A_{avg} are the best and average areas, respectively, of the completed layouts, while A_{σ} is the standard deviation. All values are core areas in mm^2. Fig. A7 shows the best placement obtained for the hp benchmark. As can be seen, the cells are moved only slightly during routing, reflecting a quite accurate routing area estimation.

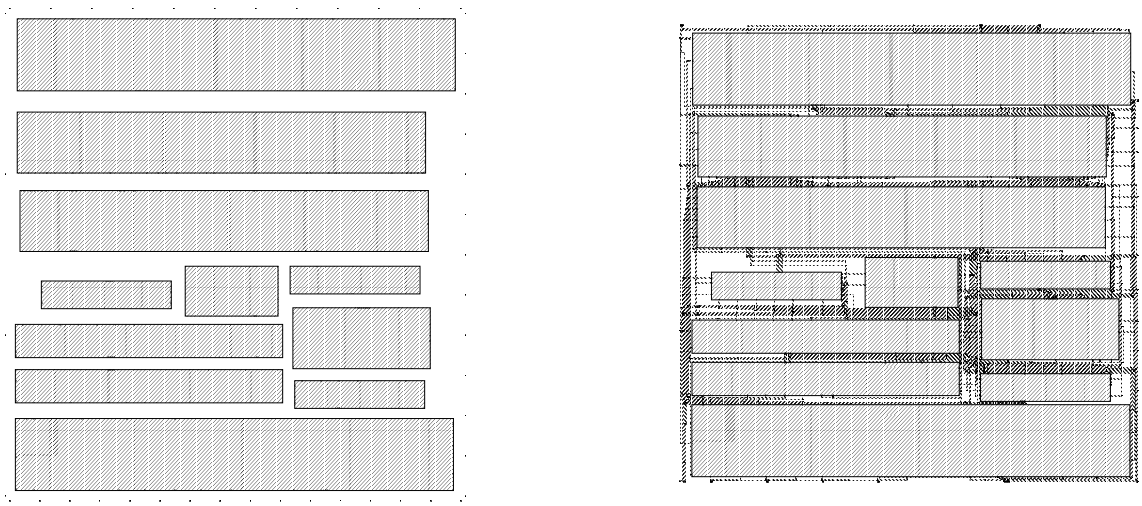

Figure A7: *The hp benchmark before and after routing.*

In Table A4 the layout quality obtained is compared to the best published results. Again, the absolute area is core area of the completed layout in mm^2. The total wirelength in mm and the total number of vias in each layout is also listed. The results referenced should be compared with some caution due to minor variations in the problem definitions used.

At first sight this may seem surprising, since this particular combination represents the highest degree of randomization among all strategies tested. However, as described in Section A.3, the selection strategy used for survival into the next generation is purely deterministic and speeds up the convergence of the algorithm. The highly randomized crossover operator is the variant that counteracts this potentially dangerous effect the best. Another possible reason for the observed results is that the complex structure of the search space prevents really good solutions from being generated using greedy strategies like β 2a and β 2b. Consequently, instead of improving the efficiency of the search, the greedy strategies actually prevents really good solutions from being found.

A.4.2 Parameter Settings

For all examples considered, the same set of parameter values have been used to control the GA, i.e., no problem-specific tuning has been performed towards each benchmark. The values used are $M = |P_C| = 25$, a mutation probability of 0.025 for each of the four types of mutation, and an inversion probability of 0.05. The algorithm was terminated when no improvement had been observed for $s = 200$ consecutive generations.

The inversion probability is the probability that a given individual $p \in P_C$ is subject to inversion in a given generation. In contrast to this definition, the mutation probability for a given type of mutation is defined relative to the total number of possible mutations of that type on the individual. This ensures problem-independent mutation rates.

Suitable values for the parameters a and b used in the routing area estimate as described in Section A.3.1.2 depends on the characteristics of the interconnections to be made, i.e., these parameters are problem dependent. Table A.2 shows the values used.

Parameter	apte	xerox	hp
a	0.0	0.2	0.0
b	12.0	10.0	7.0

Table A.2 *Values of parameters for routing area estimate*

$\alpha 1$: $|V_s|$ is chosen uniformly and at random so that $V_{min} \leq |V_s| \leq V_{max}$, where V_{min} and V_{max} are user-defined limits. These limits are chosen so that the interval $[V_{min}, V_{max}]$ is approximately symmetric around $n/2$ and so that the length of the interval is approximately $n/3$, where n is the number of cells.

$\alpha 2$: $|V_s|$ is determined as the rounded value of a normal distributed stochastic variable having mean $n/2$ and standard deviation 1. Therefore, in the large majority of all cases, $|V_s|$ will be approximately $n/2$, but occasionally it may be close to or even equal to 1 or n.

A cell c inherited from the second parent β must be added to the tree of γ by extending E_γ. Three different strategies have been tried for determining the position at which to add the node:

$\beta 1$: Choose a free position at random among all free locations.

$\beta 2$: Add c at a position which will probably lead to a high fitness of γ. Determine this position by evaluating all free positions according to some quality measure, and add c at the most promising position by employing exhaustive search. Two different quality measures have been tried:

- $\beta 2a$: Place c at the position corresponding to the lowest possible position at phenotype level.
- $\beta 2b$: Place c at a position which at phenotype level gives the best packing density of all cells placed so far. To compute the packing density of a given position, we form a rectilinear polygon enclosing all cells placed so far including the cell c at its trial position. The packing density is then defined as the ratio of the sum of the areas of all placed cells including c, divided by the area of the enclosing polygon. The closer this quantity is to 1, the better is the packing density.

All six combinations of one of $\alpha 1$ or $\alpha 2$ with one of $\beta 1$, $\beta 2a$ or $\beta 2b$ have been tried. Whether $\beta 2a$ or $\beta 2b$ is used, no noticeable change in performance has been observed. But $\beta 1$ consistently improves layout quality compared to either of $\beta 2a$ and $\beta 2b$. Furthermore, regardless of the choice of $\beta 1$, $\beta 2a$ or $\beta 2b$, layout quality is always unchanged or improved when using $\alpha 2$ instead of $\alpha 1$. In conclusion, the best results are consistently obtained by using the combination of $\alpha 2$ with $\beta 1$.

A.4 Experimental Results

An experimental version of the algorithm has been implemented in the C programming language, and runs on a DECstation 5000-240 workstation. Approximate size of the source code is 14,000 lines. The performance has been tested on three benchmarks from the 1992 MCNC International Workshop on Placement and Routing. Table A.1 lists the main characteristics of these examples. Routing and compaction of the layouts have been performed by using the Mosaico toolset [1] which is part of the Octtools CAD framework.

Benchmark	Cells	Nets	Terminals	I/O-terms
apte	9	97	287	73
xerox	10	203	698	2
hp	11	83	309	45

Table A.1: *Benchmark characteristics. The number of terminals includes the io-terminals.*

A.4.1 Experiments with the Crossover Operator

In most GA implementations crossover is a random operation in the sense that the parent from which a given feature is inherited is always determined randomly. However, in some GA implementations, a kind of local optimization is performed as an integrated part of the crossover operation. Instead of just combining the features of two given individuals in a completely randomized fashion, an attempt is made to assimilate the features in such a way that a highly fit offspring emerges. This kind of attempt, which is based on application specific knowledge, seems to be a natural way of improving the search process. A general discussion of knowledge-augmented operators can be found in Chapter 5 of Goldberg's book [6].

Experiments with six different variants of the crossover operator described in Section A.3.3 have been performed. These represent various degrees of local optimization. To determine the size of the connected subset V_s inherited from the first parent α, two strategies have been tried

is not one-to-one. This can usually be obtained easily [6], and here the decoder already has this property, as mentioned in section A.3.1.1.

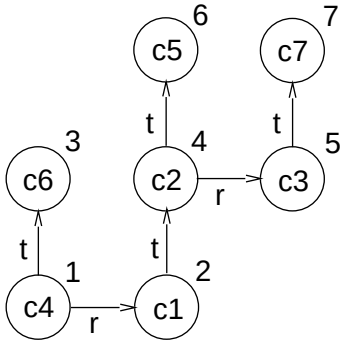

Figure A.6: *Another genotype for the phenotype in Fig. A.2.*

The inversion operator selects a subtree at random and moves it to another free position in such a way that no constraints are violated and so that the corresponding phenotype is still the same. An example of this is shown in Fig. A.6. This genotype tree is generated by moving the subtree rooted at c_2 in the genotype shown in Fig. A.2.

A.3.6 Stop Criterion

The evolution process is terminated when no improvement has been observed for a user-defined number of consecutive generations, denoted here by s. When determining if improvement has occurred or not, we consider the best existing individual as well as the average individual, and we compare individuals in accordance with our dual optimization criterion of minimized area and secondarily, minimized total interconnect length. To be specific, denote by $A_{avg}(t)$ and $A_{best}(t)$ the average and best area of the individuals present in generation t, respectively. Similarly, let $L_{avg}(t)$ and $L_{best}(t)$ denote the average and best total estimated interconnect length present in generation t. Then by definition, an improvement has occurred in generation k, if and only if for some $x \in \{avg, best\}$,

$$A_x(k) < \min\{ A_x(t) \mid k - \delta \leq t < k \}$$

or

$$A_x(k) = \min\{ A_x(t) \mid k - \delta \leq t < k \} \quad \text{and} \quad L_x(k) < \min\{ L_x(t) \mid k - \delta \leq t < k \}$$

where $\delta = \min(k, s)$.

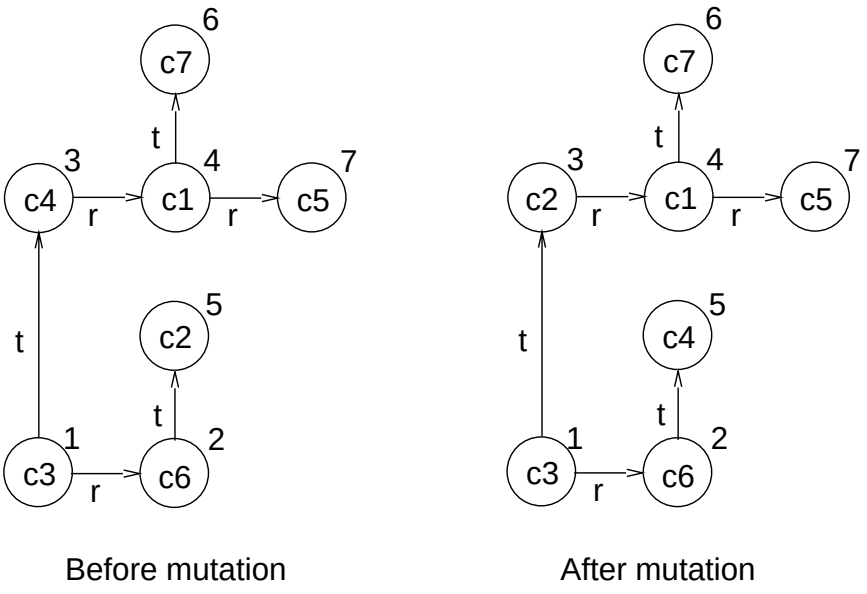

Figure A.5: *A mutation of type 2: Cells c_2 and c_4 are exchanged, while the priorities are still attached to the same positions in the tree.*

A.3.5 Inversion Operator

In a typical GA as presented in [6], the crossover operator works in such a way that the closer two components of the genotype are, the more likely it is that the offspring will inherit both components from the same parent. This is also the case in this algorithm. Let the distance between two nodes in the genotype tree be the number of edges on the unique path connecting the nodes, and let us consider the genotype tree of the first parent α (see Fig. A.4). The smaller the distance between two given nodes is, the more likely it is that both nodes will belong to the connected subset $T_{s} = (V_{s}, E_{s})$, and hence that both nodes will be inherited by the offspring. Since T_{s} is not altered during crossover, the two nodes will continue to be close in the offspring.

If the close nodes represent a good sub-placement and contribute significantly to a high fitness of the individual, then the above property of the crossover operator is beneficial. But otherwise this property may degrade the performance of the algorithm. The purpose of the inversion operator is to eliminate this problem. Given a genotype, the inversion operator computes a new genotype by rearranging the components in such a way that their mutual distances changes, while at the same time assuring that the corresponding phenotype is still the same. This means that in order to apply inversion it is required that the decoder mapping

p^γ should correspond to the order in which the cells were placed when creating E^γ. Since p is a bijection, the following constraints uniquely determines p^γ:

$$\forall\, c_i \in V_s,\quad \forall\, c_j \in V - V_s : p^\gamma(c_i) < p^\gamma(c_j)$$

$$\forall\, c_i,\ c_j \in V_s : p^\alpha(c_i) < p^\alpha(c_j) \Rightarrow p^\gamma(c_i) < p^\gamma(c_j)$$
$$\forall\, c_i,\ c_j \in V - V_s : p^\beta(c_i) < p^\beta(c_j) \Rightarrow p^\gamma(c_i) < p^\gamma(c_j)$$

A.3.4 Mutation Operators

Four different mutation operators exist. Each of these performs some random change in the given genotype. Let c_i and c_j denote two randomly chosen cells, $i \neq j$. The four operators are:

1. Alter the set of edges E by moving a leaf c_i to another free and randomly chosen position. The type of the edge going into the leaf may be changed as part of the move.
2. Alter the set of edges E by exchanging c_i and c_j. The priorities of the cells are exchanged simultaneously, so that no pair of cells are prevented a priori from being exchanged due to the constraint that any node always has a higher priority than its predecessor. An example is shown in Fig. A.5.
3. Alter p by exchanging $p(c_i)$ and $p(c_j)$.
4. Change the transformation of a cell by altering the value of $t(c_i)$.

When performing each of these mutations, a part of the genotype has to be decoded to check if the mutated individual satisfies all constraints. Mutations 1 and 4 require that all cells having priority $p(c_i)$ or higher are decoded, while mutations 2 and 3 require decoding from priority $\min(p(c_i), p(c_j))$. A mutation is only performed if it does not cause any constraint violations.

A.3.3 Crossover Operator

In this section the general functionality of the crossover operator is described. Since this operator is of paramount importance to the overall performance of the algorithm, several experiments have been carried out with respect to its detailed operation. These experiments are described in section A.4.1. Given two individuals α and β, the crossover operator generates a new feasible individual γ, the descendant of α and β. This operation is illustrated in Fig. A.4. Throughout this section, a superscript specifies which individual the marked property is a part of.

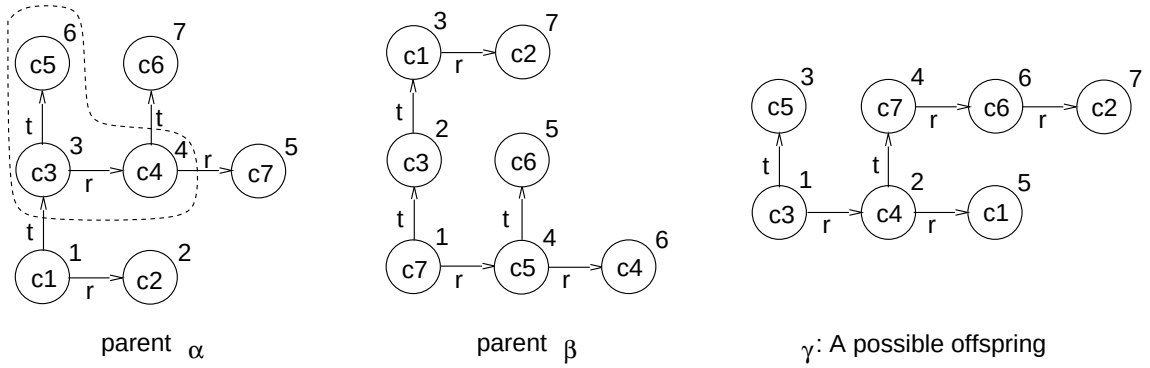

Figure A.4: *Combining α and β.*

E^γ is constructed as follows. From the cell tree of α a connected subset $T_s = (V_s, E_s)$, $V_s \subset V$, $E_s \subset E^\alpha$ is chosen. T_s is chosen at random but subject to the constraint that decoding T_s in the order defined by p^α, i.e. using $c \in V_s \mid \forall\, c' \in V_s \setminus \{c\} : p^\alpha(c) < p^\alpha(c')$ as root, causes no constraint violations. Two different schemes have been tried for the determination of the size of V_s, as will be described in section A.4.1. In Fig. A.4, the chosen T_s is indicated by the dashed line. Initially E^γ is defined to be E_s. Hence, γ has inherited all cells in V_s from α. The remaining cells $V - V_s$ are then inherited from β by extension of E^γ. The cell tree of β is traversed in ascending order according to p^β. At any node it is checked if the corresponding cell c belongs to V_s, that is, whether it has been placed in γ already. If so, the cell is skipped. Otherwise, c is added to the cell tree of γ by extending E^γ. Various schemes for determining the position at which to add c has been tried, see section A.4.1. The transformation of any cell is inherited unaltered together with the cell itself. That is,

$$t^\gamma(c) = \begin{cases} t^\alpha(c) & \text{if } c \in V_s \\ t^\beta(c) & \text{if } c \in V - V_s \end{cases}$$

individuals having the same area will have their fitness values adjusted so that fitness increases as the estimated interconnect length decreases.

The total interconnect length of an individual is estimated as in [7]: Let M denote the number of nets, and let m_k denote the total number of terminals of the k'th net. Let $t_{ki} = (x_{ki}, y_{ki})$ denote the position of terminal i in net k. The *center of gravity* of the k'th net is then defined by

$$T_k = (\overline{x}_k, \overline{y}_k) = \frac{1}{m_k} \sum_{i=1}^{m_k} t_{ki}$$

and the estimated total interconnect length L is defined as

$$L(p) = \sum_{k=1}^{M} \sum_{i=1}^{m_k} \| t_{ki} - T_k \|$$

where $\| x \|$ denotes the usual Euclidean vector norm.

Now suppose that the population is enumerated in ascending order according to F', and that $F'(p_i) = F'(p_{i+1}) = \ldots = F'(p_j)$, $i < j$. Thus, the fitness of $p_i, \ldots, p_j$ must be adjusted according to interconnect length. In order to assure that area always predominates interconnect length, this is done as follows. Sort $p_i, \ldots, p_j$ into decreasing order according to interconnect length, i.e. let us assume $L(p_i) \geq L(p_{i+1}) \geq \ldots \geq L(p_j)$. Define δF_{ij} as

$$\delta F_{ij} = \frac{\delta A}{j - i + 1}$$

where $\delta A = F'(p_{j+1}) - F'(p_j)$. A new fitness value F is then computed as

$$F(p_k) = F'(p_i) + (k - i)\delta F_{ij}, \quad k = i, \ldots, j.$$

Since the fitness values now defined can be very small, they are normalized. Finally, the values are scaled linearly as described in [6]. The purpose of scaling is twofold. At the initial phase of the evolution, a few individuals having very high fitness compared to the average will be very dominating. As a consequence, the search will be limited to a small region of the search space too early. Scaling counteracts this effect by reducing the standard deviation of the fitness at the initial phase of the process. In the final phase of the evolution, the difference between the best and the average individuals tend to be small due to the convergence of the process. Hence selection becomes almost random, thereby reducing the chances of further improvement. At this stage, scaling counteracts this effect by increasing the standard deviation.

then computed as

$$D_s = \begin{cases} \lambda\,[d_s + round(a\sqrt{\frac{h_s}{\lambda}} + b)] & \text{if } d_s > 0 \\ 0 & \text{if } d_s = 0 \end{cases}$$

where h_s is the length of side s of c_i, λ is the spacing in the routing grid, $round(x)$ is the rounded value of x and a and b are user defined parameters. The area inside the solid rectangle shown in Fig. A.3 is uniquely determined by D_s. Cell c_i can be placed at the given position if and only if this area contains no (parts of) cells apart from c_i itself. When $a = b = 0$ the estimated routing area is a lower limit of the area needed by any router regardless of the channel definition. If $a > 0$, the corresponding term of D_s increases with h_s. The argument for this definition is that the longer the channel, the more likely nets will pass through it [18]. Note that moving c_i in any direction may affect the value of D_s for all four values of s.

In summary, given V, the genotype of an individual consists of the relations E and the functions p and t. The genotype (and the decoder) has the important property of implicitly representing most constraints of the problem. This simplifies the design of genetic operators which assures the satisfaction of all constraints at all times.

A.3.2 Fitness Measure

Given (the phenotype of) an individual, its fitness is defined by the positive function F. Fitness is relative to other individuals, and therefore always computed for an entire population at a time. Since the objective is to minimize layout area, initially the auxiliary function F' is defined as

$$F'(p) = \frac{1}{A(R_p) - \sum_{i=1}^{n} A(c_i)}$$

where n is the number of cells of the placement problem, A is the area of a rectangular cell and R_p is the rectangle R of the individual p. That is, $F'(p)$ is the inverse of the total estimated routing area in p. All individuals having equal area will now have equal fitness. But when fixing the total area of a placement, the probability of a 100 % routing completion within the estimated area is likely to increase as the total interconnect length decreases. The minimization of the total interconnect length is therefore introduced as a secondary optimization criterion. All

Any B*-placement can be represented by at least one genotype, i.e., the decoder mapping is not one-to-one. Furthermore, since a placement can only be represented by a genotype if it is a B*-placement, the search space explored by the algorithm is exactly the set of all B*-placements. This is one of the significant differences to the approach in [3], in which the search space is restricted to slicing structures.

A.3.1.2 Routing Area Estimation

When decoding the binary tree, the routing area needed is estimated as each cell is placed. When placing the i'th cell, the distance needed in each direction $s \in S = \{north, east, south, west\}$ to previously placed cells is computed by a function D_s, which depends on all previously placed cells. Each cell is placed according to the B*-strategy and as close to the previously placed cells as allowed by D_s. Figure A.3 illustrates how D_s is computed.

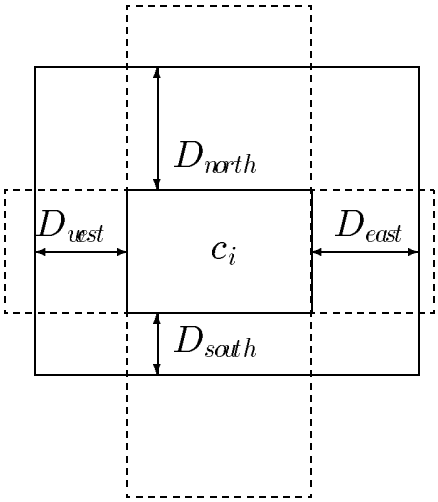

Figure A.3: *Estimation of routing area.*

When testing if cell c_i can be placed at some given position (c_i^{xl}, c_i^{yl}), the four areas indicated by dashed squares are considered. D_s depends on all terminals at side s of c_i and of all terminals in previously placed cells, which are 1) inside the square at side s, 2) placed at some side parallel to side s of c_i and 3) not shadowed by a intervening cell. Given this set of terminals, the channel density d_s is computed as if the square were the routing channel. In order to account for global routing, D_s is

$E_t \cap E_r = \emptyset$. Each node has at most one outgoing top-edge and at most one outgoing right-edge. All edges are oriented away from the root of the tree. Let $e_{ij} \in E$ denote an edge from c_i to c_j and let (c_i^{xl}, c_i^{yl}) and (c_i^{xu}, c_i^{yu}) denote the coordinates of the lower left and upper right corners of c_i, respectively. Then $e_{ij} \in E_t$ (E_r) means that cell c_j is placed above (to the right of) c_i in the phenotype. That is,

$$\forall\, e_{ij} \in E:\ e_{ij} \in E_t \Rightarrow c_j^{yl} \geq c_i^{yu}\ ,\quad e_{ij} \in E_r \Rightarrow c_j^{xl} \geq c_i^{xu}$$

The tree is decoded as follows. The cells are placed one at a time in a rectangular area having horizontal length W and infinite vertical length. Each cell is moved as far down and then as far left as possible without violating the routing area estimate described in section A.3.1.2. The cells are placed in ascending order according to their *priorities* defined by the one-to-one function $p : V \rightarrow \{1, \ldots, n\}$. Any node has higher priority than its predecessor in the tree. In Fig. A.2 the priorities are indicated at the top right hand side of each node. The transformation of each cell is defined by the function $t : V \rightarrow \{0,1,2,\ldots,7\}$, which is also part of the genotype.

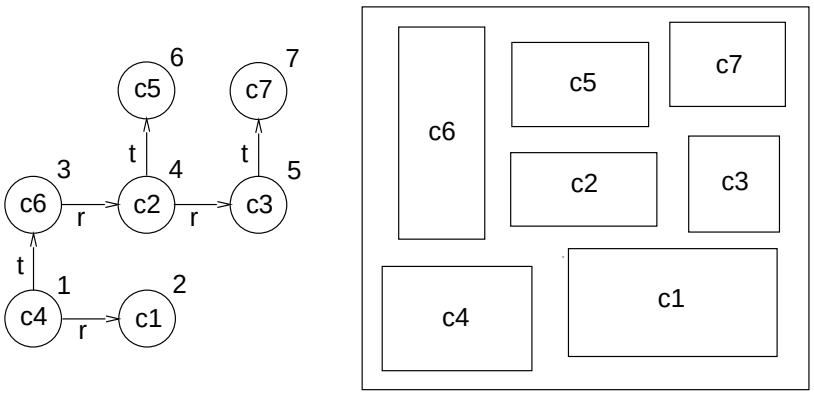

Figure A.2: *An example genotype and the corresponding phenotype.*

When all cells are placed the smallest rectangle, R_{se} enclosing all cells, is computed by the decoder. Each of the four sets of i/o-terminals are then uniformly positioned along the corresponding edge of R_{se}, so that the ordering within each set is preserved. Finally, the rectangle R is constructed by extending R_{se} until the routing area estimate is satisfied along all edges of R. Note that the genotype itself contains no explicit representation of the i/o-terminals and no absolute coordinates of the cells. At the time of computation of a phenotype from a given genotype the decoder determines those quantities.

The drawback of the second strategy is that it is more complex in nature and its associated genetic operators are slow owing to the fact that they insure constraint satisfaction at all times.

While the first strategy is used in [2], the algorithm presented here is based on the second strategy. To avoid too complex and slow operators, a genetic encoding has been developed in which some of the constraints are implicitly represented and therefore need not be considered by the genetic operators.

A.3.1 Genetic Encoding

In GAs a distinction is made between the *genotype* and the *phenotype* of an individual [6]. A genotype is an encoding, or representation, of the information constituting an individual, while the phenotype is the physical appearance of the individual. Reproduction and mutation are performed in terms of genotypes, while fitness is expressed in terms of a phenotype. A *decoder* is used to compute the phenotype corresponding to a given genotype. Estimation of the routing area needed is performed during decoding.

The genetic encoding is inspired by the two-dimensional bin packing problem, which is the problem of compactly packing a number of rectangular blocks into a bin having fixed width and infinite height in such a way that the distance from the top edge of the highest placed block to the bottom edge of the bin is minimized. The standard algorithm for this problem places the blocks one at a time at the downmost and then at the leftmost position. The placement algorithm is based on a generalization of this scheme. For a given instance of the macro-cell placement problem, let a *BL-placement* (bottom-left) denote a solution, in which no cell can be moved further down or to the left without causing a violation of the routing area estimate. The solution space considered by the algorithm is restricted to the set of all possible BL-placements.

A.3.1.1 Genotype and Decoder

Assume that the given problem have n cells $c_1, \ldots, c_n$. An example genotype with $n = 7$ cells is shown in Fig. A.2 together with the corresponding phenotype. A binary tree (V, E), $V = \{c_1, \ldots, c_n\}$, in which the i'th node corresponds to the cell i, represents the absolute positions of all cells. Two kinds of edges exist: top-edges and right-edges, so that $E = E_t \cup E_r$,

simply defined to be the M fittest individuals of $P_C \cup P_N$.

With a small probability, each individual in P_C is now subject to mutation. The main purpose of mutation is to insure that information lost by reproduction can be recovered. In Section A.3.4 four different kinds of mutations are described. If the mutation probabilities are too high, frequent mutations will prevent the convergence of the process and turn it into a random walk. If the mutation rates are too low, the search may prematurely converge to a local minimum only. Following mutation, each individual is subject to an inversion operation with a small probability. The purpose and the operating principle of this operator is explained in Section A.3.5. Each generation is completed by evaluation of all individuals as the basis for the selection to take place in the next generation. Furthermore, the best individual ever seen is updated.

When the last generation has been simulated, an attempt to optimize each of the individuals $P_C \cup \{ q \}$ a little further is made using routine optimize(p), which executes a sequence of fitness-improving mutations on each individual. An exhaustive strategy is used, so that when optimize(p) has been performed, no single mutation can improve p further. The best individual following the optimization then constitutes the resulting macro-cell placement.

When applying a GA to a complex optimization problem, there are two main strategies for handling the constraints imposed on any solution:

1. Allow constraint violations during the optimization process, and control the degree of violation by adding one or more weighted penalty-terms to the fitness function.

2. Ensure that throughout the optimization process, each individual always satisfies every constraint.

The choice of strategy has significant implications. The first choice leads to the simplest and fastest genetic operators, since these need not enforce the satisfaction of all constraints by the produced individuals. However, the fitness measure has become more complicated and more importantly, it may be difficult to adjust the weights introduced in the measure in a way so that the contribution regarding the quality of the solution and the contributions regarding constraint violations are appropriately balanced at all times. On the other hand, the second strategy allows a simple fitness definition, since no penalty terms are needed. Thus, we are guaranteed that the measure always expresses something meaningful.

```
generate(P_C);
evaluate(P_C);
q = best(P_C);
repeat until stopCriterion():
    P_N = ∅;
    repeat M times:
        select p_1 ∈ P_C, p_2 ∈ P_C;
        P_N = P_N ∪ crossover(p_1, p_2);
    end;
    evaluate(P_C ∪ P_N);
    P_C = reduce(P_C ∪ P_N);
    ∀ p ∈ P_C: possiblyMutate(p);
    ∀ p ∈ P_C: possiblyInvert(p);
    evaluate(P_C);
    q = best(P_C ∪ {q});
end;
∀ p ∈ P_C ∪ {q}: optimize(p);
q = best(P_C ∪ {q});
output q;
```

Figure A.1: *Outline of the algorithm*

repeated selection and mating of individuals from P_C, a set of offspring P_N of size M is generated. The selection strategy should reflect the principle of survival of the fittest, and using the terminology of [6], the scheme used here is stochastic sampling with replacement. That is, the individual $p_i \in P_C$ is selected with probability

$$\frac{F(p_i)}{\sum_{p \in P_C} F(p)}$$

where F is the fitness measure, cf. Section A.3.2. The two mates needed for one crossover are selected independently of each other and any individual may be selected any number of times in the same generation. By replacing some individuals in P_C with individuals from P_N, a new current generation P_C emerges. As opposed to the selection scheme used for crossover, the selection performed here is deterministic. The new P_C is

A.3 Description of the Algorithm

The concept of the GA, which was introduced by Holland [8], is based on the idea of optimizing by simulating biological evolution. In nature, the individuals of a population adapts to the environment in which they live. The fittest individuals have the highest probability of survival and tend to increase in numbers, since their reproduction rate is high and their characteristics are inherited by their descendants. On the other hand, the less fit individuals tends to die out. This Darwinian principle of "survival of the fittest" can be used in optimization. Given some optimization problem, e.g. the macro-cell placement problem, define an *individual* to be a solution and define a measure of *fitness* of an individual. Then generate a *population*, and simulate an evolution process. The most important components of this process are *reproduction* and *mutation*, for which application specific operators have to be designed. After simulation of a number of *generations*, highly fit individuals will emerge, which correspond to good solutions of the given optimization problem. A general introduction to GAs is given in e.g. [6].

Many variants of GAs can be found in the literature [2, 4, 6, 17]. An outline of the GA used here is shown in Figure A.1. After briefly presenting an overview of the algorithm its various components are discussed in detail in the following subsections.

