Bibliography


D.5. Conclusion and Future Work

In this paper a new approach to global rating of macrocell latches based on genetic algorithms has been presented. The performance of the
latch is compared to that of ThruWires. Experimental results show that the quality of completed latches improves when using the GA-based latch instead of ThruWires assuming that the quality of the given placement is sufficiently high. The latch is inferior to ThruWires with respect to runtime but major improvements are possible. Since the work presented here is a first approach to global rating based on genetic algorithms, future improvements of
the latch quality obtainable are also very likely. We conclude that the genetic algorithms are well suited as the basic algorithm of a global router.
final this asumption. The topology of the routing graph of ai\$3-2M is unaltered throughout the process and the performance of the G\$based router is in fact better than that of ThirWMT, as similar results are observed for ai\$3MT. The routing graph topology is significantly altered during the layout process. The placement of ai\$2Ms obtained the same as ai\$3-2M and the performance of the G\$based router is significantly better in this example.

The significant routing graph alterations for squares are a consequence of rather poor initial placements. It is not clear how better placements will affect the relative performance of the two routers. A placement quality increases, the relative effect of eliminating a wire from the longest path in a planar graph increases, indicating a potential advantage for the G\$based router. On the other hand, a good placement contains less routing suggesting that the performance gap could be improved.

For the test examples considered here, most routing demands on the longest paths are captured to their minimum width by the router, cf. the second assumption discussed in Section D.3.1. However, in most cases at least one demand 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 2 to 12 and 30 minutes to route examples based on xerox, ai\$3 and ai\$10, respectively. ThirWMT spends about 30 seconds for examples based on xerox and ai\$3, and about 5 minutes for ai\$based examples. Hence, the G\$based router is clearly inferior to ThirWMT with respect to runtime. The total layout generation process performed by Mii (and i.e., excluding placement) requires about 15 minutes for examples based on xerox and ai\$3, and about an hour for ai\$based examples, when ThirWMT is used. Here, the use of the G\$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 spent creating demand densities. When estimating the area of a solution, all densities are recomputed whether the routing in a demand is actually assigned or not. Keeping track of the next to recompute demand
10(\[\text{result}/\text{Wresult} - 1\]). Here, a negative value indicates a reduction in \textit{pmax} obtained by the GA-based router, while a positive value indicates a percentage increase as compared to TihrWMT. Despite the inherent problem of this kind of calculation, as discussed in Section D4.2, it is clear that in general the GA-based router obtains the best layout quality for the problem instances considered.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
<th>(A_{\text{tot}})</th>
<th>(A_{\text{route}})</th>
<th>(W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>xoroM</td>
<td>best</td>
<td>-19</td>
<td>-4 7</td>
<td>+0 0</td>
</tr>
<tr>
<td></td>
<td>avg</td>
<td>-1 4</td>
<td>-3 5</td>
<td>-0 8</td>
</tr>
<tr>
<td>airoB</td>
<td>best</td>
<td>-3 2</td>
<td>-5 1</td>
<td>-3 2</td>
</tr>
<tr>
<td></td>
<td>avg</td>
<td>4 6</td>
<td>2 5</td>
<td>-0 2</td>
</tr>
<tr>
<td>airoB2</td>
<td>best</td>
<td>-3 0</td>
<td>-4 7</td>
<td>-1 5</td>
</tr>
<tr>
<td></td>
<td>avg</td>
<td>-1 1</td>
<td>-1 7</td>
<td>-0 2</td>
</tr>
<tr>
<td>airoD</td>
<td>best</td>
<td>-1 9</td>
<td>-3 3</td>
<td>-1 5</td>
</tr>
<tr>
<td></td>
<td>avg</td>
<td>0 5</td>
<td>0 8</td>
<td>-0 3</td>
</tr>
<tr>
<td>airoD2</td>
<td>best</td>
<td>-4 2</td>
<td>-7 3</td>
<td>-4 0</td>
</tr>
<tr>
<td></td>
<td>avg</td>
<td>-3 7</td>
<td>-6 3</td>
<td>-2 9</td>
</tr>
</tbody>
</table>

The D2: 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 D3.1. The placement of xoroMs adjusted only slightly during optimization, and the routing graph topology is altered for this example, the GA-based router obtains an average reduction of 35% of the routing area with a cost at the price of a 0.8% increase in total wirelength. However, for airoB, the GA-based router achieves an average reduction in all routes than TihrWMT. In this case, the placement and hence the routing graph topology is significantly affected by the optimizer. 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 contrast this placement, a new placement airoB2 was produced by ripping up all routing in the completed layout of airoB, generated using Tihr-WMT. Since the placement thus obtained is the result of actual completion of all routing it will probably only be subjected to minor adjustments than used itself as input to Macro Experiments on
D4.2 Method