Initially, the current population P_C is constructed from randomly generated individuals by routine generate(P_C). The fitness of each individual is computed by evaluate(P_C), described in Section A.3.2. Since the quality of any individual is relative to the rest of the population, computation of fitness requires the complete population as input. Routine best(P_C) selects the best of the given individuals and is used throughout the process to keep track of the best individual q ever seen. Each execution of the outer repeat loop corresponds to a complete simulation of one generation. Throughout the optimization process, the number of individuals $M = |P_C|$ is kept constant. The number of generations to be simulated depends on the progress of the search process itself, as described in Section A.3.6.

Reproduction initiates each generation. Mating is simulated by the crossover operator described in Section A.3.3. Given a pair of individuals, the crossover operator produces one offspring. The overall purpose of crossover is to assure exploration of the promising parts of the search space. Hence the offspring produced have to resemble its parents. By

The macro-cell placement problem is then to compute the following:

- The absolute position of each cell.
- The transformation of each cell, i.e., its orientation and possible reflection(s) in one or both of the axes.
- A rectangle R which defines the shape of the layout.
- For each of the four ordered sets of i/o-terminals, an absolute position along the corresponding edge of R of each terminal in the set.

The objective is to minimize the area of R subject to the following constraints:

- No pair of cells overlap each other.
- The rectangle R encloses all cells and has approximate horizontal length W.
- The i/o-terminals are positioned so that the ordering within each set is preserved.
- The area within R, which is not occupied by cells, is sufficiently large to contain all routing needed to implement the interconnections between the cells as specified by the given netlist.

To meet the last constraint, the necessary routing area is estimated during the placement process. This estimate is based on the following assumptions:

- The area occupied by cells and the area used for routing are disjoint.
- Two layers of metal are used for routing.
- All nets will be treated as signal nets; i.e., all wires will have the minimum width allowed by the technology.

The macro-cell placement problem has been shown to be NP-hard [14]. Furthermore, the solution space is extremely large. If we ignore the placement of i/o-terminals and only consider the placement of n cells so that they constitute a matrix of some predefined shape, we obtain $O(n!8^n)$ as a lower bound on the size of the solution space, since each cell can be transformed in 8 distinct ways.

A.1 Introduction

A large number of algorithms for the placement of cells in VLSI layouts have been developed during the last two decades. A recent survey is given in [16]. At the current state of the art, simulated annealing (SA) is one of the most popular approaches. SA algorithms produces high quality placements at the cost of extensive runtimes.

A less prevalent type of placement algorithm is the genetic algorithm (GA). In [4, 15, 17] GAs for standard cell placement are developed. The performance of these algorithms is comparable to SA algorithms. High quality placements are obtained at the cost of extensive runtimes. To our knowledge only two papers have been published in which GAs for macro-cell placement are presented [3, 2]. As will be accounted for in Section A.3, both algorithms are significantly different from the approach presented here.

A GA for the two-dimensional bin packing problem has been developed by Kröger et al. [9]. The two-dimensional bin packing problem can be seen as the hypothetical special case of the macro-cell placement problem in which no nets exists, i.e. no routing will be performed. In this paper a GA for the macro-cell placement problem is developed based on comprehensive extensions of the genetic encoding and genetic operators found in [9]. The resulting algorithm is capable of producing placements having a quality comparable to the best published results.

A.2 Problem Definition

In the literature, the definition of the macro-cell placement problem varies slightly. Consistent with the specification of the MCNC benchmarks we define the problem as follows.
The input is given as:

- A set of rectangular cells, each of which has a number of terminals positioned along its edges.
- An ordered set of i/o-terminals for each side of the chip under construction. These terminals constitute the interface of the chip.
- A netlist specifying the interconnections between all terminals.
- An approximate horizontal length W of the chip under construction.

Appendix A

A Genetic Algorithm for Macro-Cell Placement

This paper describes an improved version of the algorithm first presented in H. Esbensen, "A Genetic Algorithm for Macro Cell Placement," *Proc. of The European Design Automation Conference*, pp. 52-57, 1992. The improvements are listed in Section 5.1.2, page 64.

Abstract

A new genetic algorithm for macro-cell placement is presented. The algorithm is based on a generalization of the two-dimensional bin packing problem. The genetic encoding and the genetic operators assures that all constraints of the problem are always satisfied. Consequently, the potential problems of the common approach of adding penalty terms to the cost function are eliminated. The algorithm has been tested on MCNC benchmarks and the layout quality obtained is comparable to the best published results.

[Vose 91b] M.D. Vose, G.E. Liepins, "Schema Disruption," *Proc. of The Fourth International Conference on Genetic Algorithms*, pp. 237-242, 1991.

[Weste 88] N.H.E. Weste, K. Eshraghian, *Principles of CMOS VLSI Design*, Addison Wesley, Reading, 1988.

[Winter 87] P. Winter, "Steiner Problem in Networks: A Survey," *Networks*, Vol. 17, pp. 129-167, 1987.

[Winter 92] P. Winter, J. M. Smith, "Path-Distance Heuristics for the Steiner Problem in Undirected Networks," *Algorithmica*, Vol. 7, pp. 309-327, 1992.

[Sunderam 90] V. S. Sunderam, "PVM: A framework for parallel distributed computing," *Concurrency: Practice and Experience*, Vol. 2(4), pp. 315-339, 1990.

[Suzuki 93] J. Suzuki, "A Markov Chain Analysis on A Genetic Algorithm," *Proc. of The Fifth International Conference on Genetic Algorithms*, pp. 146-153, 1993.

[Szymanski 85] T. G. Szymanski, "Dogleg Channel Routing is NP-complete," *IEEE Transactions on Computer-Aided Design*, Vol. 4, No. 1, pp. 31-41, 1985.

[Szepieniec 86] A. A. Szepieniec, "Integrated placement/routing in sliced layouts," *Proc. of the 23rd Design Automation Conference*, pp. 300-307, 1986.

[Takla 94] M. B. Takla, D. W. Bouldin, D. B. Koch, "Early Exploration of the Multi-dimensional VLSI Design Space," *Proc. of The 7th International Conference on VLSI Design*, pp. 413-416, 1994.

[Tate 93] D. M. Tate, A. E. Smith, "Expected Allele Coverage and the Role of Mutation in Genetic Algorithms," *Proc. of The Fifth International Conference on Genetic Algorithms*, pp. 31-37, 1993.

[Upton 90a] M. Upton, K. Sarii, S. Sugiyama, "Simulated Annealing Placement for Mixed Macro Cell and Standard Cell Layouts," *Proc. of the MCNC International Workshop on Layout Synthesis*, Vol. 1, May 1990.

[Upton 90b] M. Upton, K. Sarii, S. Sugiyama, "Integrated Placement for Mixed Macro Cell and Standard Cell Designs," *Proc. of the 27th Design Automation Conference*, pp. 32-35, 1990.

[Venkateswaran 94] R. Venkateswaran, P. Mazumder, "Routing Algorithms in Semiconductor Circuit Design," in preparation, 1994.

[Vose 91a] M. D. Vose, "Generalizing the notion of schema in genetic algorithms," *Artificial Intelligence*, Vol. 50, pp. 385-396, 1991.

[Sechen 88] C. Sechen, "Chip-Planning, Placement and Global Routing of Macro/Custom Cell Integrated Circuits Using Simulated Annealing," *Proc. of the 25th Design Automation Conference*, pp. 73-80, 1988.

[Shahookar 90a] K. Shahookar, P. Mazumder, "A Genetic Approach to Standard Cell Placement using Meta-Genetic Parameter Optimization," *IEEE Transactions on Computer-Aided Design*, Vol. 9, No. 5, pp. 500-511, May 1990.

[Shahookar 90b] K. Shahookar, P. Mazumder, "GASP - A Genetic Algorithm for Standard Cell Placement," *Proc. of the European Design Automation Conference*, pp. 660-664, March 1990.

[Shahookar 91] K. Shahookar, P. Mazumder, "VLSI Cell Placement Techniques," *ACM Computing Surveys*, Vol. 23, No. 2, June 1991.

[Shahookar 94a] K. Shahookar, *Application of the Genetic Algorithm for Computer-Aided Design of VLSI Layout*, Ph.D. thesis, Dept. of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, in preparation, 1994.

[Shahookar 95] K. Shahookar, P. Mazumder, "Genetic Multiway Partitioning," *Proc. of The 8th International Conference on VLSI Design*, 1995 (to appear).

[Sherwani 93] N. Sherwani, *Algorithms for VLSI Physical Design Automation*, Kluwer Academic Publishers, Boston, 1993.

[Smith 93] A. E. Smith, D. M. Tate, "Genetic Optimization Using A Penalty Function," *Proc. of The Fifth International Conference on Genetic Algorithms*, pp. 499-505, 1993.

[Suh 87] J. Y. Suh, D. Van Gucht, "Incorporating heuristic information into genetic search," *Proc. of The Second International Conference on Genetic Algorithms*, pp. 100-107, 1987.

[Rayward-Smith 86] V.J. Rayward-Smith, A. Clare, "On finding Steiner vertices," *Networks*, Vol. 16, pp. 283-294, 1986.

[Reeves 93] C. R. Reeves, "Using Genetic Algorithms with Small Populations," *Proc. of the Fifth International Conference on Genetic Algorithms*, pp. 92-99, 1993.

[Richardson 89] J. T. Richardson, M. R. Palmer, G. Liepins, M. Hilliard, "Some Guidelines for Genetic Algorithms with Penalty Functions," *Proc. of the Third International Conference on Genetic Algorithms*, pp. 191-197, 1989.

[Sahni 80] S. Sahni, A. Bhatt, "The Complexity of Design Automation Problems," *Proc. of the 17th Design Automation Conference*, pp. 402-411, 1980.

[Saravanan 93] N. Saravanan, D.B. Fogel, "A Bibliography of Evolutionary Computation & Applications," Technical Report No. FAU-ME-93-100, Rev. 1.2, Dept. of Mechanical Engineering, Florida Atlantic University, August 1993.

[Schoenauer 93] M. Schoenauer, S. Xanthakis, "Constrained GA Optimization," *Proc. of The Fifth International Conference on Genetic Algorithms*, pp. 573-580, 1993.

[Schultz 90] A. C. Schultz, J. J. Grefenstette, "Improving tactical plans with genetic algorithms," *Proc. IEEE Conf. Tools for AI*, pp. 328-334, 1990.

[Scott 85] W.S. Scott, R.N. Mayo, G. Hamachi, J. K. Ousterhout (Eds.), "1986 VLSI Tools: Still More Works by the Original Artists," Report No. UCB/CSD 86/272, Computer Science Division (EECS), University of California, Berkeley, 1985.

[Sechen 88a] C. Sechen, *VLSI Placement and Global Routing Using Simulated Annealing*, Kluwer Academic Publishers, Boston, 1988.

[Mohan 93] S. Mohan, P. Mazumder, "Wolverines: Standard Cell Placement on a Network of Workstations," *IEEE Transactions on Computer-Aided Design*, Vol. 12, No. 9, pp. 1312-1326, Sept. 1993.

[Mühlenbein 88] H. Mühlenbein, M. Georges-Schleuter, O. Krämer, "Evolution algorithms in combinatorial optimization," *Parallel Computing*, Vol. 7, pp. 65-85, 1988.

[Naderio 89] N.J. Naderio, S. Matsuda, K. Nakajima, "Via Minimization Problem is NP-Complete," *IEEE Transactions on Computers*, Vol. 38, No. 11, pp. 1604-1608, 1989.

[Nishizaki 89] Y. Nishizaki, M. Igusa, A. Sangiovanni-Vincentelli, "Mercury: A New Approach to Macro-cell Global Routing," *Proceedings of the IFIP 10/WG10.5 International Conference on VLSI*, Munich, 1989.

[Nissen 93] V. Nissen, "Evolutionary Algorithms in Management Science," *Papers on Economics and Evolution #9303*, Interdisziplinäres Graduiertenkolleg, Universität Göttingen, Germany, July 1993.

[Nix 92] A. E. Nix, M. D. Vose, "Modeling Genetic Algorithms with Markov Chains," *Annals of Mathematics and Artificial Intelligence*, Vol. 5, No. 1, pp. 79-88, 1992.

[Octtools 93] "Octtools-5.2 User's Guide", Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, 1993.

[Onodera 91] H. Onodera, Y. Taniguchi, K. Tamaru, "Branch-and-Bound Placement for Building Block Layout," *Proc. of the 28th Design Automation Conference*, pp. 433-439, 1991.

[Pattatuda 88] R. Pattatuda, D. Smith, M. Stebnisky, C. Pasdak, P. Patent, "VLA: Fully Automatic Placement Strategies for Very Large Semicustom Designs," *International Conference on Computer Design*, pp. 434-439, 1988.

[Lawler 76] E. L. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart and Winston, New York, 1976.

[Lienig 94a] J. Lienig, K. Thulasiraman, "A Genetic Algorithm for Channel Routing in VLSI Circuits," *Evolutionary Computation*, Vol. 1, No. 4, pp. 293-311, 1994.

[Lienig 94b] J. Lienig, K. Thulasiraman, "A New Genetic Algorithm for the Channel Routing Problem," *Proc. of The 7th International Conference on VLSI Design*, pp. 133-136, 1994.

[Lipsett 89] R. Lipsett, C. F. Schaefer, C. Ussery, *VHDL: Hardware Description and Design*, Kluwer Academic Publishers, Boston, 1989.

[Lucena 92] A. Lucena, J. E. Beasley, "A branch and cut algorithm for the Steiner problem in graphs," working paper, The Management School, Imperial College, England, July 1992.

[Macfarlane 90] D. Macfarlane, I. East, "An Investigation of Several Parallel Genetic Algorithms," In S. J. Turner (Ed.), *Tools and Techniques for Transputer Applications*, IOS Press, pp. 60-67, 1990.

[McCoy 94] B. A. McCoy, G. Robins, "Non-Tree Routing," *Proc. of the European Design and Test Conference*, pp. 430-434, 1994.

[Mead 80] C. Mead, L. Conway, *Introduction to VLSI Systems*, Addison-Wesley, Reading, 1980.

[Michalewicz 91] Z. Michalewicz, C. Z. Janikow, "Handling constraints in genetic algorithms," *Proc. of the Fourth International Conference on Genetic Algorithms*, pp. 151-157, 1991.

[Michalewicz 92] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer-Verlag, 1992.

[Michalewicz 93] Z. Michalewicz, "A Hierarchy of Evolution Programs: An Experimental Study," *Evolutionary Computation*, Vol. 1, No. 1, pp. 51-76, 1993.

[Janikow 91] C. Z. Janikow, Z. Michalewicz, "An Experimental comparison of binary and floating point representations in genetic algorithms," *Proc. of the Fourth International Conference on Genetic Algorithms*, pp. 31-36, 1991.

[De Jong 75] K. De Jong, *Analysis of the Behaviour of a Class of Genetic Adaptive Systems*, PhD thesis, Dept. of Communication Sciences, University of Michigan, 1975.

[De Jong 93] K. De Jong, W. Spears, "On The State of Evolutionary Computation," *Proc. of The Fifth International Conference on Genetic Algorithms*, pp. 618-623, 1993.

[Kapsalis 93] A. Kapsalis, V. J. Rayward-Smith, G. D. Smith, "Solving the Graphical Steiner Tree Problem Using Genetic Algorithms," *Journal of the Operational Research Society*, Vol. 44, No. 4, pp. 397-406, 1993.

[Karp 72] R. M. Karp, "Reducibility among Combinatorial Problems," In R. E. Miller, J. W. Thatcher (Eds.), *Complexity of Computer Computations*, Plenum Press, New York, pp. 85-103, 1972.

[Kirkpatrick 83] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, Vol. 220, pp. 671-680, 1983.

[Koza 92] J. R. Koza, *Genetic Programming. On the Programming of Computers by Means of Natural Selection*, MIT Press, Cambridge, 1992.

[Kozinski 91] K. Kozinski, "Benchmarks for Layout Synthesis - Evolution and Current Status," *Proc. of the 28th Design Automation Conference*, pp. 265-270, 1991.

[Kröger 91] B. Kröger, P. Schwerdtfeger, O. Vornberger, "Genetic Packing of Rectangles on Transputers," Transputing '91, Vol. 2, IOS Press, 1991.

[Goldberg 90] D. E. Goldberg, "The theory of virtual alphabets," in H. P. Schwefel, R. Männer, eds., *Parallel Problem Solving from Nature*, Springer Verlag, pp. 13-22, 1990.

[Goldberg 91] D. E. Goldberg, K. Deb, B. Korb, "Don't Worry, Be Messy," *Proc. of the Fourth International Conference on Genetic Algorithms*, pp. 24-30, 1991.

[Goldberg 94] D. E. Goldberg, "Genetic and Evolutionary Algorithms Come of Age," *Communications of the ACM*, Vol. 37, No. 3, pp. 113-119, 1994.

[Grefenstette 86] J. J. Grefenstette, "Optimization of Control Parameters for Genetic Algorithms," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 16, No. 1, pp. 122-128, 1986.

[Grefenstette 87] J. J. Grefenstette, "Incorporating Problem Specific Knowledge into Genetic Algorithms," in L. Davis, editor, *Genetic Algorithms and Simulated Annealing*, Chapter 4, pp. 42-60, Pitman, 1987.

[Grefenstette 89] J. J. Grefenstette, J. E. Baker, "How Genetic Algorithms Work: A Critical Look at Implicit Parallelism," *Proc. of the Third International Conference on Genetic Algorithms*, pp. 20-27, 1989.

[Herrigel 89] A. Herrigel, W. Fichtner, "An Analytic Optimization Technique for Placement of Macro-Cells," *Proc. of the 26th Design Automation Conference*, pp. 376-381, 1989.

[Holland 75] J. H. Holland, "Adaption in Natural and Artificial Systems," *University of Michigan Press*, Ann Arbor, MI, 1975.

[Horn 93] J. Horn, "Finite Markov Chain Analysis of Genetic Algorithms with Niching," *Proc. of the Fifth International Conference on Genetic Algorithms*, pp. 110-117, 1993.

[Eschermann 88] B. Eschermann, W.-M. Dai, E. S. Kuh, M. Pedram, "Hierarchical Placement for Macrocells: A "Net in the Middle" Approach," *Proc. of the International Conference on Computer-Aided Design*, pp. 460-463, 1988.

[Fiduccia 82] C. M. Fiduccia, R. M. Mattheyses, "A Linear-Time Heuristic for Improving Network Partitions," *Proc. of the 19th Design Automation Conference*, pp. 175-181, 1982.

[Fonseca 93] C. M. Fonseca, P. J. Fleming, "Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization," *Proc. of the Fifth International Conference on Genetic Algorithms*, pp. 416-423, 1993.

[Gajski 88] D. D. Gajski (Ed.), *Silicon Compilation*, Addison-Wesley, 1988.

[Glasmacher 91] K. Glasmacher, A. Hess, G. Zimmermann, "A Genetic Algorithm for Global Improvement of Macrocell Layouts," *Proc. of the International Conference on Computer Design*, pp. 306-313, 1991.

[Glasser 85] L. A. Glasser, D. W. Dobberpuhl, *The Design and Analysis of VLSI Circuits*, Addison-Wesley, Reading, 1985.

[Goldberg 87] D. E. Goldberg, P. Segrest, "Finite Markov Chain Analysis of Genetic Algorithms," *Proc. of the Second International Conference on Genetic Algorithms*, pp. 1-8, 1987.

[Goldberg 89a] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.

[Goldberg 89b] "Sizing Populations for Serial and Parallel Genetic Algorithms," *Proc. of the Third International Conference on Genetic Algorithms*, pp. 70-79, 1989.

[Todd 91] S. Todd, D. Mcfarlane, C. Harland, "Optimisation of Artificial Neural Network Structure using Genetic Techniques implemented on multiple Transputers," In P. Welch, D. Stiles, T. L. Kunii, A. Bakkers (Eds.), *Transputing '91*, IOS Press, Vol. 2, pp. 687-700, 1991.

[Donath 80] W.E. Donath, "Complexity Theory and Design Automation," *Proc. of the 17th Design Automation Conference*, pp. 412-419, 1980.

[Esbensen 90] H. Esbensen, H. S. Mølzen, A. Nielsen, "HIR: En prototype af en router til understøttelse af hierarkisk design af VLSI-kredsløb," Masters thesis, Computer Science Department, Aarhus University, Denmark, May 1990 (in Danish).

[Esbensen 92] H. Esbensen, "A Genetic Algorithm for Macro Cell Placement," *Proc. of The European Design Automation Conference*, pp. 52-57, 1992.

[Esbensen 94a] H. Esbensen, P. Mazumder, "SAGA: A Unification of the Genetic Algorithm with Simulated Annealing and its Application to Macro-Cell Placement," *Proc. of The 7th International Conference on VLSI Design*, pp. 211-214, 1994.

[Esbensen 94b] H. Esbensen, P. Mazumder, "A Genetic Algorithm for the Steiner Problem in a Graph," *Proc. of The European Design and Test Conference*, pp. 402-406, 1994.

[Esbensen 94c] H. Esbensen, "Computing Near-Optimal Solutions to the Steiner Problem in a Graph Using a Genetic Algorithm," Technical report DAIMI PB-468, Aarhus University, Denmark, Feb. 1994.

[Esbensen 94d] H. Esbensen, "A Macro-Cell Global Router Based on Two Genetic Algorithms," *Proc. of The European Design Automation Conference*, pp. 428-433, 1994.

[Dai 89] W.M. Dai, B. Eschermann, E. S. Kuh, M. Pedram, "Hierarchical Placement and Floorplanning in BEAR," *IEEE Transactions on Computer-Aided Design*, pp. 1335-1349, Vol. 8, No. 12, 1989.

[Dasgupta 94] P. Dasgupta, P. Mitra, P. P. Chakrabarti, S. C. De-Sarkar, "Multiobjective Search in VLSI Design," *Proc. of The 7th International Conference on VLSI Design*, pp. 395-400, 1994.

[Davidor 89] Y. Davidor, *Epistasis Variance - Suitability of a Representation to Genetic Algorithms*, Technical Report CS89-25, The Weizmann Institute of Science, Israel, Dec. 1989.

[Davidor 90] Y. Davidor, *An Intuitive Introduction to Genetic Algorithms as Adaptive Optimizing Procedures*, Technical Report CS90-07, The Weizmann Institute of Science, Israel, April 1990.

[Davidor 91] Y. Davidor, "A genetic algorithm applied to robot trajectory generation," in L. Davis, editor, *Handbook of Genetic Algorithms*, Chapter 12, pp. 144-165, Van Nostrand Reinhold, 1991.

[L. Davis 87] L. Davis, M. Steenstrup, "Genetic Algorithms and Simulated Annealing: An Overview," in L. Davis, editor, *Genetic Algorithms and Simulated Annealing*, Chapter 1, pp. 1-11, Pitman, 1987.

[L. Davis 89] L. Davis, "Adapting Operator Probabilities In Genetic Algorithms," *Proc. of the Third International Conference on Genetic Algorithms*, pp. 61-69, 1989.

[L. Davis 91] L. Davis, *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, 1991.

[T. E. Davis 91] T. E. Davis, J. C. Principe, "A Simulated Annealing Like Convergence Theory for the Simple Genetic Algorithm," *Proc. of the Fourth International Conference on Genetic Algorithms*, pp. 174-181, 1991.

[Cascetto 86] A. Cascetto, F. Romeo, A. Sangiovanni-Vincentelli, "A Parallel Simulated Annealing Algorithm for the Placement of Macro-Cells," *Dig. Tech. Papers, IEEE Intl. Conf. on Computer-Aided Design*, pp. 30-33, 1986.

[Chan 91] 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, pp. 49-77, Nov. 1991.

[Chopra 92] S. Chopra, E. R. Gerres, M. R. Rao, "Solving the Steiner Tree Problem on a Graph Using Branch and Cut," *Operations Research Society of America Journal of Computing*, Vol. 4, No. 3, pp. 320-335, 1992.

[Cohoon 86] J. P. Cohoon, W. D. Paris, "Genetic Placement," *Proc. of the International Conference on Computer-Aided Design*, pp. 422-425, 1986.

[Cohoon 91a] J. P. Cohoon, S. U. Hedge, W. N. Martin, D. Richards, "Distributed Genetic Algorithms for the Floorplan Design Problem," *IEEE Transactions on Computer-Aided Design*, Vol. 10, pp. 483-492, April 1991.

[Cong 92] J. Cong, A. B. Kahng, G. Robins, M. Sarrafzadeh, C. K. Wong, "Provably Good Performance-Driven Global Routing," *IEEE Transactions on Computer-Aided Design*, Vol. 11, No. 6, June 1992.

[Lai 87a] W. M. Lai, M. Sato, E. S. Kuh, "A Dynamic and Efficient Representation of Building-Block Layout," *Proc. of the 24th Design Automation Conference*, pp. 376-384, 1987.

[Lai 87b] W. M. Lai, E. S. Kuh, "Simultaneous Floor Planning and Global Routing for Hierarchical Building Block Layout," *IEEE Transactions on Computer-Aided Design*, Vol. 6, pp. 828-837, 1987.

[D. Beasley 93c] D. Beasley, D. R. Bull, R. R. Martin, "Reducing epistasis in combinatorial problems by expansive coding," *Proc. of the Fifth International Conference on Genetic Algorithms*, pp. 400-407, 1991.

[J. E. Beasley 89] J. E. Beasley, "An SST-Based Algorithm for the Steiner Problem in Graphs," *Networks*, Vol. 19, pp. 1-16, 1989.

[J. E. Beasley 90] J. E. Beasley, "OR-Library: distributing test problems by electronic mail," *Journal of the Operational Research Society*, Vol. 41, pp. 1069-1072, 1990.

[Boseniuk 91] T. Boseniuk, W. Ebeling, "Boltzmann-, Darwin-, Haeckel-Strategies in Optimization Problems," *Proc. of the 1st Workshop on Parallel Problem Solving from Nature*, pp. 430-444, Springer-Verlag, 1991.

[Bramlette 91] M. F. Bramlette, "Initialization, mutation and selection methods in genetic algorithms for function optimization," *Proc. of the Fourth International Conference on Genetic Algorithms*, pp. 100-107, 1991.

[Bui 98] T. N. Bui, B. R. Moon, "Hyperplane Synthesis for Genetic Algorithms," *Proc. of the Fifth International Conference on Genetic Algorithms*, pp. 102-109, 1993.

[Burstein 82] M. Burstein, "Amon 'placement/routing' approach to automation of VLSI layout design," *Proc. of the 1982 International Symposium on Circuits and Systems*, pp. 756-759, 1982.

[Caprani 91] O. Caprani, P. Møller-Nielsen, "An Occam Model of Two-Phase Clock Driven CMOS Circuits," In L. Asnell, M. Ottenren (Eds.), *Nordic Transputer Applications*, IOS Press, pp. 148-165, 1991.

Bibliography

[Andersen 92] J. Andersen, O. Olsen, "C++ as a Hardware Description Language and some Implications for Verification and Test," *NORCHIP Workshop on Verification and Test*, March 1992.

[Atkinsse 89] J. Atkinsse, "A New Interpretation of Schema Notation that Overturns the Binary Encoding Constraint," *Proc. of the Third International Conference on Genetic Algorithms*, pp. 86-91, 1989.

[Bäck 93] T. Bäck, H-P. Schwefel, "An Overview of Evolutionary Algorithms for Parameter Optimization," *Evolutionary Computation*, Vol. 1, No. 1, pp. 1-23, 1993.

[Bapat 93] S. Bapat, J. P. Cohoon, "A Parallel VLSI Circuit Layout Methodology," *Proc. of The 6th International Conference on VLSI Design*, pp. 236-241, 1993.

[Bapat 91] S. Bapat, J. P. Cohoon, "Sharp-Looking Geometric Partitioning," *Proc. of the Second European Design Automation Conference*, pp. 172-176, 1991.

[D. Beasley 93a] D. Beasley, D. R. Bull, R. R. Martin, "An Overview of Genetic Algorithms: Part 1, Fundamentals," *University Computing*, Vol. 15, No. 2, pp. 58-69, 1993.

[D. Beasley 93b] D. Beasley, D. R. Bull, R. R. Martin, "An Overview of Genetic Algorithms: Part 2, Research Topics," *University Computing*, Vol. 15, No. 4, pp. 170-181, 1993.

has been given. Yet, the GA will probably remain a relatively time consuming approach for the problems considered. However, as long as runtimes are reasonable from a practical point of view, there will be many situations where designers are willing to spend the time required to obtain a slightly better solution.

Important topics of future work include a needed update of the optimization criteria and the way they are combined. Furthermore, the accuracy of the crucial estimations should be improved, which ultimately leads to the consideration of simultaneous placement and global routing. Judging from the characteristics of the problems, the general characteristics of the GA and the results presented in this thesis, it is likely that the GA will be well suited for approaching the integrated problem.

Chapter 7

Conclusion

The purpose of this thesis has been to investigate the suitability of the genetic algorithm (GA) for placement and global routing of macro-cell layouts, assuring that the main objective is to obtain the best possible layout quality. Four specific questions of interest were formulated in Section 1.1, page 3, and in the following the answers are summarized.

When comparing the performance of the developed algorithms to the best existing approaches using any other method, the GAs are competitive with respect to layout quality. Consequently, the main conclusion of this thesis is that the GA is a very promising approach to the problems considered. This conclusion is further emphasized by the fact that this work amounts to three man-years only, while a much larger effort has been invested in the use of e.g. simulated annealing for these applications. Hence, significant future improvements are likely.

The performance of the developed GAs is clearly superior to that of a previous GA for macro-cell placement and a previous GA for the Steiner problem in a graph (SPG). In both cases the most significant difference between the algorithms is the strategies used for constraint handling. The main conjecture of this thesis is that the design principle of enforcing constraint satisfaction at all times is the main reason for the obtained performance. Hence, for problems having characteristics similar to those considered here, traditional binary encodings and/or standard genetic operators should be abandoned whenever they obstruct enforcement of constraint satisfaction.

A number of problems have been identified, among which the runtime requirements is the most serious. The current implementations of the developed algorithms are very slow compared to other approaches, with the SPG algorithm being the only exception. It is believed that the runtimes can be reduced significantly, and a number of reasons for this

topology of the model, i.e., the number of subpopulations, the number of processing elements per subpopulation, the connections between local controllers, etc. is defined at runtime as the first task of the global controller. This makes it easy to experiment with various degrees of distributed selection, various communication patterns between subpopulations, etc. As a special case, when the number of subpopulations equals one, selection is global as in a sequential GA. Since the main bulk of the work is performed by the processing elements, each of these are assigned to a physical processor, and the local controllers and the global controller are then evenly distributed among the processors. The model has not yet been applied to parallelize any of the developed GAs. The main reason is that although the current sequential implementations are not runtime competitive, the runtime requirements are not an obstacle for the practical experimentation either.

The typical hardware available at VLSI design sites is a set of interconnected workstations. Consequently, from an application point of view parallel implementations of the algorithms presented in this thesis should be targeted for such hardware. A good speedup of a GA can still be obtained on interconnected workstations despite the fact that the communication is very slow compared to that of other MIMD architectures. This is demonstrated in [Mahan 93] which presents a parallel implementation of the GA for standard cell placement introduced in [Shahookar 90]. A number of systems exists, which supports the development of parallel algorithms to be executed on a set of heterogeneous machines interconnected by a network. One such system is PVM (Parallel Virtual Machine) [Sunderam 90].

Comparing the characteristics of the algorithms presented in this thesis to the literature on parallel GAs, there is no reason that a good speedup should not be obtainable for these algorithms. This is another reason why the work on this issue has been given a quite low priority.

one subpopulation to another. The question is to which extent selection should be distributed as described, and to investigate this experimentally a model of a parallel GA was developed which is illustrated in Fig. 6.1. The model includes as special cases each of the models farming, diffusion and migration discussed in [Muhlenbein 90, Todd 91].

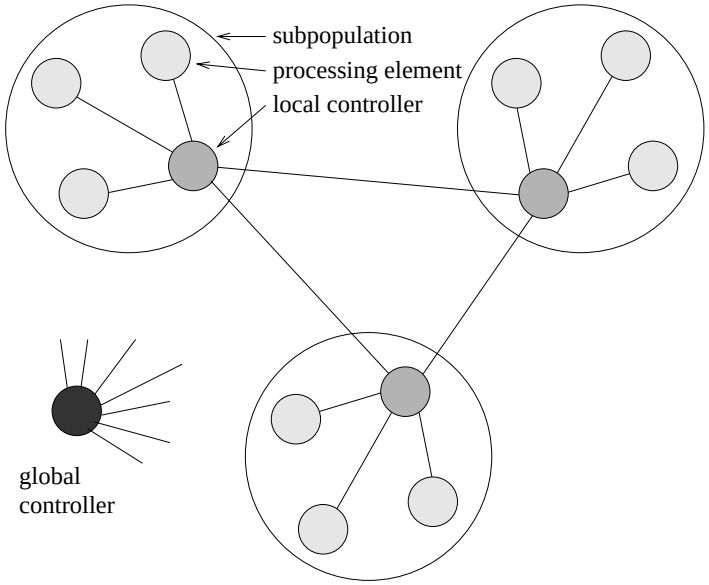

Figure 6.1: *An example configuration of the parallel GA having three subpopulations each of which have three processing elements. The connections of the global controller to all other processes are not shown.*

Three types of processes exists: processing elements, local controllers and a single global controller. A subpopulation consist of one local controller and a number of processing elements. The local controller performs the selection within the subpopulation and then delegates tasks to the processing elements. Each task correspond to one or more executions of a genetic operator on individuals of which copies are given. Occasionally, (copies of) individuals may move from one subpopulation to another, as indicated by the interconnections of the local controllers. The global controller is connected to all other processes, i.e., all local controllers and all processing elements. The tasks of the global controller are to take care of i/o, distribute the problem definition, collect statistics and results and control termination. Since all communications are asynchronous, the subpopulations never have to wait for each other, while within each subpopulation the local controller has to synchronize its processing elements once per generation due to the selection.

This model has been implemented on a transputer network. The

relaxed accordingly.

Interrupting decodings of solutions when an upper bound on cost is exceeded could also potentially improve runtime. Assume for example a steady-state GA, in which generated offspring always replaces the current worst solution. If, during a decoding process initiated by the crossover operator, it can be determined that the cost of the partly generated offspring will exceed the cost of the current worst solution, then the decoding of that offspring can be interrupted since it will not be inserted in the population anyway.

Although the improvements mentioned above are expected to be very effective if implemented, it is still an open question if the placement algorithms would be able to handle the larger benchmarks Ami33 and Ami49 in a reasonable amount of time. Some kind of hierarchical partitioning as used by e.g. the BB algorithm [Cohoon 91] and the BEAR system [Dai 87b] would probably also be needed.

6.3.2 Parallel Genetic Algorithms

Consider the characteristics of the GA with the objective of parallel processing in mind. In each generation, long sequences of the operations performed by the genetic operators are independent, i.e., they can be performed in any order. The operators rely on local information only (the input individuals), and they require non-uniform amounts of time because of the different types of operations and because of their stochastic nature. The only sequential element is the selection, which depends on the relative fitness of the individuals and hence requires knowledge of the fitness of all individuals at a certain point in time. These characteristics of the GA are the reason that the algorithm is well suited for parallel implementation on MIMD architectures. A vast literature on this topic exists and generally good speedups are reported. Notable work on parallel GAs for combinatorial optimization is done by e.g. Mühlenbein et al. [Mühlenbein 88].

The key issue when implementing a parallel GA is how to handle selection. One possibility is to relax the idea that *every* individual has a probability greater than zero of being chosen in every selection step. This can be done by dividing the population into a number of subpopulations and restrict selection so that parents are always chosen from the same subpopulation. To allow information to spread throughout the entire population, it should also be possible for an individual to move from

implementation is an important advantage of this algorithm as compared to other optimization methods such as e.g. simulated annealing.

6.3.1 Improvements of the Sequential Algorithms

The main reason for the excessive runtime requirements of the current implementations of the placement algorithms and the global router is the repeated computation of channel densities, cf. Sections 5.2.2 and 5.4.1. As various positions for a block is tried out by the placement algorithms, channel densities involving the same set of instances are computed over and over again. The only difference from one computation to the next is that the block to be placed has been moved slightly. Currently, the channel density is computed from scratch wherever a new position is tried out. Significant amounts of computation could thus be eliminated by using a data structure which allows the channel density to be dynamically updated as a block is shifted along one dimension. The situation is similar for the global router. Here the sets of terminals involved in the channel density computations are fixed, while the varying factor is the nets entering/leaving a channel. Hence, computation time could be reduced significantly by using data structures which allowed the density of a channel to be dynamically updated as the sets of nets entering/leaving the channel were altered. In the current implementation, whenever a density is needed, it is computed from scratch. It is not even checked if the routing of the channel has changed, i.e., if a new computation is at all needed.

Adapting the idea of the *steady-state GA* is another potential possibility of reducing runtime. In a steady-state GA, the crossover operator is applied only one per generation, and the generated offspring is inserted immediately into the population, for example by replacement of the current poorest solution(s). This contrasts the scheme used in the developed algorithms, in which a pool of n offspring is generated in each generation, assuring a population size of n. By eliminating the synchronous concept of the generation, the steady-state GA potentially allows good solutions to spread throughout the population much faster than the generational GA, in terms of number of performed crossings. In other words, the algorithms presented here probably spends much time in each generation producing numerous offspring in each generation which are never used. However, to avoid that the steady-state GA converges prematurely to a local optimum, the factors controlling the selection pressure should be

any limitations, and all nets are restricted to patterns of a certain type.

The BEAR layout system allows floorplanning and global routing to be closely interleaved. As described in Section 4.1.4, in BEAR a clustering tree is constructed initially, and floorplanning is then performed during a top-down traversal of the tree. Global routing can be incorporated into this traversal as described in [Dai 87b]. At each level of the hierarchy, a global routing graph is extracted when the floorplan has been determined. All nets are then globally routed in terms of the routing graph. When proceeding to the next lower level of the hierarchy, the global routing graph is refined accordingly and the global route of each net is refined in terms of the newand more detailed graph. Consequently, as the tree traversal progresses towards the leaves, the global route for each net becomes increasingly accurate. Furthermore, the global routing performed at each level affects the succeeding placement steps, which is exactly the kind of feedback wanted. A fast computation of the Steiner trees corresponding to global net routes on a given level of the hierarchy is required. For this purpose, the choice of floorplans is restricted so that the extracted global routing graphs always have a certain form in which a minimum Steiner tree can be computed in linear time. However, this scheme of simultaneous placement and global routing is not used in the later version of BEAR presented in [Dai 89, Eschermann 88].

A recent approach which closely integrates placement and global routing is SHARP, developed by Rapat and Gloor [Rapat 91, Rapat 93]. The basic idea is recursive partitioning combined with a collection of pre-computed optimal Steiner trees which can be used at each level of the hierarchy. Global routing is refined as lower levels of the hierarchy is considered and hence from this overall viewpoint, the basic ideas are quite similar to those of [Dai 87b]. In [Rapat 93] the performance of SHARP is compared to that of TimberWolf and is found to be inferior with respect to solution quality but about five times faster.

6.3 Runtime Problems

It is expected that the runtime of the presented algorithms can be reduced significantly by improving the sequential algorithms as described in Section 6.3.1. Another and independent approach to speed up computation is to parallelize the algorithms, as outlined in Section 6.3.2. The fact that the CAis parallel by nature and hence very well suited for parallel

the GA context a key issue then becomes that of determining the relative fitness of solutions while at the same time avoiding expressing solution quality by a single figure of merit. In [Fonseca 93] it is described how this can be done so that the GA searches for (samples of) the *Pareto-optimal set*, which is the set of solutions in which no solution can be improved with respect to any single criterion without degrading the value of at least one other criterion. The Pareto-set is what the designer is looking for initially. When (samples of) this set is known, the search can be focussed on certain interesting subsets by incorporating constraints for some of the criteria. As pointed out in [Fonseca 93] the GA is very well suited for this kind of optimization/exploration because of its built-in simultaneous investigation of many alternative solutions.

6.2 Simultaneous Placement and Global Routing

The inherent problems of the artificial separation of the mutually dependent placement and global routing tasks have been a main theme through Chapters 4 and 5. To make the synthesis process manageable, the tasks were separated when this field emerged and the separation has been a standard assumption ever since. But during the years a better understanding of the problems have developed and it now seems natural to investigate the possibilities of a reunification of the problems. Clearly, at least in principle solution quality should improve when a problem is solved as a whole rather than by combining solutions of dependent subproblems. On the other hand, the unified problem is of course harder to solve than the subproblems, so in practice the question is whether the unified problem can be solved sufficiently well for the potential quality improvement to appear.

Existing work on simultaneous placement and global routing is extremely limited [1]. For the simple design style of gate arrays, a scheme for simultaneous placement and routing was proposed as early as 1982 in [Burstein 82]. An approach for building block layouts restricted to slicing structures is presented in [Szepieniec 86]. However, a number of strict assumptions prevents the practical applicability of the algorithm. For example, routing can be performed over the entire chip area without

[1] It is not clear exactly what the term "simultaneous" means in this context, but for the discussion in this Section it is sufficient to think of a very close integration of the two tasks.

nets are still trees, but the criterion minimized is the radius. McCoy and Robins abandons the implicit assumption used throughout the literature for years, that a net is a tree [McCoy 94]. Using the so-called Elmore model of delay, they show that non-tree routings may significantly improve the delay at a relatively small cost in terms of wirelength.

No matter which delay model is used, to adapt the SPC algorithm to delay minimization, it is necessary and sufficient to replace the decoder (and alter the cost computation). Necessity follows from the fact that by executing an SPC-heuristic, the current decoder relies on knowledge of the criterion optimized. Sufficiency is a consequence of problem-specific knowledge being exploited by the decoder only. Hence, a new decoder should implement a heuristic which given a set of selected vertices generates a feasible, spanning graph of reasonable low cost in terms of the delay model used. A good performance would be expected, cf. the fifth conjecture of Section 5.5.

Turning to the question of the balancing of various possibly competing objectives, recall that in all presented algorithms area is given higher priority than wirelength. Since both quantities are measured using estimates rather than exact values, the strict priorities may in some cases deteriorate result quality as pointed out in Section 5.2.1. However, from a practical point of view an even more important issue is the actual needs of the designer. When entering the layout synthesis phase, the designers knowledge of the properties of the future circuit will be limited to quite rough estimates as provided by high level synthesis tools. Therefore, during the layout synthesis phase the designer will typically be interested in exploring various trade-offs between competing objectives such as e.g. area, speed and power consumption. Initially the overall objective will often not be clearly defined, as it depends on the actual and still unknown possibilities. Therefore, initially the designer is interested in a set of solutions reflecting the possible trade-offs of the criteria considered, rather than a single solution. Then, as knowledge of the possibilities is acquired, the designer may wish to enforce constraints on some criteria (now known to be satisfiable) and then explore possible trade-offs among the remaining criteria to obtain a balanced solution satisfying the constraints.

This kind of multi-objective optimization/exploration is investigated from the VLSI point of view in [Dasgupta 94, Takala 94], and from the CAD point of view in [Fonseca 93]. The quality of a solution is expressed by a vector of values, in which each entry corresponds to a specific criterion. In

Chapter 6

Future Work

Many aspects of the work presented in this thesis warrants further research. Based on the evaluations of Chapter 5 this Chapter discusses three particular important directions of possible future work. Section 6.1 is a follow up on the discussion from Sections 5.1.1 and 5.2.1 on the choice of optimization criteria and how they should be balanced. The consequences of the strong mutual dependence of the placement and global routing tasks has been a recurrent theme in preceding chapters. This suggests a closer integration of these tasks which is the topic addressed in Section 6.2. Finally, Section 6.3 presents a number of possibilities for reducing the observed runtime problems, cf. Sections 5.2.1 and 5.4.1.

6.1 Optimization Criteria

The issue of optimization criteria involves two main questions, addressed in the following: 1) Which criteria should be optimized? 2) How should they be combined?

As pointed out in Section 5.1.1 the choice of minimising area is highly relevant from a practical point of view, while the minimization of total wirelength should be replaced by the explicit minimization of delay. The effort required to implement this change depends on the extent to which the algorithms exploit problem-specific knowledge. For the placement algorithms, the change is merely a question of replacing the total wirelength estimate with a delay estimate. However, for the global routing algorithm the situation is more complicated. To minimize delay the SPG-algorithm repeatedly used in the first phase should minimize the delay of a net rather than the net length. Various models of delay exists. In [Cong 92] delay is defined from the *radius* of a net, which is the maximum route length from the source terminal to any sink terminal. Hence,

- An experience from this work is that from a practical point of view it is in fact quite easy to find settings of the GA control parameters which yields good performance. It is definitely easier than one might think after having consulted the literature, in which quite a few papers are concerned with the issue of parameter values, cf. Section 3.4.4. This is not to say that a fixed set of parameters will do for any algorithm. On the contrary, due to the complicated interactions between various selection strategies, the (non-binary) encoding used, etc. it is necessary to perform a series of experiments to find good settings whenever a new GA has been designed. However, this process is tedious rather than difficult. From experience with previous algorithms of a similar nature, one has a (rather small) interval of feasible values for each parameter. Finding a combination which works well can then be performed in some systematic way as has been done here. It is also likely that the meta-GA approach described in Section 3.4.4 is a good way of automating this work process, which only has to be done once and for all whenever a new algorithm is developed.

- Estimations of area and interconnect length should be accurate, since as already mentioned, the estimates effectively amounts to adding noise to the cost functions. If the noise margin is not minimized, the work performed by the GA in the late phase, in which most of the time is spend and only relative small improvements obtained, may be pointless. This situation suggests the use of estimates of which the accuracy is increased dynamically as the optimization process progresses.

- The use of inversion operators is being debated among researchers and there are conflicting views as to whether inversion should be used or substituted by other techniques, cf. Section 3.4.1 and [Bäck 93, Goldberg 91]. As mentioned in Section 5.1.3 all GAs developed in this work applies inversion operators, and all experiments which addressed inversion, turned out in favour of the operator. Hence, the inversion operators is one of the contributors to the obtained performance.

- The SPG-algorithm performs better than the other GAs developed in the sense that it is also runtime competitive. It is conjectured that by generalizing the principles of the SPG-algorithm, high-performance GAs for a larger class of graph algorithms can be obtained. A simple bitstring can specify selected vertices and/or edges which should be (part of) a solution to a given problem. By using a fast, deterministic heuristic for that problem, or some other repair algorithm, the decoder can insure that any bitstring is interpreted as a feasible (and possibly reasonably good) solution. Standard genetic operators can be applied. Examples of problems which presumably are well suited for this approach, are the maximum independent set problem, the maximum clique problem and graph colouring [10].

[10] In the case of graph colouring the genotype should be a string of integers specifying a colour for each vertex.

5.5 Overall Evaluation and Conjectures

Summarizing the evaluation of the developed algorithms, they are all highly competitive with respect to solution quality. For the placement and global routing algorithms, this comes at the cost of excessive computation times. In contrast, the SPG-algorithm is also competitive on runtime. Overall, the obtained results are very encouraging, especially considering the development time invested in this project, which is about 3 man-years. Significantly more time has probably been spend on a system such as TimberWolfMC.

The key question is of course what are the reasons for the obtained performance? The following *conjectures* are believed to be main elements of the answer:

- First of all, the complexity of the problems makes them well suited for GAs, cf. Section 3.1.2. This claim is supported by the fact that the developed GAs make very limited use of problem-specific knowledge, cf. Section 5.1.3. In other words, the results are not obtained by GAs heavily mixed with other problem-specific techniques, but of rather "pure" GAs [9].

- Due to the nature of the problems, constraints should be handled by enforcement, avoiding penalty terms in the cost function, cf. the discussion of Section 3.4.3. An additional reason is that the drawbacks of penalty terms become even more pronounced for the problems considered here because of the involved estimates, which effectively amounts to adding noise to the cost functions. Some experimental indications supports this claim. The most significant difference between the GA for the SPG presented in Appendix C and the GA in [Kapsalis 93] is the constraint handling strategies, enforcement versus penalty. The performance results are very clearly in favour of the GA using enforcement. Similarly, comparing the GA for placement to GAVP, one of the major differences is the constraint handling strategies. Again the algorithm using enforcement clearly performs best with respect to solution quality, while here computation times are equal within a factor of two or three.

[9] The developed representations are highly problem-specific, but in accordance with the distinctions introduced in Section 3.4, the representation and the use of problem-specific knowledge are considered two distinct issues.

5.4.2 Estimations of Routing Area and Interconnect Length

The estimate of total wirelength computed by the GA is exactly as the estimate used in TimberWolfMC which is clearly more accurate than that of Mercury due to the different strategies for positioning terminal vertices.

Comparing the routing area estimate of the GA to that of Mercury, the GA performs the most accurate computation. As explained in Section 4.2.2, in Mercury the width of the channel corresponding to an edge of the routing graph is estimated by a fixed contribution and a flexible, that latter of which is proportional to the number of nets entering or leaving the channel. In contrast, the GA computes the exact channel density considering all nets and using exact terminal locations. On the other hand, Mercury is capable of adjusting the placement within certain bounds, while the GA relies on the compactor to do that. The total area estimation of the GA is based on the assumption that all channels defining the height and width of the final layout, will be compacted to their minimum width by the compactor. If that is not the case, the GA will underestimate the total area.

A common problem of the area estimations of Mercury and the GA is that they both rely on the routing graph topology to be preserved. As is described in Section D.3.1, page 210, if the placement is adjusted after global routing so that the topology of the corresponding routing graph is altered, then the area estimate may become very inaccurate or even meaningless. Another way of saying this is that the given placement has to be sufficiently good to assure that it will only need minor adjustments later on. Again, this reflects the strong mutual dependence between the placement and global routing tasks. While the routing graph topology is preserved in all examples of Table 5.9, Section D.4.3, page 216, includes examples on what may happen when that is not the case. The issue of narrowing the gap between placement and global routing is brought up in Section 6.2.

Since there is no stochastic variation when using TimberWolfMC, only one layout were generated using that router. Table 5.9 compares the total area, the routing area (i.e., the total area minus the sum of the cell areas) and the total wirelength of the completed layouts [8]. Each entry is computed as $100(GA_{res} / TW_{res} - 1)$, where GA_{res} is the result using the GA and TW_{res} is the result using TimberWolfMC. Hence, a negative value is an improvement in percent of using the GA. The GA is clearly superior to TimberWolfMC with respect to layout quality. For Xerox, the area reduction is obtained by increasing the wirelength, while for the other examples, area as well as wirelength is reduced. However, the quality improvement comes at a high price. While TimberWolfMC spends about 30 seconds routing each of Xerox and Ami33 and about 5 minutes routing Ami49, on average the GA requires about 22 minutes for Xerox, 12 minutes for Ami33 and 130 minutes for Ami49. All values are elapsed time on a Sun Sparc IPX. The GA spends most of its time computing channel densities, which are computed for all edges of the routing graph whenever a new routing solution is evaluated. By keeping track of the actual need for recomputing channel densities it is expected that the runtime could be significantly improved.

Circuit	Solution	Total area	Routing area	Wirelength
Xerox	best	−1.9	−4.7	−0.0
	avg.	−1.4	−3.5	−0.8
Ami33	best	−3.0	−4.7	−1.5
	avg.	−1.1	−1.7	−0.2
Ami49	best	−4.2	−7.3	−4.0
	avg.	−3.7	−6.3	−2.9

Table 5.9: *Relative improvements obtained by the GA compared to TimberWolfMC. Best and avg. is best and average of the five runs performed for each circuit.*

[8] The circuits listed as Xerox, Ami33 and Ami49 corresponds to xerox-M, ami33-2-M and ami49-2-M, respectively, in Table D.2, page 217.

5.4 Evaluation of the Global Router

The following Section 5.4.1 evaluates the layout quality and computation time of the global router, while the quality of the estimates are commented in Section 5.4.2. Throughout these sections, the term *GA* refers to the global router presented in Appendix D, *TimberWolfMC* refers to the global router of that system [Sechen 88a, Sechen 88b] (Section 4.2.1) and *Mercury* is the approach of [Nishizaki 89] (Section 4.2.2).

5.4.1 Performance

Since the GA minimizes area and secondarily total interconnect length, Mercury minimizes the length of critical nets and secondarily area, and TimberWolfMC minimizes total interconnect length, the performance of the GA should preferably be compared to that of Mercury. Unfortunately, for purely technical reasons that was not possible[6]. Instead the GA was compared to TimberWolfMC. Despite of the distinct optimization criteria, this comparison still provides some insight. In the first phase of global routing, both routers attempts to find the shortest possible routes, and consequently their phase two algorithms are given similar input. Furthermore, the selection of short routes often leads to reasonably small areas.

In other respects the comparison of the two routers is presumably as fair as this kind of comparison will ever be. All input placements are generated by Puppy, an SA-based tool interfaced to Octtools and both routers are also interfaced to Mosaico/Octtools. Consequently, all steps of the layout synthesis process, except the global routing itself, is performed by the same set of tools, which makes it fair to compare the completed layouts. Furthermore, the routers are run on the same machine, which makes the computation times reasonably comparable.

Placements of the MCNC benchmarks Xerox, Ami33 and Ami49 were used for the experiments. Due to a technical problem it was necessary to remove all i/o-terminals (pads) from the examples[7]. For each example, five completed layouts were generated using the GA for global routing,

[6] Mercury is included in Octtools version 5.2 installed at Aarhus University, but our version of the program is not functioning.

[7] In Octtools 5.2, the placement of pads produced by the program Padplace can not be handled by the channel definition program Atlas. This is the same problem that prevents the use of the GA-based router on placements generated by SAGA, as mentioned in Section 5.2.1.

Table 5.8 compares the absolute computation times required by SPHI, BK1, BK2 and the GA. For each algorithm and each class of graphs, the two entries gives the minimum and maximum computation time required to solve one problem. I.e., considering BK1 and class C, the easiest (for BK1) of the 20 problems was solved in 10 seconds while the hardest (for BK1) required 45,848 seconds. For the GA, the values listed are the minimum and maximum of the average values obtained for each graph. When comparing the absolute values of Table 5.8 one should keep in mind the different machines used. BK1 are run on a VAX8700 and BK2 is run on a Sun Sparc 2. For SPHI and the GA, a Sun Sparc IPX was used for graphs from classes B, C and D, while a DEC Vaxstation 5000-240 were used for the graphs of class E.

	BK1		BK2		SPHI		GA	
Class	min	max	min	max	min	max	min	max
B	-	-	0	18	0	1	0	4
C	10	45,848	5	20,726	61	11,374	79	601
D	47	245,192	37	304,380	486	679,000	504	3,441
E	179	-	411	-	7,334	4.3×10^7	7,395	29,105

Table 5.8: *Comparison of computation times. All values are CPU seconds on the respective machines used. A hyphen indicates a non-available value. The maximum values of SPHI for classes D and E are estimates.*

Despite the use of different machines it is clear that the GA runtimes are always very moderate compared to any of the other algorithms. Both BK1 and BK2 are able to solve some problems extremely fast, and much faster than the GA. But for other problems, the runtime of the branch-and-cut algorithms explode and even prevents some problems from being solved. The maximum values for class E are not available for these algorithms, since BK1 failed to solve one of the problems within a 10 day CPU-limit, while for BK2, [Lucena 92] only lists runtimes for 5 of the 20 graphs, presumably because of runtime problems. Considering the estimated times of SPHI for the largest graphs, from a practical point of view this algorithm is not able to handle all problems either.

Summarizing from Tables 5.7 and 5.8, the GA generates solutions which are very close to the global optimum with a high probability. The GA is also competitive with respect to runtime and is the only algorithm capable of generating a solution for all problems within a reasonable amount of time.

- A branch-and-cut approach by Chopra et al. [Chopra 92] denoted here by BC1.
- The branch-and-cut approach by Lucena and Beasley [Lucena 92] described in Section 4.2.3 and denoted here by BC2.

The benchmark data used are from the ORLibrary [J. E. Beasley 90] and consists of 78 randomly generated STP instances, which are divided into four classes B, C, D and E according to size. Graphs in class B have at most 100 vertices, while in classes C, D and E, each graph has 500, 1000 and 2500 vertices, respectively. The number of vertices to be spanned varies from 5 to half of the vertices of the graph, and average vertex degree varies from 2.5 to 50. Hence the largest graphs have 62,500 edges. Class B contains 18 graphs while each of classes C, D and E contains 20 graphs. Optimal solutions are known for all examples, which were initially found by a branch-and-cut algorithm executed on a Cray supercomputer [J. E. Beasley 89].

	SPH-I		GA	
Class	= 0%	< 1%	= 0%	< 1%
B	94.4	94.4	100.0	100.0
C	50.0	80.0	78.0	93.5
D	35.0	65.0	77.5	92.5
E	20.0	45.0	55.0	85.0

Table 5.7: *Comparison of the solution quality obtained by the SPH-I and the GA.*

As is accounted for in Section C4.5, page 190, the genetic algorithm presented in [Kapsalis 93] is clearly inferior to all other algorithms considered here, and are not discussed further in this Section. While BC1 and BC2 finds a global optimal solution for all graphs which they can handle, this is of course not the case for SPH-I and the GA. Table 5.7 compares the solution quality obtained by the latter algorithms. For each class of graphs and each algorithm, two entries of the table gives the percentage of all performed executions which gave a solution within 0 %, respectively 1 % from the global optimum. Since SPH-I is deterministic it was executed once for each graph in each class. The GA was executed 10 times on each graph in classes B, C and D, and once per graph in class E. So, as an example, 92.5 % of the $10 \times 20 = 200$ executions of the GA on the graphs of class D gave a solution within 1 % from the global optimum, while in 77.5 % of these runs a global optimum was found.

easily be constructed for which neither NRP nor GASCA can ever find a globally optimal solution. Terminal locations can be assigned to the placement of Fig. 5.1 (c) so that this placement corresponds to the global optimum for the completed layout.

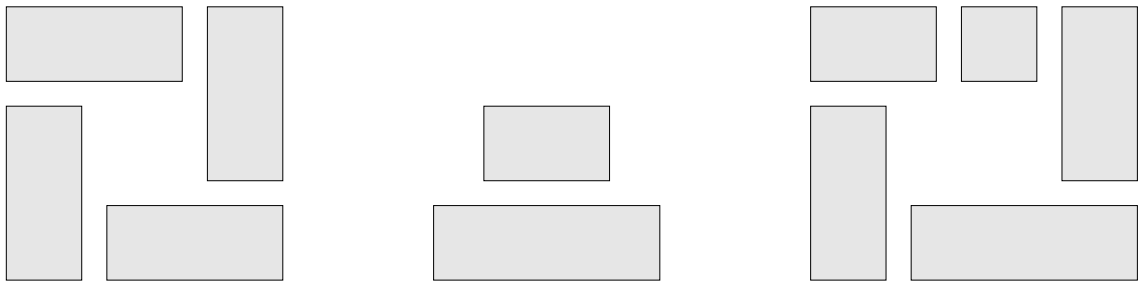

Figure 5.1: *(a) A BL-placement which is not a slicing-structure, (b) A slicing-structure which is not a BL-placement, and (c) a placement which is neither a BL-placement nor a slicing-structure.*

These potential problems of GASCA and NRP do not disclose themselves on the benchmarks. On the contrary, these systems are the ones obtaining the best layout quality. But the problem could surface on other examples and this is more likely for GASCA than for NRP since for a given problem the space of slicing-structures is presumably larger than the space of BL-placements. Furthermore, the reduction to slicing-structures has some advantages with respect to routing which the BL-placements do not, cf. Section 2.3.

5.3 Evaluation of the Steiner Tree Algorithm

In this Section the term GA refers to the algorithm for the SPG presented in Appendix C. Its performance has been compared to that of four other approaches:

- A deterministic heuristic denoted here by SPHI, described in Section C.4.2, page 182. According to a comparative study presented in [Winter 92] SPHI is among the very best deterministic heuristics, and is superior to e.g. a popular algorithm by [Hayward-Smith 86].
- A genetic algorithm by Kapsalis et al. [Kapsalis 93], which to my knowledge is the only genetic algorithm for the SPG published prior to this work.

than the other systems. However, TimberWolfMC includes a third factor which depends on the location of the channel in the layout, so that channels close to the center of the circuit will be wider, cf. Section 4.1.5. There is no doubt that this is a good idea. Congestion at the center region of a circuit may significantly affect the size of channels everywhere, and this phenomenon is neither captured by MBP nor the CA.

Finally, some comments on the area estimation used in the CA, concerning the relation between the two terms measuring local and global congestion. It should be noted that independent of the setting of the parameters α and β, the global routing factor is not equivalent to simple block expansion, since if the channel density is zero, abutment of blocks will be allowed. However, the presence of user-defined parameters is a drawback, although it is not too difficult to find suitable settings. For a given problem, the expected number of terminals involved in a channel density computation can be estimated, for example by assuming that all terminals are uniformly distributed over all block sides. This gives bounds on the possible channel densities, i.e., the magnitude of the local congestion term. The desired magnitude of the global routing term can then be determined, considering block lengths and the total number of nets. Nevertheless, this issue requires further investigation aiming at eliminating the parameters.

Another concern is the inbalance of the two terms with respect to their accuracy. The local congestion estimate is very accurate but also extremely time consuming and the main reason for the extensive computation time of the CA. Combining it with a less accurate term raises the question whether the time spend computing channel densities is fully justified, although the accurate estimation is presumably one of the reasons for the layout quality obtainable by the CA.

5.2.3 Search Space Reductions

CA and SACA differs significantly from the other approaches discussed by the way the search space is reduced. The restriction to BB-placements followed from the view of the placement problem as a generalization of a bin-packing problem. In contrast, MBP uses the common reduction of the search space to slicing-structures, while neither BB, CAMP or TimberWolfMC reduces the space explored. As illustrated in Fig. 5.1, placements exist which are contained in the search space explored by MBP but not by CA/SACA and vice versa. Furthermore, problems can

5.2.2 Estimations of Routing Area and Interconnect Length

As stated previously the issue of estimations is a crucial one. The challenge is to find a suitable balance between accuracy and computation time. The accuracy determines the amount of noise on the cost function optimized. In the following the estimates used in the GA (and hence SAGA) are discussed and compared to those used in the other systems presented in Chapter 4. However, since the nature of the estimations of the BEAR system are significantly different from those used by the other systems, comparison with BEAR is difficult and have been omitted.

Starting with wirelength, MIMAP, GANGA and TimberWolfMC all estimates total wirelength by the sum of the half-perimeters of all nets, and BBI implicitly uses the half-perimeters in the cost function estimating total area. In contrast, the GA estimates the length of each net by a sum of the Euclidean distance from each terminal to a constructed center-point of the net, as described in Section 5.1.2. The half-perimeter of a net is a theoretical lower bound on the required wirelength. For two-terminal nets the GA also underestimates the wirelength, but for nets with three or more terminals the wirelength may be overestimated. Consequently, it is hard to say which estimate is the most accurate. Since the computation time for wirelength estimation is not likely to be a serious bottleneck in any of the algorithms, time is hardly an important criterion here.