To factors make it difficult to derive 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 heuristically optimizing steps needed to generate a complete layout. Here, when considering a specific result, it may be influenced by a pattern of interactions with other tools, which are typically factors of the routers. Secondly, the optimization strategies used by the two routers are not identical. As described earlier, the Optimized router explicitly attempts to minimize area and secondarily wirelength. While ThrWIM also generates the shortest possible routes in phase one, area is an explicit constraint of the optimization criterion used in the second phase. Instead, ThrWIM selects the shortest possible routes subject to drain capacity constraints.

The chosen strategy for experiments are as follows. For each of the placed examples listed in Table D1, Micro was executed to generate a complete layout, using either ThrWIM or the Optimized router for the global routing task. Here, all other steps of the layout process are performed by the same tools.

Micro was executed five times for each example using the Optimized global router in order to capture the variance caused by the stochastic nature of the applied algorithm. The same set of parameters are used for all parameterizations, i.e., no parameter tuning is performed. For each net, at most \( R = 30 \) alternative routes are generated. The parameters of the Optimized phase are as given in [3]. The phase two GA is executed with population size \( M = 40 \), stopping criteria \( S = 10^2 \), mutation probability \( p = 0.02 \), and crossover probability \( p = 0.10 \). There is no variation of results when applying ThrWIM.

D4.3 Layout Quality

Table D2 summarizes the impact on the completed layouts of using the Optimized router instead of ThrWIM. A

\[
A_{\text{route}} \quad \text{drives routing area, i.e., the part of the total area not occupied by cells and wires. A \text{total area,} \quad td \quad \text{drives total area,}}
\]

Each entry is calculated as
for a ring. A pair of the ring is then selected at random 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 rewording
\[
\pi' = \pi((x + i \mod N) \mod N) = \pi((y - i \mod N) \mod N)
\]
if \( 0 \leq i \leq (y - x) \mod N \), and
\[
\pi' = \pi((x + i \mod N) \mod N)
\]
otherwise; \( i = 0, 1, \ldots, N - 1 \).

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 microcell layout system Mikio, which is a part of the Octo tools CAD framework developed at the University of California, Berkeley. This integration allows for rapid prototyping of the routers performed to that of TrimW [8], a state of the art global router, also interfaced to Mikio.

4.1 Test Examples

Three microcell block sizes, \( \rho_{\rho_{\rho_{\rho}}} \) and \( \rho_{\rho_{\rho_{\rho}}} \), were used for the experiments. However, due to a technical problem, it became necessary to rename all pads from these examples before using them.\(^3\) The microcell block sizes are referenced as \( \rho_{\rho_{\rho_{\rho}}} \).

<table>
<thead>
<tr>
<th>Problem</th>
<th>#cells</th>
<th>#pts</th>
<th>#wts</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \rho_{\rho_{\rho_{\rho}}} ) M</td>
<td>10</td>
<td>33</td>
<td>66</td>
</tr>
<tr>
<td>( \rho_{\rho_{\rho_{\rho}}} ) M</td>
<td>33</td>
<td>5</td>
<td>42</td>
</tr>
<tr>
<td>( \rho_{\rho_{\rho_{\rho}}} ) M</td>
<td>33</td>
<td>5</td>
<td>42</td>
</tr>
<tr>
<td>( \rho_{\rho_{\rho_{\rho}}} ) M</td>
<td>49</td>
<td>30</td>
<td>93</td>
</tr>
<tr>
<td>( \rho_{\rho_{\rho_{\rho}}} ) M</td>
<td>49</td>
<td>30</td>
<td>93</td>
</tr>
</tbody>
</table>

The D4 Problem characteristics.

The D4 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. \( \rho_{\rho_{\rho_{\rho}}} \), \( \rho_{\rho_{\rho_{\rho}}} \), \( \rho_{\rho_{\rho_{\rho}}} \) and \( \rho_{\rho_{\rho_{\rho}}} \) were placed by Hopf, a planar tool, based on simulated annealing also included.

\(^2\) The definition of \( \pi' \) relies on the mathematical definition of modulo, in which the remainder is always non-negative.