Turning to area estimation, the simplest strategy is static block expansion, i.e. initially expanding each block by a fixed amount in each direction, depending on the terminals. This scheme, which is applied by BBI and GANGA, can be characterized as being inaccurate but very fast to compute. However, in BBI the estimated area is also indirectly affected by the estimated wirelength through the cost-function. The area estimates of MIMAP, TimberWolfMC and the GA has some common features. In all three systems, the estimated width of a channel involves two (artificially separated) contributions, one accounting for local congestion, and one accounting for global routing, i.e., nets passing through the channel. In MIMAP and TimberWolfMC the local congestion is estimated by a simple function of the number of involved terminals, while in the GA, the exact channel density is computed. In all three systems, the global routing contribution is a simple function of the length of the channel. Hence, the accuracy of the global routing contributions are about the same for the three systems, while the GA estimates local congestion more accurately

Benchmark	System	Area		Wirelength		Time
		absolute	relative	absolute	relative	
Ami49	SAGA	53.58	1.00	489	1.40	55
	GA	53.99	1.01	563	1.61	52
	BB	54.05	1.01	460	1.31	-
	MP	54.77	1.02	350	1.00	3
	GAMP	61.80	1.15	591	1.69	28
Xerox	MP	25.79	1.00	601	1.08	9
	BB	26.17	1.01	628	1.13	67
	GA	26.58	1.03	556	1.00	156
	SAGA	27.15	1.05	679	1.22	220
	BEAR	28.47	1.10	633	1.14	2
	MOSAICO	29.01	1.12	650	1.17	-
	VITAL	31.17	1.21	866	1.56	-
	GAMP	32.60	1.26	1,088	1.87	68
Hp	SAGA	11.81	1.00	261	1.31	51
	MP	11.85	1.00	200	1.00	5
	GA	11.95	1.01	262	1.31	55
	BB	12.15	1.03	278	1.39	-
	GAMP	-	-	365	1.83	40
Ami33	BB	2.24	1.00	109	1.20	89
	MP	2.42	1.08	91	1.00	49
	BEAR	2.83	1.26	131	1.44	13
	VITAL	3.12	1.39	135	1.48	-
	MOSAICO	3.16	1.41	152	1.67	-
	GAMP	-	-	279	3.07	112
Ami49	MP	48.79	1.00	904	1.00	178
	BB	51.49	1.06	1,021	1.13	-
	GAMP	-	-	-	-	359

Table 5.6: *Comparison of layout qualities and computation times. Absolute area is the core area in mm^2 and absolute wirelength is the total interconnect length in mm. Time is CPU-time in minutes. To ease comparisons, relative areas and wirelengths are also given by normalizing the best result for each benchmark to the value 1.00. For each circuit the results are ordered according to obtained area. A hyphen indicates that the value is not available. Regarding the areas obtained by GAMP for Hp and Ami33, the values given in [Chan 91] are not comparable to the other values of this table, presumably due to incompatible scalings.*

It is interesting to compare the total wirelengths obtained by the GA/SAGA to those of MP. While on Xerox there are no significant differences, for Apte and Hp the GA/SAGA values are 31 - 61 % higher than those of MP. Three factors can contribute to these significant differences:

- To some extent there is a tradeoff between area and wirelength. Hence it is likely that a very small area is obtained because of longer routes for some nets. However, this can hardly account for much of the observed difference.
- The fitness computation of the GA/SAGA always gives higher priority to area than wirelength. If solution A has a much larger estimated total wirelength than B but just a slightly smaller estimated area, then A is always considered better than B. Hence, this priority relies heavily on a very accurate routing area estimation, and will not always be reasonable. This issue is discussed further in Section 6.1.
- The global routing of the GA/SAGA placements could be inferior to that of MP's placements. As mentioned previously, the GA/SAGA placements are input to Mosaico, which uses the global router of TimberWolfMC described in Section 4.2.1. As will be described in Section 5.4.1, the GA-based router presented in Appendix D is clearly superior to TimberWolfMC both with respect to area and wirelength. Consequently, it is very likely that by using the GA-based global router rather than TimberWolfMC, the areas and wirelengths reported for the GA and SAGA would improve further. Unfortunately, due to purely technical problems this have not been confirmed experimentally [5].

Regarding the reservation concerning the fairness of comparing completed layouts, it can be noted from the last point above that the GA/SAGA are probably disadvantaged with respect to the influence of global routing.

[5] The GA-based global router is interfaced to Octtools version 5.2, installed at Aarhus University. In this Octtools version some bugs not present in version 5.1 prevents i/o-terminals (pads) from being handled. Specifically, the channel definition program Atlas can not handle the placement of pads generated by Padplace. The global router could then be interfaced to Octtools 5.1 installed at University of Michigan. But because of other differences between the two Octtools versions, this integration would require a significant amount of work.

- The listed CPU-times are for the placement tools only, i.e., they do not include routing and compaction, etc. The time for BB is measured on a DEC3100 while MVP uses an Apollo DN4000. For BEAR the times listed are elapsed time rather than CPU-time and are measured on a VAX 8800 with workload 10.5-12.5. Finally, GASP is run on a Sun Sparc 1+ and the GA and SAGA on a DEC Mips 5000-240. For the latter two algorithms the listed run-times are average values.

The remaining of this Section comments on the results listed in Table 5.6, keeping in mind the above reservations. Both SAGA and GA performs very well with respect to area, the main layout quality criterion. For Apte and Hp, the best published results are obtained by SAGA, although differences between the best results are small. For Xerox, the two algorithms are inferior to MVP and BB, but are still doing better than e.g. BEAR. Turning to runtime, we see that the GA and SAGA requires about the same amounts of time and that they are significantly slower than all other algorithms. The runtime requirements of the algorithms prevents Ami33 and Ami49 from being placed. Where sufficient results are available, GA/SAGA are about 2-3 times slower than BB and GASP, the other GA-based approach. Compared to MVP the algorithms are 10-20 times slower. The machine used for the GA/SAGA is presumably at least as fast as the other machines, which further amplifies these differences. On the other hand, sequential programs can often be speeded up significantly by a mere programming effort. Only very limited time has been spend on this for the GA/SAGA.

It is possible that to some extent the small improvements obtained by using significantly more computation time merely reflects the nature of the problems. Many researchers have worked with these benchmarks, which have been available at least since 1990. As the gap to the global optima is narrowed it becomes increasingly hard to obtain even small improvements. Despite such factors, the runtime requirements of the GA/SAGA are not satisfactory, and at least the algorithms should be able to handle all benchmarks. In Section 6.3 a number of possibilities of improving runtime are discussed, and I believe that significant improvements can be obtained.

system [Gasotto 86] and the VIPLA system [Petatunda 88] are referenced here as found in [Dai 89, Upton 90a]. No results for TimberWolfMC are given since none were found for the MCNC benchmarks.

Benchmark	Cells	Nets	Terminals
Apte	9	97	287
Xerox	10	203	698
Hp	11	83	309
Ami33	33	123	522
Ami49	49	408	953

Table 5.5: *Characteristics of the MCNC benchmark circuits. The number of terminals includes the number of i/o-terminals (pads) to be positioned along the periphery of the circuit.*

All listed areas and total wirelengths are measured for the completed layouts, i.e., after routing and compaction, etc. In other words, since it is not possible to compare placements directly, the table compares the performance of complete layout systems including routers and compactors, etc. At least three other factors complicates the comparison

- The exact problem definitions used varies slightly. For example, the BBL system defines an aspect ratio goal for some circuits and a width goal for others.

- Some approaches are stochastic and others are deterministic. Average results could then be given for the stochastic ones. However, the results for MP, GA/HAP and possibly other approaches are "best of a small number of runs".[3] An argument for this is that a slightly better result will be obtained by executing the stochastic algorithm a few times, which is useful if layout quality is the top priority. Of course the obvious effect on computation time should then be taken into account. To make the comparison to the other stochastic approaches as fair as possible, the results for GA and SAGA listed in Table 5.6 are the best obtained of ten runs. Average results and standard deviations are given in appendices A and B. All GA and SAGA results have been obtained using the parameter settings given in Section 5.1.3[4] and routing and compaction, etc. has been done using Mosaico, cf. Section 5.1.

[3] Personal communication with authors of [Chan 91, Upton 90a].

[4] SAGA dynamically adjusts the values of some of the parameters, but the values listed in Table 5.4 are valid in the initial generation.

As can be seen from the table the parameter values used are quite similar. Variations in population sizes and stop criteria reflects various tradeoffs between solution quality and computation time. The latter increases at least linearly with the population size since the work required in each generation increases linearly and the number of generations needed for convergence is likely to increase, too. For CAcute, if $r_k = 2.0$ the mutation rate for net k is 0.0475, i.e. similar to the mutation rate of CAsyn. The mutation rate for CAplace is significantly higher, which coincides with the views of [Tate 93] who states that "...problems requiring non-binary encodings may benefit from mutation rates much higher than those generally used with binary encodings." Extensive experiments have shown that the algorithms are quite robust to changes of the parameter settings. For example, altering the mutation rate by e.g. 10 % will not affect performance much.

5.2 Evaluation of the Placement Algorithms

The following Sections compares the placement algorithms with other approaches. Section 5.2.1 compares layout quality and computation times, Section 5.2.2 compares the area and wirelength estimations and Section 5.2.3 discusses the differences regarding search space reductions. Throughout these Sections, *GA* will refer to the algorithm presented in Appendix A, *SAGA* refers to mixed-mode executions of the algorithm presented in Appendix B, *MBP* is the approach of [Upton 90a] (Section 4.1.1), *BB* refers to [Cohoon 91] (Section 4.1.2), *GAMP* is the approach of [Chan 91] (Section 4.1.3), *BEAR* refers to [Lai 87b] (Section 4.1.4) and *TimberWolfMC* refers to the placement algorithm used in the TimberWolfMC system [Sechen 88a] (Section 4.1.5).

5.2.1 Performance

The main characteristics of the MCNC benchmarks [Kozminski 91] used for performance evaluation are listed in Table 5.5.[1] The obtained results are compared to other approaches in Table 5.6, page 76, which to the best of my knowledge includes all the best published results. The MOSAICO[2]

[1] A sixth circuit was added recently but no results have been reported yet.

[2] This should not be confused with Mosaico, the macro-cell layout system which is part of Octtools [Octtools 93]. We will use capital letters for the placement algorithm described in [Casotto 86].

In GAplace most constraints are enforced through the representation. However, both the decoder and the genetic operators have to actively consider constraint violations since infeasible solutions can be expressed as a genotype. In GAsparg constraint satisfaction is enforced by the decoder exclusively.

	GAplace	GAsparg
Representation	yes	no
Decoder	yes	yes
Genetic operators	yes	no

Table 5.3: *Applied constraint enforcement methods.*

Selection of Parameter Values

Of the four strategies for finding suitable parameter settings for a GA listed in Section 3.4.4, the second alternative have been used for all algorithms. A fixed set of parameter values have been found for each algorithm based on experiments and general guidelines provided in the literature. The resulting setting has then been used for all program executions for which results are reported. That is, no data specific tuning have been performed.

	GAplace	GAsparg	GAroute
Population size	25	40	40
Stop criterion	200	50	100
Mutation rate	0.025	0.005	$0.1 - 0.025(r_k - 1)$
Inversion rate	0.05	0.1	0.1

Table 5.4: *The fixed values used for the control parameters.*

Table 5.4 lists the values used. The stop criterion is the number of consecutive generations during which no improvement has been observed upon termination of the algorithm. Since all mutation operators performs "point-wise" mutations, the mutation rates are the probabilities that a specific component of a genotype is mutated when the individual is subjected to mutation. For GAroute, r_k is the number of alternative routes of the k'th net. Hence, the integer value identifying a specific route for net k is altered with the listed probability. The inversion rate is the probability that a given individual is subjected to inversion in a given generation.

Exploitation of Problem-Specific Knowledge

In Table 5.2 the four ways of exploiting problem-specific knowledge presented in Section 3.4.2 are listed and the algorithms classified accordingly.

	GAplace	GAsparg	GAroute
Heuristics in decoder	no	yes	no
Heuristics in existing operators	no	no	no
New operators added	no	no	no
Seeding of initial population	no	no	yes

Table 5.2: *Exploitation of problem-specific knowledge in the algorithms.*

GAplace do not exploit any problem-specific knowledge. Although rather complex, the decoder as well as the genetic operators merely assures feasibility of the generated individuals and do not attempt to discard poor solutions. Section A.4.1, page 128 describes experiments on exploiting problem-specific knowledge in the crossover operator of GAplace. These attempts did not result in any improvement of the performance of the algorithm.

As mentioned when describing the GA template applied, all algorithms uses hill climbers, i.e., they have added operators. However, these are not problem-specific but simply executes a sequence of fitness-improving rotations. Summarizing, it is evident that the use of problem-specific knowledge in the algorithms is very limited, and hence leaves a large potential for improvement of the algorithms, cf. Section 3.4.2 and [Michalewicz 93].

Constraint Handling

In a sense any problem have some constraints which needs to be satisfied. For example, a solution may have to be an integer value or it may be restricted to a certain interval. Such constraints are trivial in a GA context as they are easily enforced. Therefore, when talking about "constraints" and "constraint handling", what is meant is (handling of) non-trivial constraints. As stated in the GA template description, all algorithms applies the strategy of enforcing constraint satisfaction at all times. Table 5.3 lists the three ways of enforcing constraints introduced in Section 3.4.3. Since the problem definition for GAroute contains no constraints, that algorithm is excluded.

to a great extent the performance of a GA is the result of the complex interaction of a number of design choices made. For example, a statement like "selection scheme A for crossover is better than selection scheme B" rarely makes sense on its own, since it depends on a number of other issues such as e.g. the selection scheme used for survival into the next generation.

In the following the algorithms are categorized in terms of the issues discussed in Section 3.4. The placement algorithms described in Appendices A and B are identical as far as their GA parts are concerned, and hence are treated here as one algorithm referred to as *GAplace*. The GA for the SPG (Appendix C) is referred to as *GAspg* and *GAroute* denotes the GA for global routing (Appendix D), i.e., the GA used in the second phase of the global router.

Encoding and Search Space Reductions

Table 5.1 lists the genetic encodings used and indicates whether the search space is reduced or not. The search space considered by GAplace is limited to B*-placements only, cf. Section 5.1.2, while the search space reduction in GAspg stems from an upper bound on the number of Steiner vertices in a minimum Steiner tree, deduced by [Lawler 76]. Hence, the search space reduction has the form of an upper limit on the number of Steiner vertices selected by any individual, i.e., the number of 1's in any binary string. It is important to note that a global optimum is guaranteed to exist within the reduced search space considered by GAspg.

	GAplace	GAspg	GAroute
Encoding	binary tree	bitstring	integer array
Search space reduction	yes	yes	no

Table 5.1: *Representations and the use of search space reductions.*

This is not the case for GAplace, since the global optimum may not be a B*-placement, as will be illustrated in Section 5.2.3. Another major difference is the way the search space reduction is incorporated into the algorithms. In GAplace the representation and the decoder enforces the reduction, while in GAspg it is handled exclusively by the genetic operators.

5.1.3 GA Template and Design Decisions

All GAs developed in this work complies to the same template by having the following properties in common:

1. Let n be the population size. In each generation, a pool of n offspring is generated by repeated use of the crossover operator. From the resulting total of $2n$ individuals, the n best is then chosen deterministically for survival into the next generation.

2. The scheme used for selecting the parent individuals for crossover is *stochastic sampling with replacement* [Goldberg 89a]: The two individuals are selected independently of each other and each individual is selected with a probability proportional to its fitness. Every individual can be selected any number of times in the same generation.

3. An inversion operator is used.

4. The number of generations is not fixed. Instead the simulation is stopped when no improvement has been observed for a user-defined number of consecutive generations.

5. Throughout the process the best individual which has ever existed is recorded. This scheme should not be confused with the common scheme of assuring that the best individual in a generation is never deleted or deteriorated. The latter means that fitness of the current best individual as a function of time is monotonically increasing, while the first does not.

6. After the last generation, the best individual ever seen (and possibly some more) are optimized further using simple hill climbers.

7. All constraints are handled exclusively by enforcement. Consequently, no cost/fitness-function involves any penalty terms.

This template was not fixed initially and then used all the way. On the contrary, for most of the algorithms, lots of experimentation have been done with each of these issues. The above template is the outcome of these experiments. It turned out to yield the best performance among the alternatives tried and consequently have been used for the final versions of all algorithms. When comparing specific items of the template to alternative possibilities from the literature, one should keep in mind that

mark data including random graphs of up to 2,500 vertices and 62,500 edges. Furthermore, I came to know of the high-performance SPG approaches discussed in Section 4.2.3. The algorithm was improved, mainly by adding reduction tests, and the implementation was improved, reducing runtime as well as memory requirements. The new version of the algorithm was compared among others to the branch-and-cut approaches of Section 4.2.3, and results were published in [Esbensen 94c] which is the paper reproduced in Appendix C.

Global Routing Based On Two GAs

The global router explicitly minimizes area as the main criterion and total wirelength as a secondary criterion. In phase one, the only criterion is to generate short routes, i.e., each net is an instance of the SPG. Two-terminal nets are handled by Lawler's algorithm [Lawler 76] while nets with three or more terminals are handled by the GA described above. Terminal vertices are added at exact locations as in TimberWolfMC [Sechen 88a].

For the second phase a new GA was developed. A solution is represented by a string of integers, where the i'th integer identifies the route chosen for the i'th net. Routing area is estimated using polar graphs as in Mercury [Nishizaki 89]. However, as opposed to Mercury, the estimate is based on computation of the exact channel density for each edge of the routing graph, improving the accuracy of the estimation. This is only possible since terminal vertices were added at exact positions in phase one, and for the same reason the estimate of total wirelength is also accurate. The initial population of the GA is seeded with the solution consisting of the shortest route found for each net, since this solution will also usually have a relatively small area. This seeding does not improve the final layout quality obtained, but speeds up convergence. The global router has been published as [Esbensen 94a] reproduced in Appendix D.

small, the total wirelengths are generally relatively high compared to layouts produced by other systems. One possible reason is that the global router used could be inferior to the routers used by the other systems. For these reasons, following the placement work, it was decided to design a global router.

The choice fell on the two-phase strategy (cf. Section 4.2) since the layouts considered are relatively small and since the successful routers TimberWolfMC [Sechen 88a] and Mercury [Nishizaki 89] are both two-phase routers. Since the optimization criterion involves total wirelength, this leads to consideration of the SPG-As discussed in Sections 4.2.1 and 4.2.2. TimberWolfMC and Mercury generates only a single route for each net having more than 11 or 5 terminals respectively, due to the runtime requirements of the involved algorithms. Potentially this limits the overall quality obtainable, as noted in [Sechen 88]. Since a GA provides a number of distinct solutions in each run it could potentially overcome this problem. This feature of the GA is especially appealing in this context: Not only the best solution, but also the second-best, third-best, etc. are actually needed, and they are generated by the GA as a "by-product" anyway.

The basic idea of the developed GA for the SPG is to represent a solution as a bitstring of length equal to the potential number of Steiner vertices. Each bit specifies whether a specific vertex is selected for inclusion in the Steiner tree or not. By using a fast, deterministic heuristic for the SPG as decoder, any bitstring is interpreted as a valid Steiner tree.

At this time, I was not aware of any benchmarks which could be used to evaluate the algorithm. Routing graphs extracted from real placements would be the ideal type of data, but since the router had not yet been developed - its future existence depended on the performance of the SPG-algorithm - the interface to Mosaico had not been investigated, so there was no easy way of obtaining real routing graphs. Consequently, random graphs were generated and used instead, and performance was compared to that of two deterministic heuristics from the literature, implemented for this purpose. The results were published in [Esbensen 94b] and were very encouraging, not only with respect to solution quality but also with respect to computation time. The latter is unusual for a GA-based approach. Consequently, I had to pursue this topic a little further, although the direct relevance for the routing application would probably be limited. In the meantime I became aware of the OR-Library [J. E. Beasley 90], a database containing challenging bench-

search process. Like any other GA, the placement algorithm initially obtains significant improvements rapidly. Then the process slows down and the algorithm spends the vast majority of its time in a phase where only minor improvements are obtained. SA-based algorithms converges much slower than a GA initially, but later on the SA may do better. Hence, the idea was to combine the GA with SA, aiming at combining the initial convergence of the GA with the convergence of SA in the later phase of the process. Previous work along this line has been done by Rosenkiewicz and Ebeling [Rosenkiewicz 91], which was the main source of inspiration. However, the approach presented here is more general. It unifies the GA and SA into one algorithm, called SACA. Both the GA and SA are special cases of SACA which are obtained by appropriate settings of the control parameters. The interesting part is of course executions in mixed GA/SA mode. Here the algorithm starts out as a pure GA. As the performance of the algorithm decreases, SACA gradually and adaptively switches towards SA. Mutations are not carried out immediately, but accepted with a certain temperature dependent probability as in SA. Each individual has its own temperature. When no improvement has been seen for a certain number of generations, a step towards SA is taken by decreasing the population size and increasing the number of attempted mutations. Ultimately, the population size may become one, in which case the process is pure SA. The problem representation, genetic operators, etc. are unchanged, i.e., they are as described in Appendix A.

Since SACA is capable of producing the same layout quality as the pure GA, but in shorter time, it meets the original objective of speeding up the search process. However, SACA can also improve layout quality further if executed for about the same time as the pure GA. Since layout quality is considered more important than runtime, cf. Section 1.2, the latter property is the one emphasized when presenting the algorithm.

SACA was published as [Kleinhans 94a]. An extended version of this paper is presented in Appendix B. For example, the extended paper illustrates in detail what happens during a mixed-mode execution.

ACAfor the SHG

As discussed in Section 2.3, following placement/floorplanning, global routing is the layout synthesis step which should be expected to influence overall layout quality the most. When examining the layouts generated using SACA and Mosaico, one notices that while the obtained areas are

where λ is the spacing of the routing grid, l_s is the length of side s, α and β are user-defined parameters and the rounding function returns the nearest integer of its argument. d_s is the exact channel density computed by considering all appropriate terminals within the square region having side length l_s next to side s of the block, as illustrated in Fig. A.3, page 121. d_s is computed without considering global routing, i.e., this term measures local congestion only. The two other terms, $\alpha \sqrt{l_s / \lambda}$ and β, are meant to account for global routing. The first of these terms grows with the length of the routing region, similar to what is done in [Upton 90a], cf. Section 4.1.1. Note that since $d_s = 0$ implies $D_s = 0$ blocks can be abutted. Total wirelength is estimated as in [Herrigel 89]. Let M denote the number of nets, m_k the number of terminals of net k and $t_{ki} = (x_{ki}, y_{ki})$ the coordinates of the i'th terminal of net k. The *center of gravity* T_k of net k is then defined by

$$T_k = \frac{1}{m_k} \sum_{i=1}^{m_k} t_{ki}$$

and the total wirelength estimated by

$$\sum_{k=1}^{M} \sum_{i=1}^{m_k} \| t_{ki} - T_k \|$$

where $\| \|$ is the usual Euclidean vector norm. From the estimated total area and the estimated wirelength, fitness is computed in such a way that smaller area always means higher fitness.

Initially, this algorithm was interfaced with the layout system Magic [Scott 85] and published as [Eschermann 92]. Later it was improved in a number of ways: The crossover operator as well as the mutation operators were improved, an inversion operator was added, runtime was improved and the algorithm was interfaced to Mosaico/Octtools, which offers more and better tools than Magic. Appendix A describes the resulting version of the algorithm, which performs significantly better than the original version described in [Eschermann 92].

A Unification of the GA and SA Applied to Macro-Cell Placement

While the layout quality obtained by the algorithm described above was very promising, the computation time required was still very large. Consequently, it was natural to look at ways of improving the efficiency of the

relevant, despite the fact that it is a frequently used criterion in the literature. Although short total wirelength will often result in small layout area and a short delay (the length of the longest path through the circuit), this will of course not always be the case. For high-performance circuits, explicit minimization of delay would be more adequate, but would require changes of the algorithms as will be discussed in Section 6.1. The first assumption prevents routing on top of blocks. This will rarely be realistic unless the design is small and area is not considered an issue. Therefore, the first assumption is the one that compromises the practical applicability of the algorithms the most. However, incorporating over-the-cell routing in the algorithms would require significant alterations.

5.1.2 What and Why - the Development History

A GA for Macro-Cell Placement

The first algorithm developed was a GA for macro-cell placement. It is inspired by a GA for the two-dimensional bin-packing problem [Kröger 91]. Bin-packing is the problem of placing a number of given rectangles in a rectangular area of fixed width and infinite height so that no rectangles overlap and so that the height of the packing is minimized. Given a target width of a layout, the macro-cell placement problem can be seen as the bin-packing problem generalized in two ways: Firstly, the placement of each rectangle (macro-cell) is directed by a function defining the minimum distance to previously placed rectangles (the routing area estimate). Secondly, each rectangle can be oriented in eight distinct ways instead of two. A traditional bin-packing algorithm places one rectangle at a time, as far down and then as far left as possible, and the problem then is to find a suitable order in which to consider the rectangles. This idea is adopted in the GA. The (main part of) the genotype is a binary tree specifying the relative positions of the blocks, and the decoder interprets the genotype by traversing the tree and placing each block as far down and then as far left as possible without violating the routing area estimate, which is computed as each block is placed. A placement generated this way is called a *BL-placement* (bottom-left). When placing a block, the distance D_s needed from side s of the block to previously placed blocks is estimated as

$$D_s = \begin{cases} \lambda\,[d_s + \mathrm{round}(\alpha \sqrt{\frac{l_s}{\lambda}} + \beta)] & \text{if } d_s > 0 \\ 0 & \text{if } d_s = 0 \end{cases}$$

2. Two layers of metal are available for routing. One is primarily used for horizontal wire segments while the other is primarily used for vertical segments.

3. All nets are treated as *signal nets*, i.e., for example power and clock nets are given no special consideration.

4. The criteria optimized are total layout area as the most important criterion and total wirelength as a secondary criterion.

The most significant advantage of this set of assumptions has to do with the evaluation of the algorithms. For the reasons discussed in Section 1.2, the aim is to evaluate performance of the algorithms by comparing it to the performance of state-of-the-art tools using benchmark data. The most widely used set of benchmarks is distributed by the MCNC Center for Microelectronics, North Carolina [Kozminski 91]. To be meaningful, comparisons have to be done in terms of completed layouts, cf. Section 2.3, and therefore the developed tools have been integrated with a complete macro-cell layout system called Mosaico, which is part of the Octtools CAD framework [Octtools 93]. Since the above assumptions are compatible with the MCNC benchmark specifications as well as the assumptions of the Mosaico toolset (routers, compactor, etc.) they provide a feasible basis for the kind of comparisons desired.

The practical relevance of the assumptions is another issue. Power nets have to be wider than signal nets, and hence are often routed by dedicated algorithms, which also takes care of sizing of the wires. Similarly, clock nets often requires special treatment to avoid various timing problems. Hence, in technologies offering two layers of metal for routing, the second assumption is realistic, while the third is not. However, the newest technologies available today provides three or four layers of metal. In such technologies, it is common to reserve one or two metal layers for the routing of power and clock nets, and perhaps other critical nets. The remaining two layers are then used for routing of signal nets only, in which case the second and third assumptions becomes adequate. In any case, since the second and third assumptions are used only in the estimates of routing area and wirelength, and therefore concerns isolated parts of the algorithms only, it would not be too difficult to adapt the algorithms to other versions of these assumptions.

While minimization of total layout area is highly relevant, cf. the fourth assumption, the minimization of total wirelength is only indirectly

Chapter 5

Summary and Evaluation of Developed Algorithms

The appendices presents two approaches for macro-cell placement (Appendices A and B), an algorithm for the Steiner Problem in a Graph (Appendix C) and a global router for macro-cell layouts (Appendix D). This work is summarized in Section 5.1 and evaluated in Sections 5.2, 5.3 and 5.4, respectively. Finally, Section 5.5 provides an overall evaluation and presents some conjectures.

5.1 Summary

All algorithms relies on the same set of basic assumptions about the problems solved, and these are discussed in Section 5.1.1. The key ideas of the four papers reproduced in the appendices are briefly presented in Section 5.1.2. This presentation accounts for the mutual relationship of the papers as well as the relationship to other papers by pointing out major similarities and differences. Furthermore, considerations that lead from one piece of work to the next are included. Section 5.1.3 summarizes key design decisions taken for the CAs, and relates these to the discussion of practical CA issues of Section 3.4.

5.1.1 Basic Assumptions

The algorithms presented in this thesis all conform to four basic assumptions:

1. The layout area occupied by blocks and the area occupied by routing are disjoint. Consequently, all terminals of blocks are positioned along the block edges.

of tracks used, the total net length and the total number of vias used. Impressive results are reported on a number of benchmarks. In all test cases the result quality obtained is as good or better than the best result obtained by any other algorithm.

adding those of the left out constraints which are violated and then re-solving the LP. When this process is terminated, the LP-solution satisfies all constraints. Furthermore, at each tree node more reduction tests are carried out and the upper bound is possibly improved, again using the heuristic by [Hayward-Smith 86].

4.3 GAs for Related Problems

References to other GA-based approaches for macro- and standard-cell placement, partitioning and channel routing are given below. The listing is not meant to be exhaustive, but is limited to the most significant GA-based approaches known to the author.

Cohoon et al. presents a distributed GA for floorplanning of building block layouts, which minimizes area and total wirelength [Cohoon 91a]. Only slicing structures are considered, which are represented by inverse Polish expressions. The algorithm is reported to compare favourably with a simulated annealing approach. Unfortunately the exclusion of benchmark examples prevents comparison of this approach to the algorithms developed in this thesis. Glasmacher et al. has developed a GA which improves a given macro-cell placement [Glasmacher 91]. The algorithm adjusts the placement to minimize channel densities such that the total layout area is reduced.

The first GA for standard-cell placement, and perhaps the first GA for VLSI layout synthesis, was *Génie*, developed by Cohoon and Paris at University of Virginia [Cohoon 86]. Later, Shahookar et al. presented another GA for this problem called *GASP* [Shahookar 90a, Shahookar 90b], which at the time of publication outperformed TimberWolf's SA-based approach called *TimberWolfSC*. In [Mohan 93] a parallel implementation of GASP is presented.

A GA-based partitioning algorithm is presented in [Shahookar 94a, Shahookar 95]. It can do bi-partitioning, i.e., recursively partition the layout in two parts, as well as multiway partitioning. A breadth-first search of the given netlist determines the relative positioning of blocks in the genotype, and consequently an inversion operator is not used. Result quality is superior to a classical partitioning algorithm of Fiduccia and Mattheyses [Fiduccia 82].

Finally, a GA for channel routing by Lienig et al. should be mentioned [Lienig 94a, Lienig 94b]. The algorithm minimizes the number

the fact that the original solution was already minimal. Consequently, given a MSPT for $\tilde{G}$ satisfying Equation 4.1, the MST for the SPG is simply $\tilde{G}_1$, and the cost of the MSPT equals the cost of the MST. Let $P_i \subseteq \tilde{E}$ be the set of edges which connects to vertex i. The problem of finding an MSPT for $\tilde{G}$ is then formulated in [J. E. Beasley 89] as follows. The variables are

$$x_{ij} = \begin{cases} 1 & \text{if } (i, j) \in \tilde{E} \text{ is in the solution} \\ 0 & \text{otherwise} \end{cases}$$

and the objective is to minimize

$$\sum_{(i,j) \in \tilde{E}} c_{ij} x_{ij}$$

subject to the constraints

$$\sum_{(p,q) \in \tilde{E}} x_{pq} = |\tilde{V}| - 1 \tag{4.2}$$

$$\forall T \subseteq \tilde{V} : \sum_{p,q \in T,\ (p,q) \in \tilde{E}} x_{pq} \leq |T| - 1 \tag{4.3}$$

$$\forall i \in V - W, \quad \forall (p, q) \in P_i : x_{0i} + x_{pq} \leq 1 \tag{4.4}$$

$$\forall (i, j) \in \tilde{E} : x_{ij} \in \{0, 1\} \tag{4.5}$$

Equations 4.2 and 4.3 assure that the solution is a spanning tree, and Equation 4.4 is equivalent to Equation 4.1. More constraints are added in [Lucena 92].

Before starting the branch-and-cut algorithm, an attempt to reduce the size of the problem is performed by applying various *reduction tests* to the given graph. By examining local properties of the graph these tests may be able to determine that certain edges can never be part of a MST or will be part of any MST. The tests used in [J. E. Beasley 89, Lucena 92] are also used by the algorithm presented in Appendix C, and hence will be discussed in Section C.3.2, page 172.

The search scheme is depth-first traversal of a binary tree. An initial good solution is obtained by the heuristic in [Rayward-Smith 83]. At each tree node extensive computations are carried out so that only very few nodes have to be visited. A lower bound is computed by solving a linear program (LP) obtained by relaxing Equation 4.5 and initially ignoring most of the constraints. The bound is then iteratively strengthened by

at Imperial College, London, UK [J. E. Beasley 89] and later improved by Lucena and Beasley [Lucena 92].

The basic idea of this branch-and-cut approach is to transform the SPG into an equivalent Minimum Spanning Tree (MST) problem subject to an additional constraint. Assume $V = \{1, 2, \ldots, n\}$, $\forall (i, j) \in E$: $i < j$ and without loss of generality, assume that $1 \in W$. Furthermore, let c_{ij} denote the cost of edge (i, j). Then an extended graph $\tilde{G} = (\tilde{V}, \tilde{E})$ is constructed as illustrated in Fig. 4.2(b) by adding a special vertex 0 and connecting it to all vertices in $V - W$ and to the vertex 1 using edges of zero cost. Specifically, $\tilde{V} = V \cup \{0\}$, $\tilde{E} = E \cup \{(0, i) \mid i \in V - W + \{1\}\}$, and $\forall i \in V - W + \{1\}: c_{0i} = 0$. Any MST for $\tilde{G}$ which satisfies the additional constraint

$$\forall i \in V - W: (0, i) \in \tilde{E} \Rightarrow \deg(i) = 1 \qquad (4.1)$$

will have the form illustrated in Fig. 4.3.

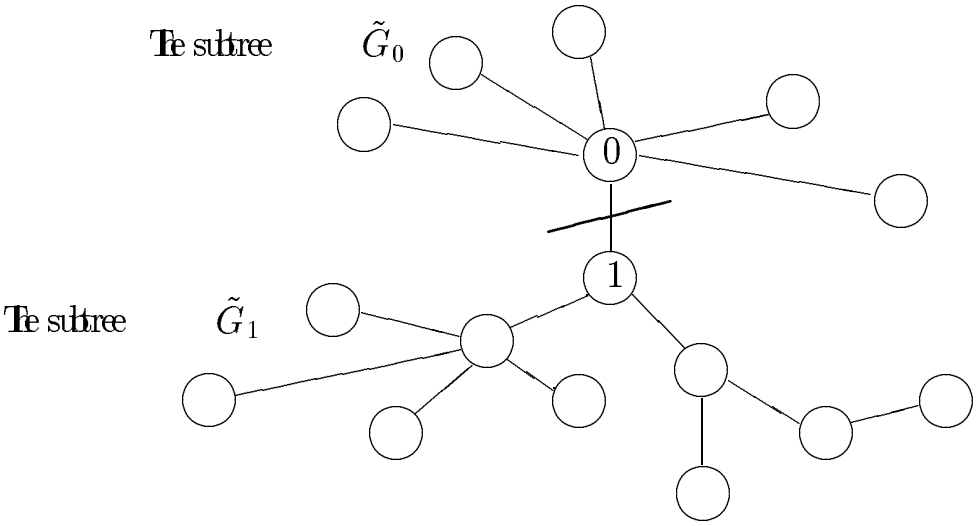

Figure 4.3: *The structure of any MSpT for* $\tilde{G}$.

If the edge $(0, 1)$ of the MST was removed, two subtrees $\tilde{G}_0$ and $\tilde{G}_1$ would emerge, as indicated on the figure. $\tilde{G}_0$ consists of a subset of the vertices $V - W$, each of which are connected by a zero cost edge to vertex 0. The vertices of $\tilde{G}_1$ includes W, and in fact $\tilde{G}_1$ is a MST for the original SPG. Assume this is not the case. Then $\tilde{G}_1$ could be replaced by the true MST for the original SPG, and any left out vertices could be added to $\tilde{G}_0$, that is, they could be connected to vertex 0 using an edge of cost 0. The resulting tree would be a feasible MST for $\tilde{G}$ satisfying Equation 4.1. But it would also be of lower cost than the original solution, contradicting

4.2.3 The Steiner Problem in a Graph / A Branch-and-Cut Algorithm

The Steiner Problem in a Graph (SPG) is the following: Given a connected, undirected graph $G = (V, E)$, an edge cost function $c : E \mapsto \Re$, $c \geq 0$, and a subset $W \subseteq V$, find a connected subgraph $G' = (V', E')$ of G such that $W \subseteq V'$ and such that $\sum_{e \in E'} c(e)$ is minimal. A solution G' is called a *Minimal Steiner Tree (MSt T)* for W in G. For $2 < |W| < |V|$ the SPG is NP-complete [Karp 72]. An example problem instance is shown in Fig. 4.2 (a).

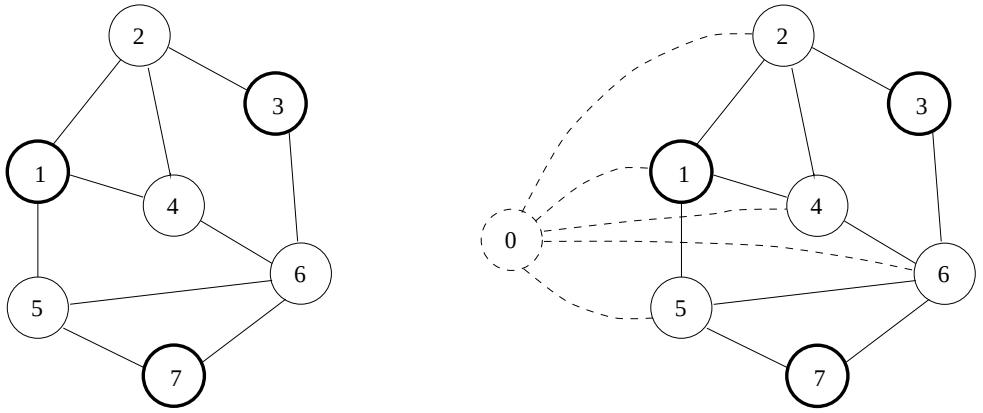

Figure 4.2: *(a) An instance of the SPG with $W = \{1, 3, 7\}$ and (b) its transformation to a constrained minimum spanning tree problem. The added vertex and edges are dashed.*

The SPG is surveyed in [Winter 87], which includes presentations of several deterministic heuristics for the problem. One of the most popular is an algorithm by Rayward-Smith and Clare [Rayward-Smith 86]. More recently other deterministic heuristics of superior performance have been reported [Winter 92]. One of these is the *Iterated Shortest Path Heuristic* presented in Section C.4.2, page 182. To the best of my knowledge, the only GA for the SPG published prior to the algorithm presented in this thesis is a recent algorithm by Kapsalis, Rayward-Smith and Smith [Kapsalis 93], which is discussed in Section C.4.5, page 190.

In [Chopra 92, J. E. Beasley 89, Lucena 92] state-of-the-art approaches to the SPG based on branch-and-cut are presented. These algorithms solve problems of up to $|V| = 2,500$ vertices to optimality. The remaining of this Section summarizes the algorithm initially developed by Beasley

capacity violations is decreased or the amount of violation is unchanged but the total wirelength is decreased.

4.2.2 An Integer Programming Approach

Mercury is a global router developed by Nishizaki, Igusa and Sangiovanni-Vincentelli at University of California, Berkeley [Nishizaki 89]. Mercury minimizes the length of specified critical nets as well as the layout area. The shortest route generated in phase one is always used for a critical net, while for non-critical nets, longer routes may be selected in phase two in order to minimize layout area. As opposed to the non-restricted positioning of terminal vertices used by TimberWolfMC in Mercury terminal vertices are either existing vertices of the routing graph or they are added at the center of edges only. Consequently, some nets may correspond to the same set of vertices, which then needs to be considered only once. The obvious drawback is a less accurate net length estimate.

The phase one algorithms are the same as those of TimberWolfMC although the search performed by Seden's algorithm for multi-terminal nets has been pruned, thereby improving the time complexity of that algorithm to $O(M^{k/(k-1)} n^3)$. By default, $M = 5$ is used when $k \leq 5$, and $M = 1$ otherwise.

The phase two algorithm minimizes layout area, which is estimated using polar graphs as described in Section 3.3.1, page 210. Two factors contributes to the estimate of the width of a channel. A *fixed* contribution accounts for critical nets and *trivial* nets, i.e., nets which are routed solely within a single channel. Since only a single, fixed route is considered for such nets, their contribution to the width of each channel can be precomputed. Exact terminal locations are used for this computation. The other factor is variable and accounts for those nets for which alternative routes are selected by the phase two algorithm. These nets are assumed to contribute to the width of a channel by an amount which is proportional to the number of nets entering the channel.

The route selection in phase two is formulated as an integer linear program (ILP), which furthermore incorporates adjustment of the placement of blocks. The adjustment is limited by the fact that the routing graph topology has to be preserved in order for the area estimate to be meaningful, as explained in Section 3.3.1. While the constraints formulated do not guarantee the preservation of the routing graph topology, the approach is reported to be "good enough for practical application".

phase method. To ease comparison, the two state-of-the-art global routers selected for presentation in Sections 4.2.1 and 4.2.2, are also both graph based, two-phase routers, which optimizes similar criteria.

The problem repeatedly solved in the first phase of a two-phase router when routing an individual net is known as the *Steiner Problem in a Graph (SPG)*, assuring that the criterion minimized is net length. This thesis also presents a new algorithm for the SPG, and to facilitate comparisons, Section 4.2.3 summarizes a state-of-the-art approach to this subproblem of global routing.

4.2.1 TimberWolfMC

The TimberWolfMC layout system was introduced in Section 4.1.5. Chapter 8 of [Sechen 88a] and [Sechen 88b] describes the global router of TimberWolfMC. It minimizes the total wire length subject to the capacity constraints of the routing regions. *Electrically equivalent pins* can be handled, that is, if two or more terminals of a block are connected within the block, only one of the terminals will be externally connected by the router. Before alternative routes for a specific net is computed, terminal vertices are added to the routing graph at positions accurately reflecting the position of the terminals. All terminal vertices are removed again before the next net is considered.

In phase one, the M shortest routes for each two-terminal net are computed by an exact algorithm due to Lawler [Lawler 76]. This algorithm requires time $O(Mn^3)$, where n is the number of vertices of the routing graph. For nets with more than two terminals, Sechen has developed a heuristic generalization of Lawler's algorithm, which attempts to find the M shortest routes. Since this algorithm requires time $O(M^{k+2} n^3)$ for a net with k terminals,[1] M has to be reduced for nets with many terminals. For two-terminal nets $M=20$ is typically used, while $M=1$ for nets with 12 or more terminals.

In phase two, a random interchange algorithm is used. The initial state consists of the shortest route for each net. If no constraints are violated, the router terminates. Otherwise, a new state is generated by randomly selecting a net passing through a channel, the capacity of which has been exceeded. A new route for the net is chosen at random among the alternatives which do not increase the total amount of capacity violations. The new state is accepted if and only if the total amount of

[1] More precisely, k is the number of terminals of the net, which are not electrically equivalent.

4.2 Global Routing

Most global routers for macro-cell layouts perform routing in terms of a rectilinear *global routing graph* or *channel graph*, which is extracted from the given placement. The edges of the graph correspond to future routing regions while the vertices correspond to intersections of routing regions. The routing graph of a simple placement is shown in Fig. D.1, page 207. To compute a global route for a specific net, vertices representing the terminals are added at appropriate locations, as illustrated in Fig. D.2, page 208. Finding a global route now becomes equivalent of finding a subtree in the routing graph which spans the terminal vertices. Every edge is assigned one or more cost values typically representing the length of the associated routing region and/or the capacity of the region, i.e., the number of nets which can pass through the region.

Typical objectives of global routing are to minimize total interconnect length, the length of certain critical nets, and/or layout area, subject to channel capacity constraints. Given a routing graph, various methods for performing the routing exist. The so-called *sequential* routers construct a complete global routing by considering one net at a time. As each net is routed, current channel congestions etc. are dynamically updated. Alternatively, all nets are routed without considering any constraints, and then the nets causing constraint violations are ripped up and rerouted. Another common method is employed by the *two-phase routers*, which generates a routing solution in two distinct phases. In the first phase, several alternative routes are computed for each net. The nets are treated independently one at a time, and no constraints are considered. In the second phase a specific route is selected for each net, attempting to minimize the objective function subject to channel capacity constraints or any other constraints.

A main drawback of sequential routers is that the result quality is highly dependent of the order in which nets are routed, and in general it is difficult to device a good net ordering a priori. This problem is avoided by the two-phase router, which in this respect treats all nets equally. On the other hand, sequential routers are faster than two-phase routers, and hence are generally preferred for very large problems. There are other types of global routers, graph based as well as not graph based. Surveys on global routing can be found in [Sherwani 93, Venkateswaran 94].

The algorithm for global routing presented in this thesis is based on a routing graph and optimizes area and total wirelength using the two-

side, three factors are considered: 1) the *average net traffic* through the corresponding channel, 2) the *position of the channel* in the layout and 3) the *relative pin density* along the block side. The average net traffic term is meant to account for global routing passing through the channel and is proportional to the TEL divided by the estimated channel length. Since most nets will be implemented by very short routes, the closer a channel is to the center of the layout, the more congested it is likely to be. The second factor accounts for this phenomenon by assuring that blocks placed at the center of the layout are allocated more surrounding space for routing than blocks placed at the periphery of the layout. Finally, the relative pin density is meant to account for local congestion, and is defined as the number of pins along the block side in question divided by the length of the side. From these three contributions, an amount of expansion is calculated. Chapter 6 of [Sechen88a] is devoted to a description of the routing area estimate.

The placement algorithm is based on simulated annealing, and the cost function minimized consists of three terms. The first term C_{teic} is the TEIC, computed as a weighted sum of the half-perimeters of all nets. The second term $C_{overlap} = w \sum_{i<j} O(i, j)$ penalizes block overlap. $O(i, j)$ is the area by which blocks i and j overlap, and w is a normalization constant defined so that $wC_{overlap} = \xi C_{teic}$ holds at the initial temperature T_0. A value of 0.5 were successfully used for ξ. The final term of the cost function has to do with the positioning of pins of flexible blocks.

Two types of moves exist. A block can be moved to another position or a pair of blocks can exchange positions. Block orientation(s) may be altered as part of a move. To optimize the performance of the algorithm, a *range limiter function* defines an upper limit on the distance by which displacement of a block is attempted in a single move. The range limiter function decreases with temperature, so that initially a block can be moved any distance while at low temperatures, only short distance moves are generated. Another mechanism similarly assures that short distance moves are not generated at high temperatures, since they are likely to be insignificant at that time.

The temperature T_{k+1} at time $k+1$ is computed as $T_{k+1} = \alpha(T_k)T_k$, where $\alpha(T_k)$ is a simple function of the given problem which at all times satisfies $0.8 \leq \alpha(T_k) \leq 0.92$. Temperature decreases the fastest towards the end of the process. For an n block problem, $400n$ moves are attempted at each temperature.

tivity between clusters i and j. The needed space s_{kl} between adjacent blocks k and l is then estimated by

$$s_{kl} = \lambda \, t_{kl} \sum_{i,j} p_{ij}^{kl} c_{ij}$$

where λ is the routing grid spacing, t_{kl} is a parameter accounting for distinct nets sharing the same track in a channel, and p_{ij}^{kl} is the probability that a connection between blocks i and j passes through the region between blocks k and l. The probability p_{ij}^{kl} is determined by considering the shortest paths from block i to block j, relying on the two-terminal representation of the nets.

4.1.5 TimberWolfMC

TimberWolfMC is another famous integrated system for floorplanning/placement and global routing of building block layouts developed by Sechen at Yale University [Sechen 88a, Sechen 88b]. The system has been continuously improved and refined for a number of years and hence offers many useful facilities. The characteristic feature of TimberWolfMC is that all main algorithms are based on simulated annealing.

The problem definition used is very general. Fixed as well as flexible blocks are handled, a block can have any rectilinear shape, and the search space is not restricted to e.g. slicing structures. The Total Estimated Interconnect Cost (TEIC) is the only criterion minimized. TEIC is a weighted sum of the estimated length of all nets. If all weights are equal, TEIC equals TEIL, the Total Estimated Interconnect Length.

TimberWolfMC consist of two main phases, initial placement and placement refinement. The latter consist of repeated execution of three steps: channel definition, global routing and adjustment of the placement. Here the placement is fine-tuned according to exact channel densities, and only three iterations of the second phase is needed for TEIC as well as total estimated area to converge. The global router used in phase two will be described in Section 4.2.1, while the remaining of this Section is concerned with the phase one algorithm for initial placement.

The basic idea of the elaborate routing area estimation is to expand each block by an amount which depends on the position of the block. I.e., as opposed to the static block expansion strategies applied in [Chan 91] and [Grodera 91] described in previous sections, the scheme used here is dynamic. To determine how much to expand a block along a given

1) The *topology* of the placement, i.e., the relative positions of all blocks, as well as the orientation of each block, is determined. 2) Global routing is performed. 3) The placement and global routing is adapted to each other through a sequence of incremental alterations of the placement as well as the global routing. 4) Routing regions are defined and ordered. 5) Local routing is performed. As each region is routed, the placement is locally adjusted according to the final region width. Note the third and the fifth step which integrates the optimization of the floorplanning/placement with the global and detailed routing, respectively. During the third step dynamic updates of the placement as well as the global routing are made possible by the use of sophisticated data structures described in [Dai 87a]. An attempt is made to preserve the topology of the placement and block orientations are not altered.

The remaining of this Section focusses on the first step. Initially, the layout is hierarchically structured by recursively partitioning the blocks into a number of groups, or *clusters*, so that each cluster contains at most 5 blocks. The partitioning heuristic considers block shapes and connectivity. A placement is generated by a top-down traversal of the resulting *cluster tree*. The search space is not restricted in any way. At each level of the hierarchy, a cost function is minimized by exhaustive search of all possible topological arrangements of the involved blocks/clusters. Furthermore, block orientations are determined at the leaf level. The cost function is a weighted sum of *geometry cost* and *connection cost*. The latter term penalizes connections between non-adjacent clusters, while the first term estimates total area and measures the relationship between actual shape and *target shape* of the placement. When choosing a specific topology, target shapes for each of the involved blocks/clusters are passed one level down the hierarchy. At the root of the tree, which corresponds to the complete chip, a target shape of the layout is given by the user. Although all possible topologies are evaluated at each level of the hierarchy, not all topologies are pursued further. Topologies for which the cost significantly exceeds the minimum cost obtained at the current level, are unlikely to lead to better placements. Consequently, such topologies are pruned from the search tree.

When evaluating a specific topology, the routing area is estimated as follows. An m-terminal net is represented as $m(m-1)/2$ two-terminal connections between all pairs of terminals. At non-leaf levels of the hierarchy, all connections are measured from center to center of the involved clusters. A *connectivity matrix* is computed in which c_{ij} is the connec-

3. A and $\bar{B}$ are divided into four submatrices each, according to the cross-point:

$$A_{n\times m} = \begin{pmatrix} A^{11}_{p\times q} & A^{12}_{p\times(m-q)} \\ A^{21}_{(n-p)\times q} & A^{22}_{(n-p)\times(m-q)} \end{pmatrix}$$

$$\bar{B}_{n\times m} = \begin{pmatrix} \bar{B}^{11}_{p\times q} & \bar{B}^{12}_{p\times(m-q)} \\ \bar{B}^{21}_{(n-p)\times q} & \bar{B}^{22}_{(n-p)\times(m-q)} \end{pmatrix}$$

4. The offspring are defined as

$$C_{n\times m} = \begin{pmatrix} A^{11} & \bar{B}^{12} \\ \bar{B}^{21} & A^{22} \end{pmatrix} \qquad D_{n\times m} = \begin{pmatrix} \bar{B}^{11} & A^{21} \\ A^{12} & \bar{B}^{22} \end{pmatrix}$$

Notice that the first step is needed to assure that each feature is copied into each offspring exactly once. A step of this type is needed by any GA applying a reordering operator. In this specific case, since inversion permutes only rows and columns of the bitmap chromosome as opposed to individual entries, the sorting required in step one is done in time $O((n+m)\log(n+m))$ instead of $O(n\,m\log(n\,m))$.

Generated offspring is not immediately included in the new population as in the simple GA of Section 3.2. Instead a pool of offspring is generated, and the new population is deterministically defined as the best individuals of the offspring pool and the previous population. To obtain sufficient diversity of the parents selected for crossover, the first parent is selected with a probability proportional to its fitness while the second is chosen uniformly at random.

4.1.4 The BEAR System

At University of California, Berkeley, a famous integrated system for floorplanning, placement and routing of building block layouts called BEAR (Building-block Environment Allocation and Routing system) has been developed by Lai, Eschermann, Koh and Pedram et al. [Lai 89, Eschermann 88]. Given a set of rectangular blocks, which may be flexible and/or fixed, BEAR minimizes layout area and total wirelength while considering given target values for the height, width or aspect ratio of the layout. The characteristic feature of BEAR is that it integrates the floorplanning and the routing steps much closer than previous systems, as reflected by the layout generation procedure which consists of five steps:

4.1.3 A GA Approach

To my knowledge only two GAs for macro-cell placement have been published prior to the algorithms presented in this thesis. GAP, developed by Chan, Shahookar and Mazumder at University of Michigan [Chan 91, Shahookar 91a], is the subject of this Section, while the other approach is discussed in Section 4.3. GAP handles macro-cells of any rectilinear shape and the search space is not restricted in any way. Three criteria are minimized: Area, total wirelength and violation of given bounds on each dimension of the final layout. Routing area is estimated by initial expansion of all blocks by a certain amount, similar to what is done in [Cohoon 91]. The total wirelength is estimated by the sum of the half-perimeters of all nets.

Given n blocks, the genotype of a solution is a $n \times m$ boolean matrix referred to as a *bitmap chromosome*. Each of the n rows represents the placement of a specific block as a concatenation of binary representations of its x and y coordinates and three additional bits selecting one of the eight possible orientations of the block. Infeasible solutions are allowed and penalized by the cost measure, which is a weighted sum of four terms. The total area, i.e., the smallest rectangle enclosing all blocks, the estimated total wirelength, the total block area exceeding the given bounds, and the total overlap area of blocks. Fitness of a solution is proportional to the inverse of its cost.

Crossover, mutation and inversion are the three genetic operators used. To allow inversion, all entries of the bitmap chromosome are tagged with identifiers of the features they encode, cf. Section 3.4.1. The inversion operator is a generalization to two dimensions of the standard one-dimensional operator. It first reverses a randomly chosen, consecutive sequence of rows and then reverses a randomly chosen, consecutive sequence of columns. The mutation operator is standard pointwise mutation. All entries of the bitmap chromosome are independently inverted with a given probability. Given two genotypes A and B, the crossover operator generates two offspring C and D in four steps as follows:

1. A copy of B, denoted $\bar{B}$, is made *homologous* to A, that is, it is reordered by permuting columns and rows so that each entry of $\bar{B}$ encodes the same feature as the corresponding entry of A.

2. A cross-point $(p, q) \in \{1, 2, \ldots, n-1\} \times \{1, 2, \ldots, m-1\}$ is chosen uniformly at random.

above on a partial placement, using the smallest side lengths of any blocks which are not yet oriented. The decision tree is traversed depth-first to reduce storage requirements.

Four bounding operations are used:

1. A lower bound on cost can be computed as just described. Initially, a good solution is obtained by solving an approximation of the cost function subject to the constraints using linear programming.

2. A given set of selected inequalities may specify an infeasible solution because of a cycle of the form "A is to the right of B", "B is to the right of C", "C is to the right of A". Fortunately, this is the only type of inconsistency possible, and it is handled by pruning the tree when a violating constraint is added.

3. All shape constraints are used for bounding.

4. Upper bounds on the length of critical nets are also used for bounding, while lower bounds are of no use.

The more constraints the user specifies, the more effective the search becomes. When no additional constraints are specified, experiments have shown that at most 6 blocks can be placed by the algorithm within a reasonable amount of CPU-time. Larger problems are handled by first partitioning the blocks into clusters of at most 6 blocks each. If needed, i.e., if the layout consist of more than 36 blocks, the partitioning is hierarchical. The branch-and-bound algorithm is then repeatedly applied on each cluster of the hierarchy, in a bottom-up order. The partitioning algorithm used is relatively simple and considers connectivity only.

At first thought one might think that since this approach is based on branch-and-bound, it should always produce an optimal placement. Note that there are two reasons why that is not the case. Firstly, "optimality" of the placement of up to 6 blocks means that an optimal value of the chosen cost function is found. The relation of the cost function to the final layout, which involves factors such as the accuracy of the routing area estimate, is another issue. Secondly, the partitioning algorithm needed for layouts of more than 6 blocks inherently leads to suboptimal results.

4.1.2 A Branch-and-Bound Approach

A branch-and-bound algorithm has been developed by Onodera at University of California, Berkeley, and Taniguchi and Tamaru at Kyoto University, Japan [Onodera 91]. It places rectangular macro-cells while minimizing layout area. The search space considered is unrestricted, i.e., not limited to e.g. slicing structures. The cost function minimized is

$$(W_x + \frac{\lambda_x}{W_y} \sum_i L_{iy}) \times (W_y + \frac{\lambda_y}{W_x} \sum_i L_{ix})$$

where W_x (W_y) is the width (height) of the smallest rectangle enclosing all blocks, L_{ix} (L_{iy}) is the width (height) of the smallest rectangle enclosing all pins of net i, and λ_x (λ_y) is the routing grid spacing in the horizontal (vertical) dimension. Hence, the function optimized is the area of the smallest rectangle enclosing all blocks, which is expanded to account for routing. The expansion in e.g. the y-dimension is λ_y times $\frac{1}{W_x} \sum_i L_{ix}$, i.e., the accumulated number of horizontal wire segments spanning the width of the layout. Routing is also accounted for by initially expanding all blocks with an amount depending on the number of terminals along each of its sides.

The cost function is minimized subject to user-defined constraints on the shape of the layout and on critical nets. The shape constraints can be a target aspect ratio, bounds on aspect ratio, or upper bounds on one or both dimensions of the layout. Constraints on critical nets can be upper and/or lower bounds on the length of these nets.

A placement is described by specifying an orientation of each block and the topological relationship between every pair of blocks. The topological relationship of blocks A and B is either "A is to the right of B", "B is to the right of A", "A is above B", or "B is above A". Each of these relations can be expressed as a linear inequality. To avoid overlapping of blocks, at least one of the inequalities should be satisfied. Each decision variable of the algorithm specifies either an orientation of a block or a topological relation between a pair of blocks. The latter is done by selecting one of the four inequalities to hold.

To optimize the branching schedule, larger blocks and/or blocks related to critical nets are considered before other blocks. Furthermore, the topological relationship of the blocks constituting a partial placement is always determined before the orientation of the blocks. A lower bound on cost can then be obtained by evaluation the cost function described

MB applies a classical estimate of the total net length known as the *half-perimeters* of the nets. The length of each net is estimated as half of the perimeter of the smallest rectangle enclosing all terminals of the net. Total net length of the layout is estimated as the sum over all nets of the half-perimeters.

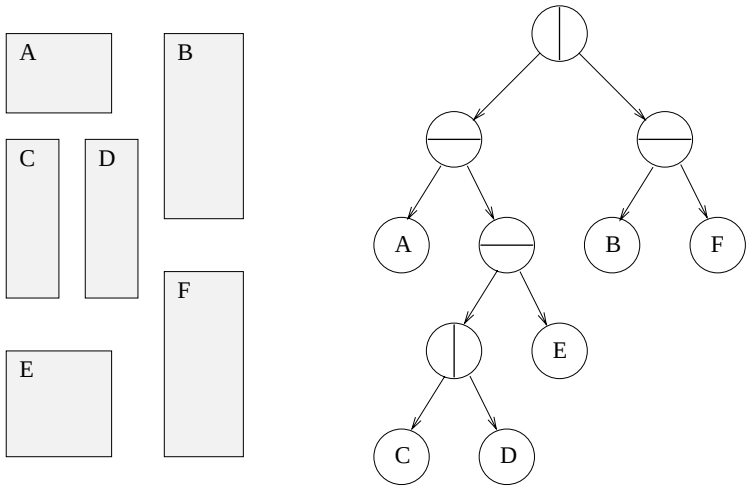

Figure 4.1: *A slicing structure and its corresponding slicing tree. A leaf corresponds to a block and an intermediate node corresponds to a channel.*

An important feature of MB is that the two cost factors, area cost and net length cost, are not combined into a single cost measure using a weighted sum, as is done in many algorithms. As pointed out in [Upton 90a, Upton 90b], a weighted sum often introduces balancing problems caused by the different nature of the involved functions. These problems are of the very same nature as the problems of penalty terms discussed in Section 3.4.3. Instead, in MB the two criteria are evaluated independently. Two temperatures are maintained, one for each criteria, using the same cooling schedule. If a move decreases both cost values, it is always accepted. When one or both cost values increases, the move is accepted if and only if it is accepted with respect to each criteria separately, considering each criteria in the usual manner and using its associated temperature.

Four types of moves exist. Two subtrees of the slicing tree can be exchanged, a subtree can be moved, and the orientation of a subtree can be altered. A subtree can consist of any number of blocks. The fourth move type is alteration of the aspect ratio of a flexible block.

4.1.1 A Simulated Annealing Approach

The Macro Block Placement program (MBP), developed by Upton, Sarii and Sugiyama at Seattle Silicon Corporation is described in [Upton 90a, Upton 90b]. MBP handles layouts which are a mixture of standard-cells and macro-cells. This capability is highly relevant for real-world designs, cf. Section 2.2. A placement is generated in three main steps:

1. The standard-cells of the layout are partitioned to form a number of flexible blocks. Criteria minimized are the size difference between the blocks and the number of nets cut by the partitioning. The technique used is simulated annealing.
2. Using the terminology introduced in Section 2.3, this step is floor-planning. An aspect ratio is to be determined for all flexible blocks just created, and all blocks are to be placed and oriented. Total area, total wirelength, and the deviation from a target aspect ratio is minimized using simulated annealing.
3. When all aspect ratios have been determined and all blocks have been placed and oriented, the standard-cells within each of the blocks created in the first step are placed. Again simulated annealing is used.

The remaining of this Section discusses the second step, which is the key step of MBP. The search is restricted to slicing structures, and the algorithm operates in terms of the *slicing tree*, a binary tree representing a slicing structure as illustrated in Fig. 4.1.

Three factors contributes to the estimated area cost: A routing area estimate, i.e., the estimated area of channels, an estimate of the empty space, i.e., the area which is neither occupied by a block nor a channel, and finally, a penalty for deviation from the target aspect ratio. The routing area and the empty space is estimated by a depth-first traversal of the slicing tree. At each intermediate node the width of the channel separating the two subtrees are estimated. The width of the channel between blocks A and B is estimated as $\lambda \sqrt{t_A + t_B} + \sqrt{\max(l_A, l_B)}$, where λ is the spacing of the routing grid, t_X is the number of terminals along the relevant side of block X and l_X is the length of that side. The first term estimates congestion in the channel from the number of terminals present, while the latter accounts for the global routing passing through the channel without being connected within the channel. The longer a channel is, the more likely it is that nets will pass through it.

Chapter 4

Related Work

This Chapter presents selected algorithms and tools which to the best of my knowledge constitutes the current state-of-the-art in macro-cell placement and global routing. To facilitate meaningful performance comparisons, a further selection criterion has been that the used problem definitions are similar to those used in the work presented in this thesis. For example, state-of-the-art tools explicitly optimizing circuit performance were excluded. The applied optimization method has not been an issue, although for the problems where a previous GA-based approach exist, it has been included to facilitate detailed comparisons to the GAs presented in this thesis.

Section 4.1 presents approaches for macro-cell placement, and Section 4.2 presents approaches for global routing, including an algorithm for the Steiner problem in a graph. References to GAs for related problems such as partitioning and channel routing are provided in Section 4.3. The performance of the approaches will be compared in Chapter 5. Familiarity with the optimization techniques of simulated annealing and branch-and-bound is assumed throughout the chapter.

4.1 Macro-Cell Placement

Five approaches to macro-cell placement are presented, including algorithms based on simulated annealing, branch-and-bound, and the GA. For a survey on (macro-cell) placement techniques in general, the reader is referred to [Shahookar 91, Sherwani 93].

be caused partly by another operator, which provided some "good" input individuals. Therefore, a scheme is needed in which credit is propagated "backwards" through a sequence of operators, which produced (the ancestors of) a specific individual.

of times for each problem, using the parameter values represented by the individual of the meta-GA. The search space explored by the meta-GA is very small compared to usual GA applications, e.g., in [Grefenstette 86] the space merely consists of 262,144 points. This is one reason why it is reasonable to assume that the parameters used for the meta-GA are not too critical. Otherwise, of course nothing would have been gained Grefenstette used the parameter values recommended in [De Jong 75] for the meta-GA. Other consequences of the extremely small search space of the meta-GA is that a very small population is sufficient and that a good result can probably be obtained in very few generations. Still, the complex interaction between the various parameters of the GA to be tuned is captured much better by the meta-GA than by simple strategies applied when searching for parameter settings manually. The obvious drawback of the meta-GA approach is the runtime requirements. In [Grefenstette 86] it is said that "...the meta-level experiments represents a sizable number of CPU-hours". On the other hand, it only needs to be done once and for all when a new GA has been developed.