\(^3\) In our version of Octo tools (5.2) the channel definition program Atlas cannot handle the pad placement generated by Padplace.
Te population \( P = \{ p_0, p_1, \ldots, p_M \} \) is thensorted lexicographically using area as a significant criterion and wavelength as a secondary criterion. Assume that \( P \) is sorted in decreasing order with respect to this ordering. Then \( F \) of \( p_0 \ldots p_i \) is thensorted as \( F(p_0 \ldots p_i) = 2^i / (M - 1) \) for \( i = 0, 1, \ldots, M - 1 \). This scheme, called ranking, assures constant variance of fitness throughout the optimization process. Biking 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 \( \alpha \) and \( \beta \), the crossover operator generates two offspring \( \phi \) and \( \psi \). The parent individuals are not altered by the operator. In the following a (secret) subscript specifies which individual the masked property is a part of. Crossover consists of two steps:

1. One of the parents, say \( \beta \), is chosen at random and a copy \( \gamma \) of \( \beta \) is made. \( \gamma \) is then replaced so that it becomes indistinguishable from \( \alpha \), that is, \( \pi_\gamma = \pi_\alpha \).

2. The offspring are generated according to their parents: \( \pi \)

\[ \phi = \pi_\alpha \]
\[ \psi = \pi_\gamma \]

A standard one-point crossover is then performed. A crossover point \( x \) is selected at random \( \{0, 1, \ldots, N - 2\} \). The selected rate of \( \phi \) is then defined by \( q = \pi(k)_\alpha, \phi = q \pi(k)_\alpha \) if \( k \leq x \) and \( q = \pi(k)_\gamma \) otherwise, where \( \pi = \pi_\alpha \). Similarly, the selected rate of \( \psi \) is defined by \( q = \pi(k)_\gamma, \psi = q \pi(k)_\alpha \) if \( k \leq x \) and \( q = \pi(k)_\alpha \) 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 rate for the \( k \) th bit with probability \( p \), and \( r_k = 1 \), where \( p = 1 \) is a user-defined probability. This scheme is called pointwise mutation.

A mutation in Section D.3.2.1 a given global rating solution can be represented by several equivalent individuals because of the independence of the ordering \( \pi \). However, the fitness of offspring produced by crossover depends on the specific ordering of the given parent individuals. The purpose of inversion is to optimize the performance of the crossover operator. With a given probability \( p \), the inversion operator alters the ordering \( \pi \) of a given individual. To obtain a uniform probability of renewal of all tuples, we consider the set of tuples to
D3. Representations

A genetic solution is represented by specifying for each individual which of the possible rates is used. More specifically, assume a bijection

\[ 0, 1, \ldots, N - 1 \]

of the nets, let \( \pi : \{ 0, 1, \ldots, N - 1 \} \mapsto \{ 0, 1, \ldots, N - 1 \} \) be a bijection and define by \( r_k \leq R \) the number of rates generated in phase one for the \( k \)th net. An individual is then a set of \( N \) tuples

\[ \{(\pi(0), q_0), (\pi(1), q_1), \ldots, (\pi(N - 1), q_{N - 1})\} \]

where \( 1 \leq q_k \leq r_k \) for all \( k = 0, 1, \ldots, N - 1 \). For example, the tuple (37) specifies that the 3rd net uses its 7th rate. Terminating \( \pi \) defines an ordering of the nets, the purpose of which is explained in Section D3.2.4. Note that the genetic solution specified in each individual is lexicographic order.

D3.2.2 Definition of Fitness

Given a population \( P \), the routine `evaluate` in 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 wavelength is computed as the sum of the lengths of the rates 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 operators of this process are crossover, which mimics reproduction, 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. An introduction to GA is given in [4].

```plaintext
initialize \( P \cup c \);
evaluate\( P \cup c \);
\( s = \text{best}(P \cup c) \);
repeat until stop criterion:
\( P_N = \emptyset \);
repeat \( M/2 \) times:
select \( p_1 \in P \cup c, p_2 \in P \cup c \);
crossover\( p_1, p_2, q, s \);
\( P_N = P \cup c \cup \{ q, s \} \);
and
evaluate\( P \cup c \cup P_N \);
\( P_c = \text{reduce}(P \cup c \cup P_N) \);
\( \forall p \in P_c: \text{possibly mutate}(p) \);
\( \forall p \in P: \text{possibly mutate}(p) \);
evaluate\( P \cup c \);
\( s = \text{best}(P \cup c \cup \{ s \}) \);
and
optimize\( s \);
return \( s \);
```

Figure D5 Outline of phase two.