The final approach to parameter setting considered in this Section is radically different. Rather than attempting to find a fixed set of parameter values, Davis devises a scheme for dynamically updating some parameters during the execution of the GA [L. Davis 89]. The parameters in question are the probabilities which defines the frequency of applying each operator. The idea is to dynamically update the application frequency of each operator according to its current performance, measured in terms of fitness-change of the individuals altered by the operator. The current best performing operators should be applied the most. Initial values for all parameters are still needed, but are less critical. Apart from partly avoiding the problem of finding good parameter values, this strategy also has the potential of improving the performance of the algorithm, since fixed parameter values throughout the run is unlikely to be ideal.

The adaption scheme itself introduces some new parameters, e.g., how often to perform dynamic updates. In [L. Davis 89] five new parameters are introduced. However, this number is fixed regardless of the number of parameters dynamically updated. Furthermore, the introduced parameters are probably less sensitive. But a more serious problem remains, which is that of credit assignment: How to update the operator probabilities fairly. It is not sufficient to consider the performance of each operator isolated, since the high performance of a specific operator may

However, the main limitation of both [De Jong 75] and [Grefenstette 86] is that the proposed values only apply to GAs based on fixed length, binary representations.

In [Goldberg 89b] Goldberg presents a theoretical approach to the determination of an "optimal" population size, where "optimal" is defined in terms of schemata processing per individual. This approach is also limited to fixed length, binary representations. The results presented suggests a population size which grows exponentially with the length of the encoding. As pointed out in [Reeves 93], such population sizes will make the GA inferior to other optimization methods on real-world problems. Instead, Reeves determines a lower bound on the usable population size, the objective being that of using the smallest possible population [Reeves 93]. A reasonable criterion is that any point in the search space should be reachable using crossover only. A necessary condition for this is that every possible gene value is present at every gene location in the initial population. Assuring that the initial population is generated uniformly at random, a lower bound on population size can then be determined. Reeves shows that the minimum population size computed this way grows dramatically with the cardinality of the alphabet used for the encoding, and presents this as an argument in favour of binary representations [Reeves 93]. However, based on very similar considerations, a different conclusion can also be reached. In [Tate 93] it is suggested that when the probability of having every possible gene value represented at every position in the initial population becomes too low, one should compensate by increasing the mutation rate rather than increasing the population size. If this can be done without compromising convergence, the runtime penalty of a large population is avoided.

The third approach listed is to consider the search for good parameter values as a meta-level optimization problem, which is solved as such by using e.g. a GA, referred to here as the meta-GA. This approach was introduced by Grefenstette, who as mentioned previously, generated a new set of generally accepted default values this way [Grefenstette 86]. In [Shahookar 90a] a meta-GA is used to find suitable values for a GA for standard-cell placement. The individual of the meta-GA is a representation of the parameter values of the GA to be tuned. Note that this approach is quite general in the sense that other options such as selection schemes, alternative operators, etc. can also be incorporated and optimized this way. A single meta-GA fitness computation consists of executing the GA on a representative set of problems, preferably a number

3.4.4 Selection of Parameter Values

Finding suitable values of the control parameters of a GA, e.g., the population size, crossover rate and mutation rate, is in general a non-trivial task, since the parameters interacts in a complicated way. Yet, from a practical point of view this problem is very important. A fixed set of parameter values is needed, which yields good results on a wide range of problem instances, since tuning the parameters towards a specific problem instance is a very tedious and time consuring task. Furthermore, it does not provide a fair basis for comparing the performance of the GA to that of other approaches.

Four main approaches for selection of parameter values can be identified in the literature, which will be commented in the following:

1. Ignore the problem and do problem specific tuning. Unfortunately, it is not hard to find papers following this approach.
2. Find a fixed set of parameters by extensive experimentation and/or by using general guidelines provided in the literature.
3. Consider the problem as an optimization task at a meta-level and approach it by applying another GA (a meta-GA).
4. Eliminate the problem by introducing an adaptive scheme for the parameter values.

Finding a fixed set of parameter values by extensive experimentation is the most common approach, and is also the one used in the work presented in this thesis. Various parameter settings are simply tried out in some systematic way on a set of test problems. Due to the stochastic nature of the algorithm, a number of runs is needed for each parameter setting on each problem, which of course makes this approach very time consuring. Furthermore, the complex interaction of the parameters are only captured to a very limited extent.

The literature does offer some guidelines for finding a fixed set of parameter values, although they are of limited applicability. As early as 1975, De Jong suggested a set of general applicable parameter values based on extensive work with a test suite of functions, which is still widely used [De Jong 75]. Later, Grefenstette suggested a set of parameter values generated by a meta-GA [Grefenstette 86], which were shown to outperform De Jong's values. Grefenstette's settings has been widely used by other researchers and are generally accepted as reasonable defaults.

remains the easiest and best method when the feasible region is large relative to the total domain, or when the problem is "smooth". Otherwise, when possible, constraint enforcement is "probably the best way to tackle constraints". This view coincides with the experience of the author of this thesis. In a Masters project the highly constrained problem of VLSI floor-planning was approached using a GA based on the penalty method. The algorithm never worked well, since it turned out to be next to impossible to find suitable values of the weights of the penalty terms. In other words, the modified cost function c^- was optimized quite well, but had only very limited relation to c. The work on highly constrained problems presented in this thesis is based on constraint enforcement, as will be discussed in Chapter 5.

Two recent approaches for constraint handling do not quite fit into the above discussion, since they rely on constraint enforcement while still being quite general. The GENOCOP system presented in [Michalewicz 91, Michalewicz 92] is a general GA based system for numerical optimization problems with any set of linear constraints. Constraint satisfaction is enforced by a scheme which relies on the convexity of the feasible regions. Consequently, it can not easily be generalized to nonlinear constraints. In [Michalewicz 91] GENOCOP is compared to a GA using the penalty method, a specialized GA using constraint enforcement (GENETIC-2) and a package for mathematical programming (GAMS), on a test suite of six functions. The best results are obtained by GENETIC-2, which is slightly better than GENOCOP. However, GENOCOP is clearly superior to GAMS, while the penalty-based GA fails to find any feasible solutions at all.

The other approach is presented in [Schoenauer 93]. The basic idea is to execute the GA several times, each time satisfying a not yet satisfied constraint. In the first execution, the cost function is simply p_1, hence, the final population is (ideally) solutions satisfying the first constraint. From that starting point, the GA is executed again, this time with p_2 as cost function. Solutions now violating the first constraint are eliminated by assigning them zero fitness. For a given problem with k constraints, this process is repeated k times, to generate a population satisfying all constraints. Then, from this starting point, the original cost function is optimized by executing the GA the $k+1$'th time. In principle this approach is generally applicable. But as noted by the authors it is computational expensive, and the success of the approach relies on diversity in the populations being carefully maintained in each execution.

should estimate the *expected completion cost*, which is the additional cost needed to transform the infeasible solution into a feasible one. A suitable estimate of completion cost is given for a three-dimensional problem, but it is noted that the technique can not be easily generalized. A scheme for dynamically updating of penalty terms is proposed in [Smith 93], and promising results are reported. However, only one problem instance is considered which has only one type of constraints, i.e. $k = 1$.

Property	Penalty method	Enforcement method
Generality of approach	High	Low, none
GA theory applicable	Yes	No
Algorithm development time	Short	Long
Characteristics of function optimized	As cost function	Worse than cost function
Size of search space	Large	Small
Some feasible solution guaranteed	No	Yes
Weight adjustment problems	Likely	No
Solution quality obtained	Problem dependent	Problem dependent
CPU-time requirement	Problem dependent	Problem dependent

Table 3.1: *Comparison of the two main methods for constraint handling.*

It is especially difficult to compare the two constraint handling strategies with respect to performance, both in terms of solution quality and computation time. The few results and opinions reported in the literature on this issue are conflicting. According to [Smith 93] the optimal solution(s) of highly constrained problems tend to lie on the boundary of the feasible region, and therefore, many neighbours of an optimum are infeasible. Consequently, in order to find a path to an optimum, it is important to allow intermediate, infeasible solutions to be considered [Richardson 89, Smith 93]. In terms of schemata, the argument is that the feasible solutions may contain a relatively low proportion of the building blocks, which should be put together to form the feasible, global optimum [Smith 93]. Consequently, it may be difficult for a GA based on constraint enforcement to find a path to a good solution, let alone a global optimum. Smith and Tate [Smith 93] further points out that due to the complexity of constraint enforcing decoders and operators, these operations can be the bottleneck of the search, which is another argument in favour of the penalty method. On the other hand, a GA based on the penalty method may spend most of its time evaluating infeasible solutions, which is avoided by the constraint enforcement method. The best choice of constraint handling method undoubtedly depends on the specific problem. According to [Schoenauer 93] the penalty method

Clearly the two methods can be combined so that some of the constraints of a given problem are handled by penalty terms while others are enforced. Each method has its advantages and disadvantages to be discussed in the following. The reader should keep in mind that since this research topic is still in its infancy other GA researchers might not agree with the views to be presented. The main points of the discussion are summarized in Table 3.1.

The penalty method is the most general, since it only modifies the cost function, while all other components of the algorithm remains unaltered. For the same reason, existing GA theory is applicable, while this is not the case when a specialized representation and/or specialized operators are used to enforce constraint satisfaction. Such representations and operators are furthermore nontrivial to design, and consequently constraint satisfaction is the most expensive approach in terms of development time.

Assuming that the cost function c has properties which makes it hard to optimize and that the penalty functions p_i are simple, e.g., linear or quadratic, c^- will not be significantly harder to optimize than c. In contrast, when enforcing constraints by some repair-method, the decoder maps many genotypes to the same point in the phenotype space, perhaps in a very "non-smooth" way. Hence, the function actually optimized, when seen as a function from the genotype space, will be harder to optimize than c. On the other hand, constraint enforcement gives a much smaller search space, especially for highly constrained problems where the feasible solutions may constitute only a disappearing fraction of the domain.

The penalty method requires the design of suitable penalty functions and corresponding weights, which is not a trivial task. If penalties are too low, no feasible solution may ever be found, while this is guaranteed by constraint enforcement. Too high penalties may turn the optimization into a search for a feasible solution only, while not being able to distinguish the quality of distinct feasible solutions. Furthermore, if the penalty functions differs in nature, e.g., p_i is linear and p_j is cubic, or if some constraints are simply much easier to satisfy than others, the relative importance of the penalty terms may change during the optimization process. To overcome this problem the weights needs dynamically adjustment. Some general guidelines for the design of penalty functions are given in [Richardson 89]. They conclude that a good penalty function should not just count the number of constraint violations. Instead it should estimate the distance from a feasible solution. That is, the penalty

4. Rather than initializing the population randomly, it can be seeded with individuals generated by heuristics, as discussed in [Schultz 90], [Grefenstette 87], among others.

GAs exploiting problem-specific knowledge is sometimes referred to in the literature as *hybrid GAs*, *knowledge-augmented GAs* or *Evolution Programs*. In [Michalewicz 93] a case study is presented in which more and more problem-specific knowledge is incorporated into a specific GA. As one would expect, the study shows that the more problem-specific knowledge is exploited, the better performance is obtained.

On the other hand, as problem-specific knowledge is exploited, the generality of the algorithm is compromised. Furthermore, any knowledge of algorithmic properties obtained by theoretical analysis as discussed in Section 3.3 will be sacrificed. And finally, the time it takes to develop the algorithm will increase significantly [Michalewicz 93].

3.4.3 Constraint Handling

Since almost all real-world problems involve nontrivial constraints, techniques for constraint handling in GAs is a very important, but almost unexplored topic. Further research is indeed needed. There are two main approaches to constraint handling in GAs:

1. The penalty method. Infeasible solutions are allowed but penalized, typically as follows: For a given problem with k constraints and cost function c, the cost function is replaced by $\bar{c}$ of the form

$$\bar{c}(s) = c(s) + \epsilon \sum_{i=1}^{k} \lambda_i p_i(s)$$

where s denotes a solution, the function $p_i \geq 0$ measures the degree of violation of the i'th constraint, $\lambda_i > 0$ is a weight determining the relative importance of violations of the i'th constraint and ϵ equals 1 for minimization problems and -1 for maximization problems.

2. Constraint satisfaction. Infeasible solutions are avoided and only feasible solutions are ever considered. Constraint satisfaction can be enforced at all times by using a representation in which only solutions satisfying (some of) the constraints can be expressed, and/or by using a decoder which "repairs" any (remaining) constraint violations, and by using genetic operators which generates only feasible solutions.

3.4.2 Exploiting Problem-Specific Knowledge

The preceding Section discussed how to improve the performance of a GA by using a problem-specific encoding. Another performance enhancing technique is to incorporate the use of problem-specific knowledge into the algorithm in various ways. One of the strongest advocates of doing so is Lawrence Davis. In [L. Davis 89] he wrote:

> "... it has seemed true to me for some time that we cannot handle most real-world problems with binary representations and an operator set consisting only of binary crossover and binary mutation. One reason for this is that nearly every real-world domain has associated domain knowledge that is of use when one is considering a transformation of a solution in the domain. It is a truism in the expert system field that domain knowledge leads to increased performance in optimization, and this truism has certainly been borne out in my experience applying genetic algorithms to industrial problems. Binary crossover and binary mutation are knowledge-blind operators. Hence, if we resist adding knowledge to our genetic algorithms, they are likely to under-perform nearly any reasonable optimization algorithm that does take account of such domain knowledge.
>
> [...] I believe that genetic algorithms are the appropriate algorithms to use in a great many real-world applications. I also believe that one should incorporate real-world knowledge in one's algorithm by adding it to one's decoder or by expanding one's operator set."

Problem-specific knowledge can be exploited in at least four ways:

1. Heuristics can be used in the decoder to interpret a genotype "sensibly" with respect to the problem.

2. The existing genetic operators can be altered so that e.g. the crossover operator combines the parent individuals using a heuristic to improve the fitness of the produced offspring. This is discussed in e.g. [Grefenstette 87, Goldberg 89a, Davidor 91].

3. New operators can be added, which performs local optimization of a given individual using any problem-specific methods available. This is investigated in e.g. [Suh 87].

presented in [Davidor 89], but as noted by the author the approach has some serious limitations. One of these is that to avoid visiting all points of the search space, epistasis has to be estimated by a sampling method, for which no confidence measure is provided.

A technique for reducing epistasis by increasing the size of the representation is presented in [D. Beasley 93]. However, this method is limited to combinatorial optimization problems, and the price paid for lowering epistasis is that of a much larger search space. There is no general applicable techniques available to facilitate the design of a low epistasis representation and consequently, this important task has to be solved ad-hoc on a case-to-case basis. This is a major reason why some talk about the "art" of designing GAs.

As for the second property mentioned above, the desirable low distance between mutually dependent genes, the situation is somewhat better. In general, it is not known a priori how the genes are related, and consequently it is generally impossible to determine a good ordering statically. Instead, the most common approach is to add a so-called *reordering operator* to the set of genetic operators. As the GA is executed, such an operator reorders given genotypes, thereby attempting to group together the mutually dependent genes dynamically [Goldberg 89a]. This requires each gene to be labelled so that the interpretation of a genotype becomes independent of the ordering of its genes. The most used reordering operator is *inversion* introduced in [Holland 75], which selects a substring of genes at random and reverses it. The drawback of using reordering operators is that the required order-independent genotype effectively represents a significant expansion of the search space. An alternative to reordering operators is presented in [Goldberg 91]. Here the optimization process is divided into two distinct phases. The sole purpose of the first phase is to find a suitable ordering while the optimization is performed in the second phase. However, it is hard to judge the feasibility of this approach, since it is only tested on rather small problems. Another approach is presented in [Bui 93], which is applicable to a certain class of graph problems only. It is assumed that the genotype is a bitstring, in which each bit selects or de-selects a certain vertex of the graph. In a preprocessing phase the ordering is defined, for example as the order of traversal of the graph by a depth-first or a breadth-first search. Extensive experimental results are reported, which show that the preprocessing significantly improves performance for certain graph types, while it has no effect on other graph types.

processed as expressed in the Building Block Hypothesis and the Schema Theorem.

If the epistasis of a representation is very low, i.e., there is little or no nonlinear interaction between the components, the problem is easy in the sense that it can (almost) be solved by optimizing along one component at a time. Consequently, the GA will be outperformed by simple hillclimbing techniques. On the other hand, if epistasis is extremely high, the GA will fail to process schemata in a useful manner, and the search becomes random. Hence, relative to competing optimization techniques, the GA will perform its best when the epistasis is neither too low nor too high, as illustrated in Fig. 3.3. The important point here is that the epistasis level is *not* fixed for a given problem but depends on the representation of the problem, which is chosen by the designer. This coincides with the remarks of Section 3.1.2 regarding the application area of GAs, since any encoding of a difficult optimization problem will have some degree of nonlinear gene interaction. Yet, the designers task is to optimize performance by devising an encoding giving the lowest possible epistasis level.

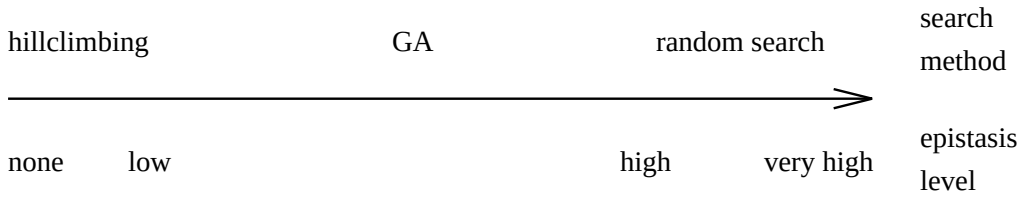

Figure 3.3: *Relative GA performance depends on the epistasis level of the encoding.*

Theoretically, for a large class of problems, a representation exists which will make the problem easy to solve for a GA. More specifically, Vose and Liepins have shown in [Vose 91b] that for any problem with an injective cost function, a representation exists which effectively transforms the given problem into a problem which is known to be easily solvable by a GA [1]. This theoretical result can not be directly applied in practice since the construction of the representation effectively requires the given problem to be solved. However, an important consequence of the result is that the pursuit of a good encoding is not in vain.

To facilitate the development of a low epistasis representation for a given problem, a general applicable measure of epistasis of a representation would be very useful. An attempt to develop such a measure is

[1] This problem is the *counting 1's problem*, i.e., the problem of counting the number of 1's in the binary representation of a given positive integer. The counting 1's problem is of course easily solved by a trivial, deterministic program but can also, if one insists, be solved by a GA.

natural way of representing a solution. And more seriously, it is often very hard if not impossible to obtain a competitive performance on a real-world problem when insisting on a binary encoding. A lot of work has been done on codings based on integers, e.g. [Bramlette 91] and floating-point values, e.g. [Janikow 91, Michalewicz 91]. But when the parameters of a problem are not numeric, as for example in combinatorial optimization, completely different representations based on e.g. graphs may be more suitable. In [Eshelman 92] a binary tree is part of a genotype.

Today the GA community is divided in those who believe that the binary representation should always be used and those who are in favour of applying any "natural" representation, whether it is binary or not. To a large extent this division in viewpoints is probably caused by different objectives of working with GAs. Generally speaking, when using a highly specialized (non-binary) representation, performance is often gained at the cost of sacrificing the general applicability as well as the theoretical foundation of the algorithm. Skeptics of GA theory might argue that there wasn't much theory to sacrifice anyway.

However, the Building Block Hypothesis and the Schema Theorem still provides useful guidance on how to design a good (non-binary) representation for a real-world problem. The schema theory suggests that a good encoding is one which has two properties, which will be explained below:

1. The dependence between the components of the representation is small.

2. The distance between mutually dependent components is small.

The components of a representation, or genotype, for example the individual bits of a bitstring, is commonly referred to as *genes* and, in analogy with biology the degree of nonlinear dependence between components is denoted *epistasis* [D. Beasley 93a]. There is no generally accepted and exact definition of the term epistasis. But the idea is that low epistasis refers to a low degree of nonlinear gene interdependence, that is, the fitness of the individual is close to being a linear combination of the gene values. Similarly, high epistasis means that fitness is a highly nonlinear function of the gene values. The term "mutually dependent components" refers to components, which by their interaction effects fitness significantly. In other words, the necessary criteria of a good encoding is that mutually dependent genes are close together and that epistasis is as low as possible. If a genotype satisfies these criteria, it allows schemata to be usefully

is assumed to be infinite. And in [Goldberg 87] and [Horn 93] only two possible individuals, "0" and "1", exists. Consequently, the insight gained so far from Markov chain analysis is limited, but hopefully future research will bring significant progress.

3.4 Practical Issues of Genetic Algorithms

When designing a GA for a specific application a lot of practical issues needs to be addressed. For example, to avoid quickly getting trapped in a (poor) local minimum while also avoiding very slow convergence, the variance of the fitness values needs to be controlled throughout a GA execution. For that purpose, the fitness measure is most often defined as a non-trivial transformation of the given cost function. For a discussion of this and other problems for which effective standard solutions can be found in the literature, the reader is referred to [Goldberg 89a].

This Section focusses on four specific problems that are especially important for the algorithms presented in this thesis but for which no easy or generally accepted solutions exists. On the contrary, these problems are open research questions.

The key point in designing any GA is the design of a suitable genotype, which is the topic of Section 3.4.1. Exploitation of problem-specific knowledge is discussed in Section 3.4.2. The topic of Section 3.4.3 is techniques for constraint handling, since non-trivial constraints are almost always introduced by real-world problems. Finally, Section 3.4.4 presents strategies for finding suitable values of the control parameters, i.e., the population size, the crossover rate, the mutation rate, etc.

3.4.1 What is a good encoding?

Traditionally, GA research has focussed on algorithms based on binary representations for many reasons [D. Beasley 93b]. The binary representation matches the view of the GA as a robust, general-purpose approach to optimization. Holland's original work [Holland 75] focussed mainly on binary representations and the main body of GA theory assumes a binary representation. As mentioned in Section 3.3 the binary representation is generally believed to be preferable from a theoretical point of view, since it maximizes the number of schemata, although this has been questioned.

However, for most real-world problems, a binary encoding is not a

fully processed in the sense that the number of representatives increases or decreases exponentially. Holland showed that the number of schemata usefully processed in this sense is in the order of $O(n^3)$. This phenomenon is known as *implicit parallelism* and is often referred to as the reason for the good performance obtainable by GAs.

To fully exploit the effect of implicit parallelism, it should be advantageous to have as many schemata as possible. Consider an alphabet of cardinality k and a string length of l. The size of the search space spanned is then k^l and the number of possible schemata is $(k+1)^l$. For a fixed search space size it can then be seen that the maximum number of schemata is obtained by minimizing the cardinality of the alphabet. This is one of the main reasons that the binary representation has been dominant in the GA literature and is still preferred by many researchers. However, [Antonisse 89] interprets schemata differently and concludes that the binary alphabet does *not* maximize the number of schemata. On the contrary, and in correspondence with intuition, alphabets of higher cardinality has more expressive power and represents more schemata. Goldberg has later presented arguments why high-cardinality alphabets may perform well, thus attempting to account for the different viewpoints [Goldberg 90]. Another critical view of the traditional schema theory is presented in [Grefenstette 89]. Here it is pointed out that Holland's $O(n^3)$ estimate of implicit parallelism assumes independence of the individuals and hence only holds in the first few generations. It is also noted that the Schema Theorem is formulated in terms of the fitness function rather than a given cost function. This is problematic since the fitness function is part of the GA itself and as mentioned previously it is typically defined as a non-trivial function of the cost function.

Since the Schema Theorem does not guarantee that representatives of a specific (above average) schema will ever emerge, it does not directly provide insight into the global behaviour of the GA in terms of its overall convergence properties. The global behavior of the simulated annealing algorithm has been analyzed successfully using Markov chains, and it is therefore an obvious idea to investigate the use of Markov chains for a similar analysis of GAs. Some first attempts in this direction is presented in e.g. [Goldberg 87, T. E. Davis 91, Nix 92, Suzuki 93, Horn 93]. However, since the entire population constitutes the state of the process, the number of possible states is enormous, which greatly complicates such analysis unless extreme simplifying assumptions are made. For example, in [Nix 92] population size and/or the length of the genotype string

the population at generation t, let $m(H, t)$ be the number of individuals in the population at generation t which represents H, and let $\bar{f}(H)$ be the average fitness of all individuals representing H at generation t. Then the expected number of individuals representing H in the next generation, i.e., $E(m(H, t+1))$, can be estimated as

$$E(m(H, t+1)) \geq m(H, t)\frac{\bar{f}(H)}{\bar{f}}\left[1 - p_c\frac{\delta(H)}{l-1} - o(H)p_m\right]$$

This important result is called the *Schema Theorem*. A derivation of the theorem can be found in [Goldberg 89a]. If $o(H)$ and $\delta(H)$ are small, and $\bar{f}(H) > \bar{f}$ then $m(H, t+1)$ is expected to be $m(H, t)$ multiplied by a factor greater than one. In other words, the Schema Theorem states that in the SGA the expected number of individuals representing a schema with short defining length, low order and above average fitness, will increase exponentially.

What does the Schema Theorem have to do with GA performance? The answer depends on the validity of *The Building Block Hypothesis* [Goldberg 89a], which states that by combining good part-solutions, called *building blocks*, good complete solutions emerge. By further *assuming* that good building blocks corresponds to schemata with short defining length, low order and above average fitness, the Schema Theorem tells us that good building blocks are usefully processed in the sense that their number of representatives increases exponentially, and hence we have an explanation why the SGA works.

The Schema Theorem has been generalized in various ways, notably by Vose. In [Vose 91a] he generalizes the concept of a schema to that of a *predicate*, which is defined as *any* set of genotypes. A Schema Theorem in terms of predicates is then developed, although the effect of mutation is ignored. Since the concept of a predicate is representation independent, so is the resulting version of the Schema Theorem. As pointed out in [Vose 91b] it is also independent of the specific genetic operators. The two last terms of the theorem measuring the probabilities of disruption by crossover and mutation, respectively, can be replaced by functions measuring the disruption caused by any other set of operators used.

Let us return to the original Schema Theorem to discuss another important phenomenon. Since a binary string of length l represents 2^l distinct schemata, somewhere between 2^l and $n2^l$ schemata will be represented in a population of size n. However, crossover destroys schemata of relatively high defining length, hence not all schemata will be use-

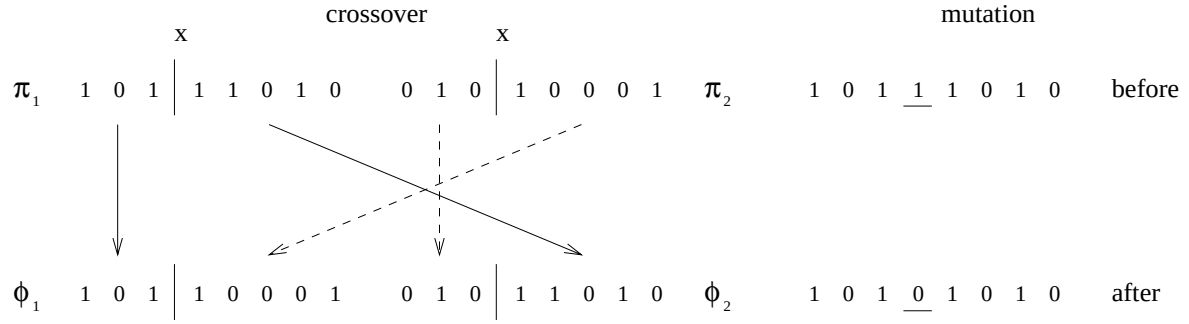

Figure 3.2 *One-point crossover and pointwise mutation. The vertical lines marked 'x' indicates the randomly chosen crosspoint.*

3.3 Theory of Genetic Algorithms

There is no generally accepted and "complete" theory which fully explains the properties of the GA. However, hypothesis have been formulated which at least partially explains the behaviour of GAs and also provides insight into the mechanisms of the algorithm. This Section briefly presents the classical explanation why the simple GA from the previous Section works and comments on recent theoretical developments.

In [Holland 75] the search performed by the SGA is investigated in terms of sampled *schemata*, or hyperplanes. Let $v = v_1 v_2 \ldots v_l$, $v_i \in \{0, 1\}$ denote the genotype of an individual of the SGA, and let a *schema* be a string of length l over the alphabet $\{0, 1, \#\}$. The symbol # matches a 0 or a 1, and a given string (genotype) is said to represent a given schema if it matches the schema at all its fixed positions. Eg, if $l = 6$, the schema #0## is represented by e.g. 00111 and 11001. A schema specifies a hyperplane in the search space corresponding to the set of strings which represents the schema. The idea of Holland's argument is that the strings present in a population estimates the fitness of the hyperplanes they represent, that is, they estimate the average fitness of *all* possible strings representing the hyperplane. Holland has shown that if the estimated fitness of a hyperplane is above average, the number of strings representing that hyperplane in the following generations will increase exponentially, until the representatives occupies a substantial proportion of the population. More specifically, if H is a schema, the *order of* H denoted $o(H)$, is the number of fixed positions of H, and the *defining length of* H denoted $\delta(H)$, is the distance between the first and last fixed position in H. Eg, $o(\#0\#\#) = 3$ and $\delta(\#0\#\#) = 4$. Furthermore, let $\bar{f} = \frac{1}{n}\sum_{i=1}^{n} f_i$ be the average fitness of all individuals in

in Fig. 3.2. When crossover is not performed, copies of the selected parent individuals are added to Π_{new}. Following this step called *reproduction*, every individual is subjected to possible mutation. The mutation operator performs *pointwise* mutation, i.e., given a bitstring, each bit is independently inverted with a given small probability p_m, as illustrated in Fig. 3.2. Each generation is completed with an update of Π_{cur} and evaluation of all individuals, as a basis for the selection to take place in the next generation.

```
generate(Π_cur);
evaluate(Π_cur);
repeat G times
   Π_new := ∅;
   repeat n / 2 times
      select π_1 ∈ Π_cur, π_2 ∈ Π_cur;
      if rand(p_c) then
         crossover(π_1, π_2, φ_1, φ_2);
         Π_new := Π_new ∪ {φ_1, φ_2};
      else
         Π_new := Π_new ∪ {π_1, π_2};
      end;
   end;
   ∀ π ∈ Π_new : mutate(π);
   Π_cur := Π_new;
   evaluate(Π_cur);
end;
```

Figure 3.1: *The SGA. The population size n is assumed to be even.*

Crossover is the main operator of the GA. By recombining good partial solutions, even better solutions will often emerge. Hence, promising regions of the search space are explored and at the end the population converges. Mutation is a secondary operator, although still important. If, at a specific bit position, all individuals have the same value, say 0, the value 1 can never be recovered if only crossover is performed. The main purpose of mutation is to insure that this lost information can be recreated.

are either not applicable or have difficulties finding good solutions. As will become clear in Section 3.3, there is no guarantee that a GA will find a "good" solution to a given, hard problem in the sense that the solution is within some pre-specified distance from the global optimum. However, from the practitioners point of view, a "good" solution to a hard problem is simply one which is better than the best already available solution obtained by any other method.

3.2 The Simple Genetic Algorithm

This Section presents the simplest possible genetic algorithm, denoted SGA. Although the practical value of SGA is very limited, as will be discussed in Section 3.4, a closer look at SGA provides insight into the basic mechanisms of the GA. Furthermore, the original theoretical arguments on why GAs work, which will be presented in Section 3.3, are based on the SGA.

In SGA, the genotype of an individual is simply a bitstring of fixed length l and the fitness of any individual is defined by a function, which given the phenotype represented by a bitstring returns a positive, real value. The SGA is outlined in Fig. 3.1. Routine $generate$ generates the initial, current population Π_{cur}, which consists of n random bitstrings of length l. The population size n is kept fixed throughout the process. Routine $evaluate$ computes the fitness of every individual. One iteration of the outer repeat loop corresponds to the simulation of one generation, hence the parameter G defines the total number of generations. In each generation, a new population Π_{new} is generated. A pair of individuals, π_1 and π_2, is selected from Π_{cur}, a total of $n/2$ times. The selection is proportional to the fitness of the individuals, that is, individual π_k is selected with probability

$$\frac{f_k}{\sum_{i=1}^{n} f_i}$$

where f_k denotes the fitness of individual π_k. The two individuals are selected independently, and every individual can be selected any number of times in the same generation. The parameter p_c is the crossover probability. Routine $rand(x)$ returns true with probability x. When crossover is performed, it generates two offspring ϕ_1 and ϕ_2, which are then added to Π_{new}. Simple $one\text{-}point$ crossover is performed by selecting a cross-point at random and then recombining the two substrings as illustrated

considers only a single solution at a time. Clearly, both EAs and SA are based on models of nature simplified to the extreme extent where it can hardly be recognized. In this sense the analogies to phenomenons from nature should be seen only as sources of inspiration, nothing else. Nevertheless, both types of algorithms have proved to be very useful in optimization.

3.1.2 The Application Area of GAs

Compared to other optimization techniques the advantage of the GA most often pronted in the literature, is the robustness of the algorithm [Goldberg 89a, D. Beasley 93a]. As opposed to the vast majority of other methods, the GA does not rely on any specific properties of the objective function. No information on derivatives are used, in fact the function need not even be continuous. Consequently, the GA is robust in the sense that it can be used for optimization in highly complex and irregular search spaces. This claim of generality of the GA is supported by the literature, in which applications from diverse fields are reported. References to applications of GAs in e.g. biology, engineering, operations research, business and social sciences can be found in [Goldberg 89a]. In [Saravanan 93] more than 400 references to papers on EAs and applications are given, and [Nissen 93] lists about 230 references on EAs in management science. More specific application areas of GAs includes numerical function optimization, combinatorial optimization, image processing, pattern recognition, design and machine learning [Goldberg 89a, D. Beasley 93a]. While most GA research is still being carried out at universities, an increasing number of GA projects are reported from industry [Goldberg 94]. For example, General Electric is using a GA based system to design gas turbines and jet engines, Hughes Missile Systems Company in California is using genetic programing for infrared image target discrimination and Applied Geophysical, Colorado, uses a GA to solve problems arising from seismic surveys related to oil exploration [Goldberg 94].

The price paid for the robustness of the GA is that in general, the algorithm is not competitive for relatively easy or small-scale optimization problems. When highly specialized optimization techniques exist for a given problem, the GA will most likely show inferior performance, both in terms of solution quality and runtime. Rather, the natural application area of the GA is that of very hard problems, for which other methods

current research topics. Since Holland's original work [Holland 75] is very general and quite formal it is hardly suitable as a first approach to GAs.

The GA belongs to a wider class of algorithms, the *Evolutionary Algorithms (EAs)*, surveyed in [Bäck 93]. Although the GA is by far the most well-known EA, this class also consists of *Evolution Strategies (ES)* and *Evolutionary Programming (EP)*. The common feature of EAs is that they all maintain a population of individuals and apply various operators to evolve individuals of increasing quality during time. But there are also a number of significant differences between the algorithms. First of all, while crossover is the most important operator of the GA and mutation is only considered a background operator, as will be discussed in Section 3.2, in ES and EP algorithms, mutation is the main operator and crossover is less important. The EP algorithm doesn't have a crossover operator at all. Furthermore, while the parameters of the GA are fixed, they are dynamically adjusted in ES and EP. The ES algorithm differs from EP and GA algorithms in two major ways. Firstly, the fitness function equals the objective function, while in EPs and GAs, the (relative) fitness of each individual is a (non-trivial) computation based on the objective values of all existing individuals. Secondly, selection is deterministic in ES algorithms, while it is probabilistic in EPs and GAs. Of course, the description of the EA subclasses given here just outlines the general differences. Given a specific EA, it may not clearly belong to any of the three categories as described here. The current state of EA research is surveyed in [De Jong 93].

An interesting special case of the GA is *Genetic Programming (GP)*, introduced by [Koza 92]. Given a specific problem, the idea is to let the GA evolve a complete computer program, which solves the problem sufficiently well. An individual is a program, and fitness is computed by executing the program and measuring how good the program was at solving the problem. By using the genetic operators as usual, the GA evolves programs which perform increasingly well with respect to solving the problem. The language Lisp is especially well suited for GP.

Other types of algorithms than the EAs are inspired by nature. A well-known example is the *simulated annealing (SA)* algorithm introduced in 1983 by [Kirkpatrick 83], which is inspired by thermodynamics. Optimization is performed based on an analogy to the process of cooling down a solid in such a way that thermal equilibrium is obtained. The most fundamental difference between the GA and SA is that the GA considers a number of solutions simultaneously, while the SA algorithm

the capability of adapting to the environment in which they live. The fittest individuals has the highest probability of survival and reproduce the most. The produced offspring resembles their parents so that highly fit individuals usually produces highly fit offspring. Therefore, during the evolution process the fittest individuals tends to increase in numbers while the less fit individuals tends to die out. This well-known principle of *survival of the fittest* was first introduced in 1859 by Charles Darwin in his famous book *The Origin of Species by Means of Natural Selection.*

A natural evolution process can be viewed as an optimization process in the sense that the individuals are 'optimized' for survival. This view is the underlying idea of the GA, which performs optimization by simulating a process of evolution. The algorithm maintains a *population* of *individuals* each of which corresponds to a specific solution to the given optimization problem. A measure of *fitness* defines the quality of a solution. Starting from a population consisting of randomly generated individuals, the evolution process is simulated by considering the population through a number of *generations*. In each generation, new individuals called *offspring*, are generated from existing ones using a *crossover* operator, which imitates sexual propagation. The crossover operator is designed in such a way that the generated offspring resembles the parent individuals. Furthermore, parents are selected for crossover with a probability which depends on their fitness, so that the fittest individuals are selected for crossover with the highest frequency. This scheme enforces the principle of survival of the fittest. With a small probability, each individual is subjected to mutation, or random change, by the *mutation* operator. After having simulated a number of generations, highly fit individuals will emerge, corresponding to good solutions to the given optimization problem.

A distinction is made between the representation, or genetic encoding of a solution and the natural appearance of a solution. In analogy with biology, the genetic encoding is called the *genotype* and the natural appearance is called the *phenotype*. The genetic operators manipulate solutions in terms of their genotypes, while fitness is measured in terms of phenotypes. A function called the *decoder* computes the phenotype corresponding to a given genotype.

Introductions to GAs can be found in many texts, e.g. [Davidor 90, L. Davis 87, L. Davis 91, Michalewicz 92], and [Goldberg 89a] has become the reference textbook. A recent two-part paper [D. Beasley 93a, D. Beasley 93b] provides an introduction to GAs as well as a survey of

Chapter 3

Genetic Algorithms

The purpose of this chapter is to introduce the basic concepts of the Genetic Algorithm (GA), to outline the current status of GA theory and to discuss important practical issues of applying GAs. Section 3.1 introduces the basic idea of GAs. The simplest possible GA is presented in Section 3.2 and Section 3.3 presents the theoretical arguments as to why the simple GA work and also discusses the current status of GA theory. Readers familiar with GAs can skip Sections 3.1, 3.2 and 3.3, which are all introductory. Section 3.4 is devoted to four main issues of applying GAs, all of which are subject of much current research. Furthermore, these issues have been especially important for the algorithms presented in this thesis, and consequently, Section 3.4 is a prerequisite of Chapter 5. Section 3.4.1 discusses the design of suitable encodings and Section 3.4.2 discusses if (and how) problem-specific knowledge should be incorporated into the GA. Various strategies for handling constraints in GAs are discussed in Section 3.4.3, and Section 3.4.4 addresses the practical problem of finding suitable values of the control parameters of the GA.

3.1 Introduction

The concept of genetic algorithms was founded by John H. Holland, whose Ph. D. thesis from 1975 [Holland 75] is considered the origin of the field. One of the main application areas of GAs is that of optimization, which is the only application area considered in this thesis.

3.1.1 The Basic Idea of GAs

The GA is inspired by the process of natural evolution studied in population biology. In nature, the individuals constituting a population has

that the entire routing area is divided into channels by this scheme, hence completely eliminating the need for switchboxes. The obvious drawback of restricting the search space to slicing structures is that if the optimal layout is not a slicing structure, it will never be found. Slicing structures are briefly considered again in in Section 5.2.3.

routing in itself requires the solution of perhaps hundreds of mutually dependent, NP-hard problems, although many of these problems may have relatively small search spaces. Channel routing as well as switch-box routing are NP-hard [Szymanski 85], and so is for example via-minimization [Naclerio 89]. In addition, the problems are mutually dependent, and hence relies heavily on estimations of what will happen in succeeding steps. In other words, the cost functions involved are not accurate, but relies on the accuracy of the estimates. These inherent problems have two important consequences: Firstly, certain steps of the layout synthesis process, or perhaps the whole process, typically have to be iterated a number of times to obtain a satisfactory result, as illustrated in Fig. 2.2. For example, as mentioned previously, it may not even be possible to complete the routing of a given placement without going back and adjusting the placement. Secondly, the quality of solutions obtained for some intermediate step of the process can not be accurately evaluated immediately. For example, two distinct placements can not be safely compared without actually completing the routing of the layouts.

In general, the sooner a step is performed the more it impacts the final layout quality. If for example a very poor placement is generated, it can not be compensated for in succeeding steps, no matter how well these are solved. In this sense the first steps of layout synthesis are more important research areas than later steps, and this is one of the reasons why this thesis focusses on placement and global routing rather than e.g. channel routing. Especially, the floorplanning and placement problems are very hard, and consequently they are often solved manually [Sherwani 93]. As mentioned previously, a key problem here is the estimation of the needed routing area. Naturally, as the layout synthesis process proceeds, estimations become increasingly accurate.

Slicing structures is a class of building-block layouts which have become very popular, since it eliminates a couple of the problems described above. A *slicing structure* is a building-block layout, which can be recursively partitioned or "sliced" by a sequence of horizontal and vertical lines each of which goes all the way through the layout, until no more than one block is present in each partition. For example, the layout of Fig. 2.3 is a slicing structure: A horizontal line can separate blocks A and B from blocks C, D and E. Then, vertical lines can separate A from B, C from D and E, and finally, D from E. A slicing structure simplifies the routing step. Each line corresponds to a channel, and a feasible routing order is the inverse of the order in which the lines were made. Especially, note

channel routers are capable of meeting the lower bound given by the channel density in almost all cases occurring in practice. And when they fail, only one or two additional tracks are needed. For this reason, the channel density is a very important and useful concept in routing area estimation. Unfortunately there is no simple relation between the number of terminals or nets present in a channel and the channel density. As illustrated in Fig. 2.4, if n nets are present, the density can be any integer between 0 and n.

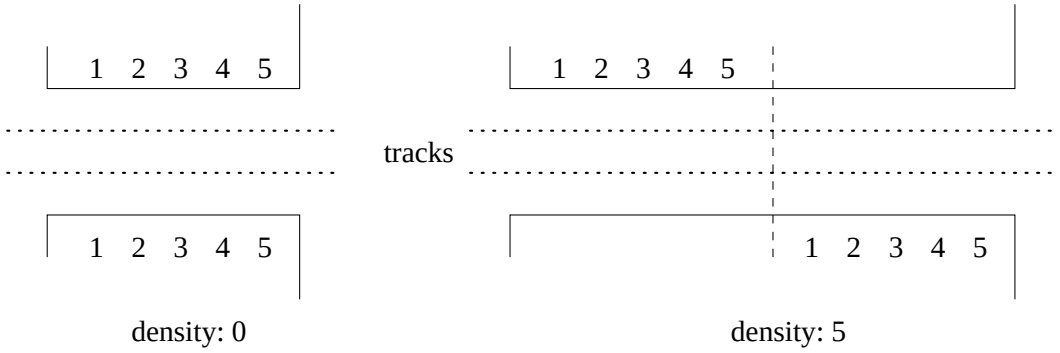

Figure 2.4: *Possible extreme values of the channel density, illustrated with 5 nets present in the channel. In the channel to the right all 5 nets have to cross the column indicated by the vertical dashed line.*

After completion of the routing step, the layout is functionally complete. However, a final step of postprocessing is often performed, in which various kinds of final optimizations are attempted. The most common kind of postprocessing is *compaction*, where the layout is compressed in one or both dimensions subject only to the design-rules. Compaction reduces the total area and may improve performance by reducing wire-lengths as well. Other kinds of postprocessing are via minimization and re-assignment of the layers of some wire segments. The metal layers used for routing have different electrical properties, and hence one layer may be preferable to another. In Fig. 2.3 (d), some wire segments of net 2 has been assigned another layer, allowing net 4 to be placed on top of it. The resulting unused track allows reduction of the total area by subsequent horizontal compaction.

Each step of the layout synthesis process described above is computationally hard to solve. Formulations of the partitioning and floorplanning/placement problems are all NP-hard [Sahni 80, Donath 80]. Common formulations of the global routing problem is equivalent to that of finding a minimal Steiner tree in a graph, which is NP-hard for a single net having more than two terminals [Karp 72]. Hence, global

one at time, using a channel router or a switchbox router as appropriate. Nets may enter a channel by passing one of the two opposite sides at which there are never any fixed terminals from blocks. The channel router assumes that it can decide itself the exact position of where any net enters and/or leaves the channel. When fixing the position of a net crossing on the border between two neighbouring channels, this of course affects both channels and consequently it imposes a partial ordering of the routing of channels. For example, in Fig. 2.3, the channel between blocks C and D has to be routed before the channel between blocks C and A, since otherwise the terminal positions along the bottom side of the latter channel is not fixed before the channel is routed. Given an arbitrary routing region definition, a suitable channel ordering may not exist and it then becomes necessary to introduce a switchbox. Therefore, the channel ordering is also defined during the earlier routing region definition step. In Fig. 2.3 (b), a possible routing order is to first route all vertical channels, i.e., those with terminals along vertical sides, in any order, and then route all horizontal channels, in any order.

The objective of the detailed routers is first of all to complete the routing within the available area. Channel routers also typically attempts to minimize the total wirelength together with two other criteria to be explained shortly: the number of vias used, and the number of tracks used. A *via* is a connection from one routing layer to another, i.e. it is needed wherever a net switches from one layer to another, as illustrated in Fig. 2.3 (c) and (d). Due to the poor electrical properties of vias their usage are often minimized. For a given channel, a set of design-rules and a routing layer, a two-dimensional lattice can be defined in a channel which determines the minimum spacing needed between wires in that layer. The lines of the lattice which are parallel to the sides having fixed terminals are called *tracks*, while the perpendicular lines are called *columns*. By minimizing the number of tracks used for routing the channel router increases the possibility of improving the layout in the succeeding post processing step, which will be described later.

When considering a specific column, a lower bound on the number of wires which has to cross that column is the number of nets having terminals on both sides of the column. When maximizing this quantity over all columns, the resulting value is known as the *channel density*. Given the channel density, the number of available routing layers and the design-rules, the minimum channel width needed by any router to implement the routing can easily be computed. Todays state-of-the-art

The wiring needed to connect a set of electrically equivalent terminals, as specified in the netlist, is called a *net*. In the global routing step, a "global" route for each net is determined in the form of a listing of the routing regions it will use. Hence, the global router determines the approximate route of each net, while not defining the exact position of each wire segment. The typical objective of global routing is to minimize the estimated area and/or total estimated wirelength or the estimated length of specific critical nets, while not exceeding the estimated capacity of any routing region. The result of routing region definition and global routing is shown in Fig. 2.3 (b). In this example all routing regions are channels.

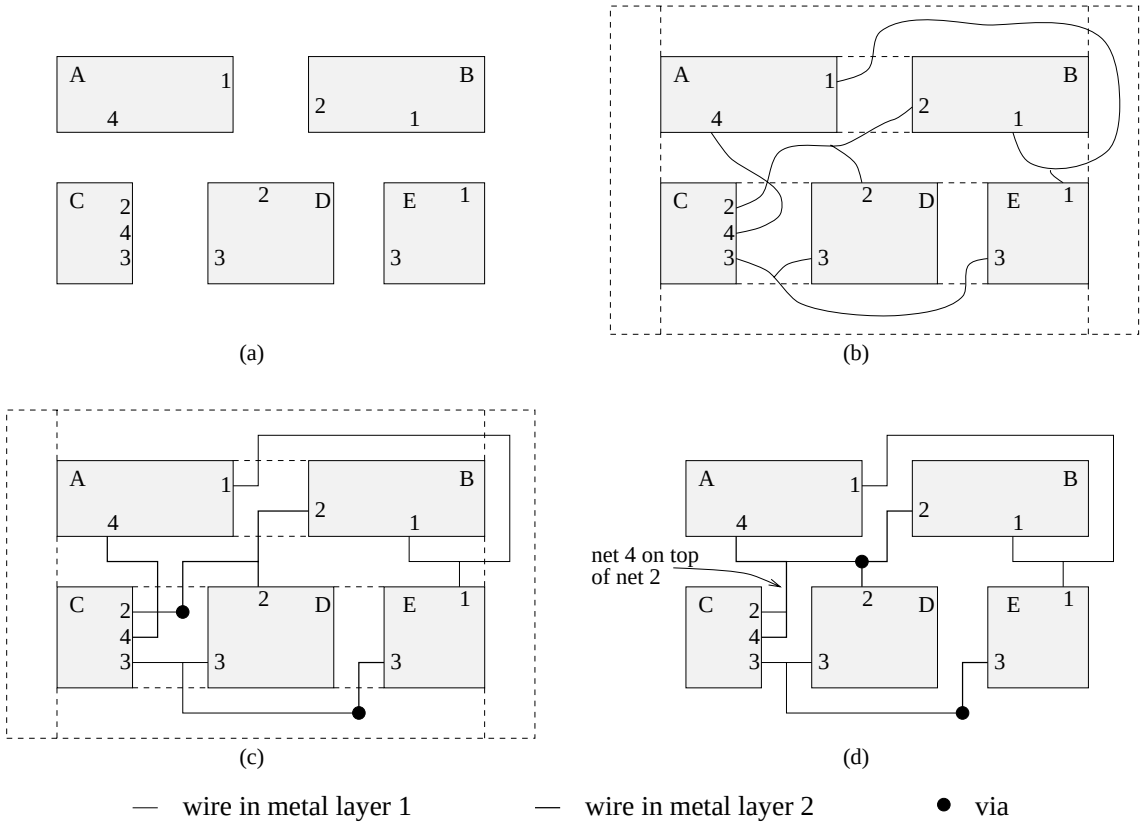

Figure 2.3: *A macro-cell layout with five macro-cells denoted A through E. The numbers 1 through 4 are net-identities, positioned at the terminal locations. (a) shows the layout after placement, and (b) after channel definition and global routing. The dashed lines indicates the borders of the channels. (c) is the result after detailed routing, and (d) is the final layout after a change of layer assignment of net 2 followed by compaction.*

In the detailed routing step, the exact physical position and layer assignment is determined for each net. The routing regions are treated

and different positions of its terminals. A *terminal*, or *pin*, is a point within the block which should be electrically connected to one or more points of other blocks. A block is called *flexible* if its size is known, but its shape and pin positions have not yet been defined. For some subcircuits completed blocks may have been designed earlier, i.e., their exact size, shape and pin positions are known. Such a block is called a *fixed* block or a *macro-cell*.

The purpose of the floorplanning/placement step of layout synthesis is twofold. Firstly, a specific implementation of all flexible blocks is chosen, that is, all flexible blocks are turned into fixed blocks. Secondly, an absolute position and an orientation is determined for each block. The spacing between blocks should be sufficient to allow for all needed interconnections to be implemented in succeeding steps. In the literature, the most common optimization criteria used are minimization of total estimated area and/or total estimated interconnect length. If one or more of the blocks are flexible, step two is referred to as *floorplanning*, while if all blocks are macro-cells, this step is called *placement*. Hence, placement is a special case of floorplanning. A crucial issue of floorplanning/placement is to estimate the area needed between the blocks for routing. The accuracy of this estimate determines the accuracy with which the quality of the placement is assessed. In particular, if the routing area is underestimated, it may not even be possible to implement the interconnections later on without altering the placement. Fig. 2.3 (a) shows a placement of five macro-cells.

Following floorplanning, the purpose of the *routing* step is to implement all interconnections between the macro-cells in accordance with the netlist. As indicated in Fig. 2.2, routing consist of three subtasks: *routing region definition*, *global routing* and *detailed routing*. The first task is to divide the area not occupied by blocks into a number of rectangular areas called routing regions. It is a common assumption that the area used for routing and the area occupied by blocks are disjoint. Consequently, all terminals of a block have to be placed at the periphery of the block. A routing region with terminals along zero or one side only, or with terminals along two opposite sides, is called a *channel*. A routing region, which is not a channel, is called a *switchbox*. The objective of routing region definition is to divide the routing area into as few regions as possible. Especially, the number of switchboxes should be minimized, since a switchbox is much harder to handle in the later detailed routing step than a channel.

design or not, if the design is hierarchical, it will consist of building-blocks from a certain level.

2.3 Building-Block Layout Synthesis

Fig. 2.2 outlines the layout synthesis process for the building-block design style. Due to the inherent complexity of the process it is divided into a number of subtasks, which are solved one at time although they are mutually dependent [Sherwani 93].

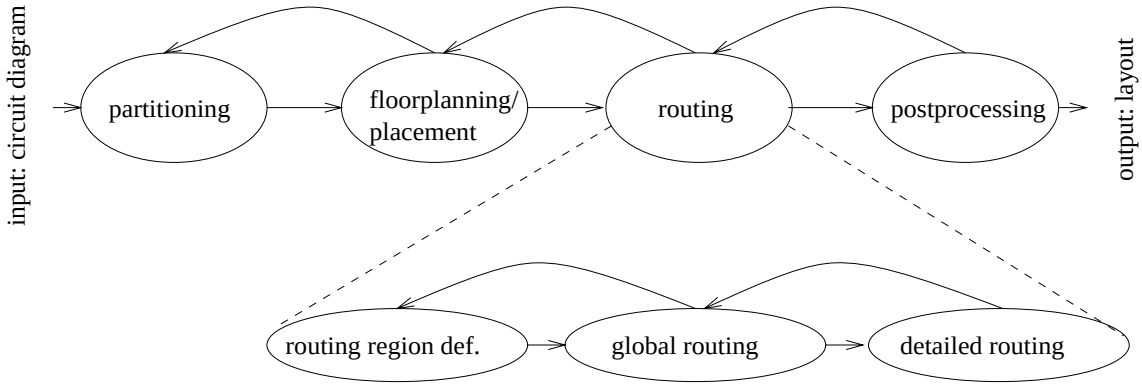

Figure 2.2: *Overview of the layout synthesis process for building-block layouts. The arrows indicates the order in which the steps are performed. After each step the designer either proceeds to the next step if everything is all right, or (s)he may have to go back and redo one or more previous steps in order to meet the overall objectives.*

Most circuits can not be handled by the CAD tools as a whole, due to the required computation time as well as the memory consumption. Therefore, the first step of layout synthesis is *partitioning*, in which the design specified by the given circuit diagram is partitioned into a number of subcircuits of manageable size. For large circuits the partitioning may be hierarchical, cf. Section 2.2. Standard criteria considered by a partitioning algorithm are the number of subcircuits, the size of each subcircuit and the connectivity between them. The output of the partitioning step is a set of subcircuits and a *netlist*, which is a specification of the interconnections to be made between the subcircuits.

Each subcircuit will be implemented by a building-block. The size of a block implementing a specific subcircuit can be estimated from the number and types of components it contains. However, several alternative layouts of the block are possible, leading to different shapes of the block

mance requirements can be met. Prototyping is another application area of semi-custom designs.

A main reason why full-custom design increases the development time significantly is that many of the subproblems to be solved during the layout synthesis phase are significantly harder to solve than the corresponding problems for semi-custom designs. For example, interconnections in a standard-cell layout is typically implemented in such a way that only one dimension needs to be considered at a time. In a building-block layout this problem is truly two-dimensional. Consequently, many of the key subproblems of layout synthesis are solved by tools, which are design style specific. Improved CAD tool performance is first of all needed for the full-custom design style. Inherently, these tools often perform relatively poorer than their semi-custom counterparts, and this is one of the reasons why the work presented in this thesis is concentrated on the full-custom design style.

Table 2.1 summarizes the comparison of the full-custom and semi-custom design styles. Generally speaking, as more restrictions are imposed on the layout, the differences listed will be increasingly prevalent. In other words, the comparison of Table 2.1 still holds if e.g. "full-custom" is replaced by "standard-cell" and "semi-custom" is replaced by "gate array".

Criterion	Preferable design style
flexibility wrt. requirements	full-custom
performance of circuit	full-custom
layout area	full-custom
cost per circuit	full-custom
development time and cost	semi-custom
tool support	semi-custom

Table 2.1: *Comparison of design styles.*

Real-world circuit designs often consists of a mixture of layout styles. Some parts of the layout may be constructed from library standard-cells, while other, more critical parts are full-custom design. To handle the complexity of large circuits, the design may be hierarchically structured into two or more levels. As mentioned previously, blocks of a full-custom layout may be constructed from a number of smaller blocks. Standard-cells may also be grouped together to form a building-block at the next higher level of the hierarchy. So whether standard-cells are used in a

cells. The cells are placed in rows as illustrated in Fig. 2.1 (right) and they are almost always designed so that some of the wires, typically power supplies and clock signals, are positioned at the same height in all cells. These wires are then automatically connected by abutment of the cells. While a full-custom layout rarely consist of more than 50 cells, a standard-cell layout may consist hundreds of cells.

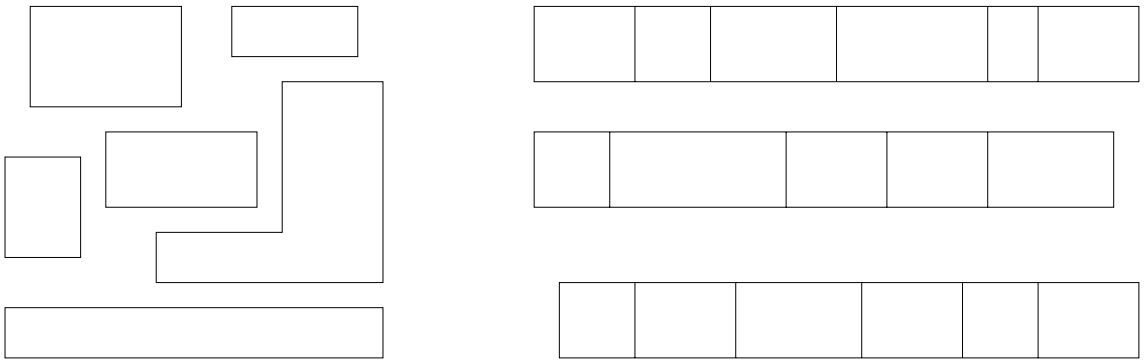

Figure 2.1: *Left: A full-custom layout. Right: A standard-cell layout. Only outlines of cells are shown.*

The other semi-custom layout styles mentioned are all more restrictive than the standard-cell layout style. Broadly speaking, they all restrict each cell to implement a single gate only, and the placement of the gates are restricted to matrix-like structures. For a further discussion of these layout styles, the reader is referred to [Sherwani 93].

The choice of a design style for a given application depends on many factors. The layout density, i.e., the number of transistors per area unit, is highest when using the full-custom design style, which consequently gives the smallest area and the lowest cost per produced circuit [1]. The best circuit performance is also obtained by a full-custom design, and it is the most flexible with respect to atypical or severe circuit requirements, which may not be satisfiable using standard-cells from a library. For these reasons, the full-custom layout style is generally preferred for mass-produced circuits and for circuits having to meet strict performance requirements, such as CPUs. The main drawback of the full-custom design style is that it increases the development time and hence cost, significantly. Therefore, a semi-custom layout style is generally preferred for circuits to be produced in medium sized quantities, provided per-

[1] Here it is assumed that the circuit do not have an obvious regular structure. For example, RAM circuits are highly regular and have very high transistor densities. However, application specific circuits are rarely very regular by nature.

to obtain a satisfactory result.

The silicon compilation process is divided into two consecutive phases, *high-level synthesis* and *layout synthesis*, the latter of which is also known as *physical design automation* or *physical layout generation*. Given a high-level description of an algorithm, the output of the high-level synthesis phase is a detailed *circuit diagram*, which describes the circuit solely in terms of gates and interconnections between gates. Then, from the circuit diagram, a layout of the circuit is generated as the result of the layout synthesis phase. High-level synthesis is not discussed in this thesis.

[Mead 80] and [Weste 88] are classical textbooks on VLSI design in general. More advanced topics are discussed in [Glasser 85]. Silicon compilation is the topic of [Gajski 88], which also presents a number of specific silicon compilers. A recent and well-written textbook on layout synthesis is [Sherwani 93].

2.2 Design Styles

Fundamental to the layout synthesis phase is the choice of a *design style* or *layout style*, which specifies various degrees of structural regularity of the generated layout. The layout styles can be classified as either *full-custom* or *semi-custom*.

In a full-custom layout the circuitry is partitioned into a relatively small number of *cells*, each of which implements a specific part of the required functionality. A cell of a full-custom layout is also called a *block* or a *building-block*. A block may be constructed from smaller blocks or it may be designed manually or automatically by e.g. a *module generator*. The blocks are then placed and interconnected. The characteristic feature of the full-custom layout is irregularity. Each block can have any size and, in general, any rectilinear shape. Furthermore, the blocks can be placed at any position subject only to the limitations imposed by the design-rules, as illustrated in Fig. 2.1 (left).

In contrast to the full-custom layout, the semi-custom layout styles introduces various degrees of regularity. *Standard-cells*, *gate arrays*, *sea-of-gates*, and *gate matrix* layouts are all semi-custom layout styles. A standard-cell layout is made up of *standard-cells*, which are rectangular and have identical height but varying width. Each cell is either designed manually or stems from a *cell library* of predesigned cells. Commercial silicon compilers typically comes with libraries of a few hundred standard-

or using abutment) assigned to an appropriate layer. Similarly, a transistor is specified as a specific combination of shapes of different layers at the same physical location, and a connection between two distinct layers is specified by an appropriate shape at the required position, which is assigned a layer dedicated to this purpose.

The software used for silicon compilation is called a *silicon compiler* or *VLSI CAD tools*. As indicated by the first designation, a silicon compiler has a lot in common with a traditional compiler for a programming language. For example, the generated code has to obey certain syntax rules and it should be effective. Similarly, the layout generated by the silicon compiler has to obey a set of *design rules* given by the manufacturer. The design rules specifies minimum sizes of shapes, minimum distances between shapes of distinct layers, limitations as to which layers can be placed on top of each other, etc. The generated layout should also be "effective" in the sense that certain criteria should be optimized. Exactly which criteria are important depends on the specific application. If the circuit will constitute the CPU of a computer, speed will be crucial. For satellite applications, a low power consumption will be of major importance. For medical equipment, reliability hopefully has the highest priority. To control the programs in a washing machine, production cost per unit will be a major concern, and for applications in military equipment, insensitivity to large temperature variations may be a very important requirement. The optimization criteria most frequently studied in the literature and also adopted in this work, are *minimization of layout area and total wire length.* Minimizing area means maximizing yield and hence minimizing cost. It also means maximizing the functionality which can be implemented on a single chip. Minimizing wire-length to some extent means minimizing delay and hence maximizing speed.

Several other similarities between compilers and silicon compilers exist. However, there are also major differences caused by the tremendous difference of the complexity of the problems considered. As will be described in Section 2.3, the silicon compiler has to deal with a sequence of mutually dependent, NP-hard optimization problems. Consequently, it is based on a large number of heuristics, some of which are not always capable of producing satisfactory results. Therefore, a silicon compiler is not a single program, rather it is a collection of a (large) number of integrated tools, each of which can be executed individually. Most silicon compilers allow for user intervention at various points in the process so that one or more critical steps of the process can be carried out manually

Chapter 2

Layout Synthesis

The aim of this chapter is to present the basics of layout synthesis, of which familiarity will be assumed in succeeding chapters. The introduction is brief, but provides references for further reading. Section 2.1 introduces layout synthesis in general, and Section 2.2 presents the concepts of full-custom and semi-custom layouts. An overview of layout synthesis of building-block layouts, which is the specific problem area of this thesis, is given in Section 2.3. Readers familiar with layout synthesis can skip this chapter.

2.1 Introduction

Given a high-level description of an algorithm, the task of (semi-)automatically translating the description into an exact specification of an integrated circuit, which implements the algorithm, is referred to as *design automation* or *silicon compilation*.

Hardware description languages, of which VHDL is the most popular [Lipsett 89], is commonly used for the input high-level descriptions. Alternatively, general purpose programming languages such as Caml or C++ can be used [Caprani 91, Andersen 92]. The output generated is a detailed description of the circuit, which comprises all information needed by the manufacturer for the production. The description, commonly referred to as the *layout*, is a specification of a (large) set of planar, geometric shapes. Each shape is assigned to a specific *layer* of the circuit, which is identified by a name or a colour. Typically rectangles are the only shape allowed. Some layers of the circuit are used for interconnections, or wires, while combinations of other layers may correspond to e.g. a transistor or an electrical connection between two specific layers. Hence, in the layout, a wire is specified as a set of rectangles (overlapping

ideas on how to handle the most important identified problems of the proposed algorithms. Finally, the main conclusions of this thesis are given in Chapter 7.

1.4 Description of Papers

As previously mentioned, the thesis is based on research papers, four of which constitutes Appendices A through D. The following presents the papers by listing information on co-authorship, publication status, etc. For a description of the contents of the papers and the relationship between the papers the reader is referred to Chapter 5. The thesis is based on the following papers:

1. Henrik Esbensen, "A Genetic Algorithm for Macro Cell Placement," *Proc. of The European Design Automation Conference*, Hamburg, Germany, pp. 52-57, 1992.

2. Henrik Esbensen, Pinaki Mazumder, "SAGA: A Unification of the Genetic Algorithm with Simulated Annealing and its Application to Macro-Cell Placement," *Proc. of The 7th International Conference on VLSI Design '94*, Calcutta, India, pp. 211-214, 1994.

3. Henrik Esbensen, Pinaki Mazumder, "A Genetic Algorithm for the Steiner Problem in a Graph," *Proc. of The European Design and Test Conference*, France, Paris, pp. 402-406, 1994.

4. Henrik Esbensen, "Computing Near-Optimal Solutions to the Steiner Problem in a Graph Using a Genetic Algorithm," Technical report DAIMI PB-468, Computer Science Department, Aarhus University, February 1994.

5. Henrik Esbensen, "A Macro-Cell Global Router Based on Two Genetic Algorithms," *Proc. of The European Design Automation Conference*, Grenoble, France, pp. 428-433, 1994.

Appendix A presents an improved version of the algorithm described in the first paper. An extended version of the second paper constitutes Appendix B, the fourth paper constitutes Appendix C and the fifth paper constitutes Appendix D. The papers have been reformatted in order to obtain a consistent typography of the thesis.

a general background in computer science, but no specific knowledge of VLSI layout generation or genetic algorithms.

The main part of the thesis is the second part, which consists of Appendices A through D. Here the four main topics considered in this work are presented, each one in the form of a paper. The papers assumes some knowledge of layout generation and/or genetic algorithms, which can be acquired by reading the first part of the thesis.

The rest of the first part of the thesis is organized as follows: A brief introduction to macro-cell layout synthesis is given in Chapter 2. This includes a brief account of layout styles, with particular emphasis on the characteristics of the macro-cell layout style. An overview of the individual steps of the macro-cell layout synthesis process is given, focussing on the complexity of each step and the interrelationship between the steps. Readers familiar with macro-cell layout synthesis can skip this Chapter.

The concept of genetic algorithms is presented in Chapter 3. A brief overview is given together with an account of the underlying ideas. Focus is on theory of genetic algorithms, practical considerations of applying GAs and current research topics. Readers familiar with GAs can skip Sections 3.1, 3.2 and 3.3, while Section 3.4 discusses various design options, and constitutes the basis for later discussions in Chapter 5.

Related work are reviewed in Chapter 4. The presentation given is not meant to be exhaustive, but describes the state-of-the-art approaches to placement and global routing of macro-cell layouts. Furthermore, Section 4.3 gives a brief overview of previous applications of GAs within the layout synthesis area in general. The GA has been applied to e.g. standard-cell placement and channel routing.

A summary of the conducted research is given in Chapter 5. This includes a brief description of each of the four algorithms presented in the appendices, and an account of the relationship between the different parts of the work. The obtained results are evaluated by comparing the performance of the proposed algorithms to that of the best existing approaches, cf. Chapter 4. Advantages and disadvantages of the proposed algorithms are discussed, and based on the presentation of design options in Section 3.4, the important GA characteristics of the algorithms are summarized. The algorithms have some common properties, which is believed to be the main reasons for the obtained performance.

Chapter 6 points at some possible directions for future work on the basis of the evaluation presented in the previous chapter. This includes

on worst possible result relative to the global optimum, worst-case time analysis, etc. Apart from the fact that these kinds of analysis are very hard if not impossible to apply for CAD algorithms, such performance measures are not the immediate interest of the CAD engineer. The engineer wants to know if the result obtained is likely to be better than results obtainable using other tools, and which absolute runtime is to be expected. The latter viewpoint does not rule out or conflict with the first, instead the two viewpoints supplement each other. But in this application area it is natural to put emphasis on the latter viewpoint. For example, to obtain a good absolute runtime, the worst-case time complexity should be considered. However, if the problem sizes considered in practice can always be solved in, say, a couple of seconds, the design engineer will not worry whether the complexity of the algorithm is $O(n \log n)$ or $O(n)$. Later on (s)he might be spending CPU-hours solving another subproblem of the layout generation process. And if the input causing the worst-case time complexity rarely or never occurs in practice, the theoretical complexity of an algorithm will not provide the engineer with much information as to which runtime should be expected.

For these reasons, the performance of the algorithms developed in this work is evaluated by implementing each algorithm and interfacing it to a set of existing CAD tools. The algorithms are then tested using benchmark problem instances wherever possible, and the obtained quality for the benchmarks as well as absolute runtimes are compared to those of current state-of-the-art tools. Comparison is done to any state-of-the-art approach, no matter if it is based on a completely different strategy such as branch-and-bound or simulated annealing. Again, from the CAD engineers point of view, the CAD approach is interesting if and only if it is competitive to other approaches, whereas a typical CAD performance measure such as on-line or off-line performance is of no immediate interest on its own.

1.3 Organization of the Thesis

The thesis is based on papers written during my Ph.D. study and consists of two parts. The first part consists of Chapters 1 through 7. Here the relevant topics are introduced, the obtained results are summarized and related to earlier approaches, and possible directions for future work are discussed. This part of the thesis is written assuring that the reader has

does not capture the temperature dependency of delay. There is also geometric aspects with no obvious physical counterparts. For example, two geometrically distinct paths of a wire may be equivalent with respect to their physical properties. The choice of viewpoint is caused by the interests and background of the author as well as the need to demarcate the topic.

As described in Section 1.1 the overall purpose of this work is to investigate if the application of GAs can improve the performance of CAD tools available to design engineers. The application oriented purpose of course affects the approach taken, which also becomes application oriented. The remaining of this Section discusses important consequences of the application oriented approach.

When considering performance of the developed algorithms, the obtained layout quality is assumed to be more important than runtime. Of course runtime should be within reasonable bounds to minimize the development time of a circuit. But if a circuit is to be mass-produced, even a slight improvement of a quality factor such as area will mean a significant economical advantage. Hence, designers will most likely be willing to spend the extra development time caused by slower CAD tools. Furthermore, when producing large, high-performance circuits, the designers may not even have a choice. Just to obtain a layout satisfying the requirements with respect to, say, timing and area, the designers may have to use the tools producing the best possible layout quality, no matter what there runtime requirements are. It should be emphasized that the priority of quality as being more important than runtime does not mean that runtime has no importance. It is merely a matter of priorities.

The application oriented approach matches the application of GAs well. The GA has not been chosen as the subject of this study because of an interest for the GA in its own right, although I do find the basic idea of the GA intuitively appealing. However, the main reason to investigate the GA is that, judging from results obtained within other fields, the algorithm could potentially be able to produce high-quality layouts. Furthermore, it has not yet been investigated for this application area. The GA is widely accepted as being able to generate high-quality solutions, while they often have problems competing with respect to runtime. These characteristics matches our priorities.

Performance evaluation is strongly impacted by the application oriented approach. From a theoretical point of view, the performance of an algorithm can be evaluated in terms of convergence proofs, bounds

area and delay) of the generated layouts the most, as will be explained in Chapter 2.

There are two main reasons why the genetic algorithm (GA) has been selected as the algorithm to be investigated for the chosen problems. Firstly, almost no previous work exist on GAs for macro-cell layout problems, cf. Chapter 4. Secondly, the GA has been successfully applied for several other highly complex optimization problems, cf. Section 3.1.2. There is also a few promising applications of GAs for standard-cell layout problems. Therefore, it is a natural idea to investigate if the GA can be used to improve the performance of macro-cell layout tools.

The purpose of this thesis is to contribute answers to questions such as:

- Can the concepts of genetic algorithms be successfully applied for placement and global routing of macro-cell layouts, when the main objective is high-quality results ?
- Which performance can be obtained ?
- Are there any algorithmic design principles, which seems to be yielding the highest performance ? If so, what are these principles ?
- What are the main problems of a GA-based approach to these problems ?

1.2 Chosen Approach

This thesis focusses on the combinatorial optimization aspects of layout generation rather than on the physical aspects. A layout is mainly viewed as a set of two-dimensional geometrical objects which should be organized in the plane such that some measure of quality is optimized. The quality measure is definable in terms of sizes and shapes of geometric objects, distances between different objects, etc. This is in contrast to a physicists point of view, in which a layout is a set of interconnected transistors and the concepts discussed are rise and fall times, capacities, conductance, and so on. The two viewpoints, or "worlds", are of course very tightly related and many concepts are (completely or partly) transferable from one world to the other. For example, the signal propagation delay through a wire depends on the geometric dimensions of the wire, i.e. this aspect of delay is transferable. On the other hand, the geometric viewpoint

the use of procedures in imperative programming languages. A level in the hierarchical layout consist of a number of interconnected blocks, or cells, each of which implement some part of the required functionality. Each block is in turn made up of a number of smaller, interconnected blocks. At the bottom level of the hierarchy, a block consists of a number of interconnected transistors, which implement e.g. a single gate or a register. Commercial CAD tool packages includes libraries of so-called standard-cells, which implements a wide range of relatively simple and common functions. Standard-cells are uniform and therefore relatively easy to put together to form larger blocks. Hence standard-cells simplifies and speeds up the construction of the lowest levels in a hierarchical design. However, if the performance requirements for the circuit to be designed are very high, a design based on standard-cells may not be able to meet the requirements and the designer will then have to construct customized cells for the lowest level of the hierarchy. Whether standard-cells or customized cells are used initially, above some level of the hierarchy, the blocks will typically have varying sizes and shapes, one of the reasons being the interconnections made at lower levels. Such non-uniform blocks are called macro-cells, and a level of the hierarchy consisting of macro-cells is called a macro-cell layout. The concepts of standard-cells, macro-cells and customized cells, etc. will be discussed in more detail in Chapter 2. If customized cells are used as the basic building blocks of a circuit in order to meet the performance requirements, the layout will often be of the macro-cell type from a lower level in the hierarchy than if standard-cells were used.

Most CAD tools are customized towards a particular layout type, or layout style, so that the same sub-problem of the layout generation process is solved by distinct tools, depending on the current layout style. As one would expect, the best tools have been developed for standard-cells, since a standard-cell layout is easier to generate than a macro-cell layout implementing the same functionality. Due to the increased complexity of the sub-problems to be solved when using the macro-cell design style, manual intervention is required more often to obtain a satisfactory result. In other words, the macro-cell layout tools are the ones that needs improvement the most, and are also the ones which have the largest potential for improvement. This is the reason why this thesis focusses on tools for the macro-cell layout style rather than standard-cell layouts. Furthermore, the specific sub-problems chosen, placement and global routing, are the most important ones in the sense that they impact the quality (e.g.

Chapter 1

Introduction

The subject of this thesis is design of genetic algorithms for solving certain sub-problems arising in automatic layout generation of VLSI (Very Large Scale Integration) integrated circuits. Section 1.1 of this introductory chapter briefly introduces the subject and describes the purpose of the thesis. The approach taken is described in Section 1.2. The structure of the thesis and some guidelines to the reader are given in Section 1.3 and a brief introduction to the research papers on which the thesis is based is found in Section 1.4.

1.1 Subject and Purpose of the Thesis

During the last decades, the complexity of integrated circuits has increased exponentially. In the 1970's a typical microprocessor such as the Intel 8080 consisted of about 5,000 transistors while in 1993 Intel's state-of-the-art processor Pentium contains 3.1 million transistors on a 17.2 by 17.2 mm area. This extremely rapid development will most likely continue, at least for some years to come. To handle the complexity of todays circuits the design engineers are totally dependent on powerful CAD tools to facilitate a (semi-)automatic transformation of a high-level description of a circuit into an equivalent physical layout. The capabilities and limitations of such tools have crucial impact on the performance and cost of the produced circuits as well as on the resources required to develop a circuit, both in terms of time and financial cost. Consequently, the area of CAD tools for design of VLSI circuits, also referred to as design automation or silicon compilation, is a very important and increasingly growing research area.

Due to the inherent complexity of any non-trivial circuit, the layout is commonly hierarchically structured. This situation is analogous to

Acknowledgements

I am truly grateful for the inspiration, advice and support I have received from many people during my work on this thesis. It is not possible to mention everyone whose interaction contributed to this work, but I would like to thank some of the people that have helped me the most.

First of all I want to express my gratitude to my wife Pia for her all-out support, which included allowing me to go to Michigan for a one year period while she had to stay in Denmark.

At Aarhus University I would like to thank my supervisor Peter Møller-Nielsen for being a very inspiring advisor and for always supporting and encouraging me. Also thanks to Ole Caprani, Holger Gotup and Brian Mayoh for many constructive discussions and suggestions.

At University of Michigan I owe thanks to Trakis Mahmoud for welcoming me in his group and for his useful suggestions and encouraging comments. Also thanks to my fellow students Michael Smith, Kushro Shahookar, Sundaram Mohan and Venky Ramachandran for the invaluable interaction.

Thanks to Peter Kornerup at Odense University for helping me at several occasions, and thanks to Jens Clausen, Copenhagen University, for advice and suggestions regarding a specific part of my work. Last but not least I would like to thank Jens Lienig, Concordia University, Montreal, Canada, for all our discussions.

I am of course also very grateful for the financial support which made this work possible. Support has been provided by the Computer Science Department at Aarhus University, the ADA research project [1], and the Danish Research Academy.

Henrik Esbensen
Computer Science Department
Aarhus University
July 1994

[1] Alternative Data Representations, Algorithms and their Implementation, Danish Natural Science Research Council, grant no. 5.21.08.02.

here was a "real-world" area offering a large number of very hard and closely interrelated optimization problems, for which good solutions were actually needed. Although I did not have any background in physics, I discovered that to a large extent it is possible to abstract away the physics and consider the problems from a mathematical point of view. So in 1989 I chose routing of macro-cell layouts as the topic of my Masters thesis [Esbensen 90], which I did in cooperation with two fellow students. During this work I got some experience with heuristic algorithms, and I started gaining interest in stochastic approaches.

In March 1990 I attended the European Design Automation Conference in Glasgow, Scotland, and heard the presentation of K. Shahookar and P. Mazumder's paper "GASP: A Genetic Algorithm for Standard Cell Placement". This was the first time I ever heard about genetic algorithms and I found the underlying idea intuitively very appealing. So back home again I designed my own genetic algorithm and started doing experiments. The algorithm never worked well. In retrospect, I know a lot of reasons why, but at that time of course I didn't. However, I still liked the concept of genetic algorithms, so after receiving my Masters degree in the beginning of 1991, I applied for and got a Ph.D. scholarship with the purpose of investigating the application of genetic algorithms within the area of layout synthesis.

This had been studied for some time by Professor Mazumder and his group at Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI, USA. So in the winter of 1991/1992 an arrangement was made allowing me to visit the group at University of Michigan for one year, starting in June 1992. Hence, I ended up working with the people who's work I first heard about in Scotland two years earlier. The year in Michigan was extremely beneficial in many ways. To mention one of the specific opportunities it gave me, I attended the course 'Complex Adaptive Systems' held by John Holland, the founder of genetic algorithms.

In June 1993 I returned to Aarhus University and this thesis was submitted to the Computer Science Department, Aarhus University, in July 1994 to fulfill the requirements of the Ph.D. degree.

Henrik Esbensen

Computer Science Department

Aarhus University

July 1994

Preface

This thesis is the result of a 3 year PhD study carried out in the period from 1991 to 1994. The research was done at Computer Science Department, Aarhus University, Denmark, and supervised by Associate Professor Peter Mølle-Nielsen. As an integral part of the study I was at Department of Electrical Engineering and Computer Science, University of Michigan, MI, USA, for a one year period from June 1992 to June 1993. During this time Associate Professor Prabhakar Mazumder was acting supervisor.

The purpose of this research is to investigate the possibilities of applying evolution-based algorithms to solve various subproblems arising in layout synthesis of VLSI integrated circuits. Especially, the research has focussed on the design of genetic algorithms for placement and global routing of macro-cell layouts. The goal is to contribute to answers for questions such as: Can the concepts of genetic algorithms be successfully applied within this application domain, when the objective is to generate high-quality layouts? Which performance can be obtained? Which algorithmic design principles yields the highest performance? What are the main problems of the genetic algorithm approach?

The thesis consists of two parts, the first of which introduces the relevant topics, summarizes the obtained results, relates the results to earlier significant approaches, discusses possible directions for future work, etc. The main part of the thesis is the second part, which is structured as four appendices. Here the main results are presented in the form of four separate research papers, written during the study.

To put the work presented in this thesis into perspective it is appropriate to give a brief account of my background and the circumstances that lead me to work in this field. I first became interested in the area of layout synthesis when I had a course on VLSI design as part of my Masters programme in computer science. Layout synthesis can be studied from a number of different viewpoints, but what I found fascinating was the complexity of the optimization problems involved. Instead of creating more or less artificial problems and then study how to solve them,

komplementerer i en vis forstand hinanden, og dette var inspirationen til udviklingen af en placeringsalgoritme kaldet SACA, der kombinerer en CA med SA således at algoritmen i den initielle fase er en ren CA, men derefter gradvist og adaptivt skifter over mod SA og til slut kan være ren SA. SACA er først og fremmest inspireret af [Rosenbluth 91], men er mere generel og i højere grad adaptiv. SACA er publiceret som [Eschensen 94a], og artiklen gengivet i appendix B er en udvidet version heraf.

Som tidligere nævnt er SPG et delproblem i forbindelse med global routing, hvorfor en CA til SPG blev udviklet som det første skridt mod en CA baseret global router. Den første version af algoritmen blev publiceret som [Eschensen 94b]. Da resultaterne var meget lovende, blev algoritmen videreudviklet, selvom karakteristika for de benchmark grafer der nu blev arbejdet med, ikke længere var direkte relevante for den globale router. Appendix C gengiver artiklen [Eschensen 94c].

Endelig beskrives i appendix D den CA baserede globale router, der som en del algoritmen anvender CA'en fra appendix C. Den globale router er publiceret i [Eschensen 94d], som gengives i appendix D.

er det alligevel klart, at alle de udviklede algoritmer er konkurrencedygtige sammenlignet med state-of-the-art algoritmer mht. den opnåede løsningskvalitet. Specielt opnås meget lovende resultater med SPG-algoritmen og den globale router. På den anden side er SPG-algoritmen den eneste, der også kan konkurrere på køretid. De øvrige algoritmer er alle betydeligt langsommere end dem de sammenlignes med, ofte en størrelsesorden. Der er imidlertid et antal kendte årsager til at de nuværende implementationer af algoritmerne er så tidskrævende, hvorfor det forventes at køretiderne kan reduceres væsentligt, som beskrevet i kapitel 6. Desuden er GA's meget velegnede til parallelle implementationer.

Udover forslag til hvordan køretiderne kan forbedres, diskuterer kapitel 6 en mulig ændring af de valgte optimeringskriterier samt kombinationen af disse. Desuden diskuteres mulighederne for at sammensætte placering og global routing til én opgave mhp. at reducere de uundgåelige problemer som følger af unøjagtige estimater.

Endelig præsenterer kapitel 7 afhandlingens hovedkonklusioner. Da de udviklede algoritmer er konkurrencedygtige mht. løsningskvalitet konkluderes det, at GA's er en lovende metode til de undersøgte problemer. Med undtagelse af SPG-algoritmen er køretiderne ikke tilfredsstillende, men det forventes at betydelige forbedringer kan opnås på dette punkt. Den anvendte metode til håndtering af begrænsninger menes at være den væsentligste årsag til de opnåede resultater. Derfor konkluderes at for problemer med lignende karakteristika skal man fravælge den traditionelle binære problemrepresentation og de traditionelle genetiske operatorer, an disse forhindrer at alle løsninger altid kan opfylde alle begrænsninger. I stedet opnås bedre resultater ved design af problemspecifikke repræsentationer og operatorer.

Artiklerne: Appendices A til D

Artiklen gengivet i appendix A præsenterer en GA til placering af makroceller. Hovedidéen er at betragte placeringsproblemet som en generalisering af det to-dimensionelle binpacking problem, til hvilket en GA beskrives i [Kröger 91]. Placeringsalgoritmen blev først publiceret som [Esbensen 92] men er senere blevet forbedret på en række punkter, og artiklen i appendix B beskriver den nye algoritme.

De typiske konvergensforløb af en GA og simulated annealing (SA)

ulated annealing [Upton 90a, Upton 90b], en branch-and-bound algoritme [Goodera 91], og to større layoutsystemer, BEAR [Eschermann 88, Dai 89] og TimberWolfMC [Sechen 88a, Sechen 88b], der begge indeholder gode placeringsalgoritmer. Desuden er GARP medtaget, somvidt vides er den eneste tidligere GA til makro-celle placering [Chan 91, Shahookar 94a]. To globale routere præsenteres: Mercury [Nishizaki 89], der er baseret på heltalsprogrammering, og den globale router indeholdt i TimberWolfMC. Et delproblem, der opstår i forbindelse med global routing er Steiner problemet i en graf (SPG), somdenne afhandling også præsenterer en GA til. Derfor præsenterer kapitel 4 også en state-of-the-art SPG algoritme baseret på branch-and-cut [J. E. Beasley 89, Lucena 92]. To andre SPG-algoritmer præsenteres i appendix C, herunder en GA [Kapsalis 93] somvidt vides er den eneste tidligere GA til SPG.

Den centrale del af afhandlingens oversigtsdel er kapitel 5, hvori de udviklede algoritmer opsummeres og evalueres. Først præsenteres og diskuteres nogle antagelser vedrørende optimeringskriterier og teknologier somer lagt til grund for de udviklede algoritmer. Dernæst fortælles projektets udviklingshistorie: Hvilke algoritmer er udviklet, hvorfor, hvad er hovedideerne bag dem, og hvordan er de relateret til hinanden. Afhandlingen præsenterer to algoritmer til placering af makro-celler, en algoritme til SPG og en global router. Algoritmernes GA-karakteristika opsummeres i termer af de fire problemstillinger introduceret i afsnit 3.4. Her er den vigtigste pointe at alle algoritmerne udelukkende betragter lovlige løsninger, dvs. at repræsentationen af en løsning, dekoderen (en algoritme somtolker repræsentationen) og/eller de genetiske operatorer til enhver tid sikrer at enhver løsning er komplet og overholder alle begrænsninger. Alternativt kan man tillade løsninger, der overtræder (nogle af) begrænsningerne, og istedet tilføje straffed til costfunktionen. Det er også karakteristisk at problemspecifik viden stort set ikke udnyttes, og at det har været relativt let at finde passende værdier for kontrolparametrene. De udviklede algoritmers estimater (der direkte eller indirekte indgår i costfunktionerne) sammenlignes med de estimater, der anvendes i algoritmerne beskrevet i kapitel 4. Sammenligningerne viser, at førstnævnte estimater generelt er de mest nøjagtige og dermed også de mest tidskrævende at beregne.

Mange faktorer vanskeliggør helt retfærdige sammenligninger af de udviklede algoritmers ydeevne med eksisterende algoritmers, hvorfor de fleste sammenligninger er forbundet med visse forbehold. På trods heraf

makro-celle layout består af et hierarki af blokke (eller makro-celler), der er indbyrdes forbundne med ledninger. En blok på blad-niveauet indeholder transistorer, der implementerer en given funktionalitet, mens blokke på højere niveauer består af et antal mindre, indbyrdes forbundne blokke. Automatisk generering af et givet niveau i hierarkiet opdeles traditionelt i et antal delopgaver, der løses uafhængigt på trods af stærke indbyrdes afhængigheder. To af delopgaverne er *placering* af blokkene, som efterfølges af routing, dvs. implementation af forbindelserne mellem blokkene. Routing opdeles igen i global og lokal routing, hvor *global routing* drejer sig om at bestemme ledningernes overordnede, omtrentlige router. Det er karakteristisk for alle delopgaverne at de er NP-komplette. Desuden kompliceres layout genereringen af indbyrdes afhængigheder mellem opgaverne, som fører til udbredt anvendelse af estimater for konsekvenserne af løsning af endnu ikke udførte delopgaver. Mål andre ord er der støj på de anvendte costfunktioner. Placering og global routing er de to delopgaver, der har størst betydning for den opnåede layout kvalitet, hvilket er en væsentlig årsag til at netop disse to opgaver er valgt som emner for denne afhandling.

Genetiske algoritmer (GAs) introduceres i kapitel 3. Den grundlæggende idé er at udføre optimering ved at simulere en ekstrem forenklet biologisk evolutionsproces. Princippet om den stærkestes overlevelse (survival of the fittest) kan betragtes som en optimering, hvor kriteriet er at opnå den bedst mulige tilpasning til omgivelserne. I en GA genereres en population af individer, som hver svarer til en løsning til et givet optimeringsproblem. Ved anvendelse af rekombination, mutation, etc. genereres nye individer fra eksisterende individer, og efter nogle generationer fremkommer gode individer svarende til gode løsninger til optimeringsproblemet. Kapitel 3 diskuterer anvendelsesområdet for GAs. I en vis forstand gælder det, at jo sværere et problemer, jo mere velegnet er det for en GA. Desuden redegøres for det teoretiske grundlag for GAs og praktisk anvendelse af teorien diskuteres. Den væsentligste del af kapitel 3 er afsnit 3.4, som diskuterer fire centrale problemstillinger vedr. design af GAs: De karakteristiske egenskaber for en god repræsentation af en løsning, udnyttelse af problemspecifik viden, håndtering af begrænsninger og valg af værdier for algoritmens kontrolparametre. Diskussionen af disse fire emner danner grundlag for senere diskussioner i kapitel 5.

Udvalgte værktøjer og algoritmer, der repræsenterer state-of-the-art inden for placering og global routing af makro-celle layouts, præsenteres i kapitel 4. Placerings-værktøjerne er MVP, som er baseret på simu-

VLSI layout syntese. Der tegnes et billede af state-of-the-art inden-for området ved en præsentation af de bedste eksisterende algoritmer og værktøjer. Derefter gives et resumé af de udviklede algoritmer, og deres karakteristika og ydeevne evalueres v.hj.af sammenligninger med state-of-the-art. Endelig giver afhandlingens oversigtsdel nogle forslag til fremtidigt arbejde. Mens artiklerne i appendices forudsætter kendskab til layout syntese samt et vist kendskab til CAD, har det været hensigten med afhandlingens oversigtsdel at den ikke skulle kræve specielle forudsætninger udover en almindelig datalogisk baggrund. Layout syntese introduceres i kapitel 2 og CAD i kapitel 3, og disse kapitler skulle give tilstrækkelige forudsætninger for resten af afhandlingen. Omvendt kan læsere med kendskab til layout syntese undlade at læse kapitel 2 og læsere med kendskab til CAD kan undlade afsnittene 3.1, 3.2 og 3.3.

Oversigtsdelen: Kapitlerne 1 til 7

Afhandlingens første kapitel beskriver bl.a. den valgte tilgangsvinkel til anvendelsesområdet. Problemerne betragtes som kombinatoriske optimeringsproblemer, mens fysiske aspekter som ikke umiddelbart kan gives en ækvivalent geometrisk formulering, ignoreres. Tilgangsvinklen er anvendelsesorienteret, hvilket har mange vigtige konsekvenser, bl.a. for evalueringen af de udviklede algoritmer. I stedet for teoretiske analyser evalueres algoritmernes ydeevne sædvanligvis ved at sammenligne med eksisterende state-of-the-art algoritmers ydeevne på benchmark data. Løsningskvalitet måles relativt til hvad der er opnået med de bedste eksisterende algoritmer, og ved køretider forstås absolutte cpu-tider. Det er en grundlæggende antagelse, at løsningskvalitet prioriteres højere end køretid. I mange praktiske anvendelser vil designeren være villig til at bruge en del ekstra tid for at få en lidt bedre løsning.

Kapitel 2 introducerer layout syntese af VLSI kredsløb. Der findes et antal forskellige layout typer (eng. design styles) repræsenterende forskellige grader af regularitet. Til højtydende kredsløb som f.eks. en CPU anvendes makro-celle layouts, som er den mest fleksible design type og giver mulighed for det største antal transistorer pr. arealenhed og den bedste ydeevne af det producerede kredsløb. Kompleksiteten betyder samtidigt at makro-celle layouts er sværere at generere (manuelt eller automatisk) end nogen anden layout type, hvorfor det største behov for forbedrede værktøjer netop haves for denne layout kategori. Et

Danish Summary (Dansk Resumé)

Denne afhandling er resultatet af et 3-årigt Ph.D. studium udført i perioden fra 1991 til 1994 ved Datalogisk Afdeling, Aarhus Universitet. Lektor Peter Møller-Nielsen har været vejleder på projektet. En del af arbejdet blev udført under et 1-årigt ophold ved University of Michigan, USA, og i denne periode fungerede Associate Professor Prasanta Mazumder som vejleder.

Afhandlingens emne, formål og struktur

Afhandlingens formål er at undersøge mulighederne for at anvende evolutions-baserede algoritmer til løsning af nogle af de delproblemer, der indgår i automatisk generering af layouts af VLSI (Very Large Scale Integration) kredsløb. Kompleksiteten af VLSI kredsløb er steget eksplosivt gennem nogle år, og denne udvikling vurderes at fortsætte. Der er derfor et stort behov for fortsat forskning inden for dette område. I de seneste år har evolutions-baserede algoritmer med held været anvendt til løsning af komplekse optimeringsproblemer inden for mange forskelligartede områder, og det er derfor meningsfuldt at undersøge algoritmernes anvendelighed til layout syntese. Specielt har projektet fokuseret på design af genetiske algoritmer (GAs) til placering og global routing af makro-celle layouts under den grundlæggende antagelse, at designerens primære ønske er at opnå den bedst mulige layout kvalitet. Arbejdets formål er at undersøge hvilken ydeevne der kan opnås med GA-baserede værktøjer til disse problemer samt hvilke algoritmiske designprincipper, der giver de bedste resultater. Desuden ønskes de væsentligste problemer ved denne tilgangsvinkel belyst.

Afhandlingen er baseret på artikler skrevet i løbet af studiet, og den væsentligste del af afhandlingen udgøres af fire udvalgte artikler, gengivet i appendices A-D. Derudover indeholder afhandlingen en oversigt over det udførte arbejde i form af kapitlerne 1 til 7. Heri introduceres emneområdet, dvs. såvel GAs som de betragtede problemer inden for

Contents

Placement and Global Routing of VLSI Macro-Cell Layouts Using Genetic Algorithms

Ph.D. Thesis
by

Henrik Esbensen

Computer Science Department
Aarhus University
DK-8000 Aarhus C, Denmark

July 1994