Fig. D5 outlines the phase two algorithm. Initially the current population \( P \cup c \) of size \( M = |P \cup c| \) consists of \( M - 1 \) randomly generated individuals and a single individual consisting of the shortest route found for each net. Seeing the population with this relatively good solution does not lead to better final results, but merely speeds up the search process.
of the longest path in $H$. Then estimates the horizontal length of the layout. By constructing $W_G$ in a similar way, the area is estimated as the product of the longest path in $H$ times the longest path in $W_G$.

In [7], the cost of an edge in the polar graph 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 nets layers are available for routing and that each layer is used exclusively for routing in a specific direction. Therefore, to obtain an 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 graph is then proportional to the density of the corresponding channel.

Several factors affect the accuracy of the area estimate. The two most important have to do with the subsequent compaction/sizing of the layout:

1) If the compactor alters the placement of the topological routing graph, changes the polar graph, and changes the channel density, 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 tradeoff between compaction and global routing tasks.

2) It is implicitly assumed that the compactor generates a layout in which the routing channel has the length of the polar graph. If this is not the case, 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 number, while the less fit individuals tend to die out. This survival of the fittest Darwinian principle is the basic idea behind the GA
D 3 Phase Two of the Router

The area estimate is crucial to the phase two algorithm as discussed in Section D3.1. A detailed description of the Genetic Algorithm (GA) for finding the optimal path is presented in Section D3.2.

D3.1 Area Estimation

As in [7, 11], the area estimation is based on the concept of polar graphs as illustrated in Fig. D4. For a given placement and routing graph, two polar graphs are constructed: a horizontal (HP) and a vertical (VP). Let us start by considering HP. The vertices of HP consist of a vertex for each cell plus two additional vertices, a source and a sink. Each edge in HP corresponds to a vertical edge in the routing graph and is directed from the source toward the sink.

![Polar graphs for area estimation.](image)

Assume that each edge (v, w) has a cost which corresponds to the spring model 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].
D2.2. Two-terminal nets

For each net with two terminals, an algorithm to Laker [6] is used to compute the shortest, second shortest, third shortest, etc. rate until a minimum R rates are found or no rate exists. Laker's algorithm is exact but also quite expensive, requiring time $O(Rn)$ per net, where $n$ is the number of vertices in the rating graph.

An earlier algorithm by Yehia [1] may at first seem more attractive. It generates the $R$ shortest rates 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 Laker's algorithm and if two paths do not visit the same vertices in the same order they are considered distinct. We could then simply generate rates until $R$ loopless rates were obtained and 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 rates then required.

D2.2. Nets with at least three terminals

A net $R$ distinct rates are generated for each net having three or more terminals using a GA for the SG. For a detailed description of the algorithm used, readers are referred to [2, 3]. There are two main advantages of using the 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% of the global optimum solution in more than 92% of all runs. Te rating graph of an actual plant with $C'$ cells will have less than $3C'$ vertices. It is therefore not likely that the GA will find the shortest existing rate for every net in any reasonably sized real-world layout. The second advantage of the GA is that it provides a number of distinct solutions in a single run. The problem of Mary and Tiber [WM] that only one rate is generated for nets with many terminals is thus eliminated.

For nets with few terminals, say 67 or less, exhaustive search for the shortest rate will often be feasible. Using an algorithm by Sillivan [9], the optimum rate can be found by exhausting a search space consisting of

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

paths, where $k = \text{int}(t - 2, n)$ and $t$ is the number of terminals of the net. However, experiments have revealed that Sillivan's algorithm often
Fig D2. Addition of terminal vertices (shaded) for a net with three terminals (marked with crosses).

Fig D3 outlines phase one. A net is trivial if all its terminals are projected onto the same edge of the rating graph. Although several rates can still be generated for a trivial net, it will rarely be advantageous. Hence, global rating is skipped for such nets.

```
generate rating graph
for each trivial net do:
    add vertices to graph
    if 2terminal net:
        apply Euler algorithm
    else
        apply a G for SIG
    remove vertices from graph
end
```

Fig D3 Outline of phase one.

The SIGs in general. However, if only two vertices are to be connected, SIGs reduce to a shortest path problem which is handled by an algorithm of Euler discussed in Section D21. Nts with more than two terminals are handled by a G discussed in Section D22.
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. D1.

![Routing Graph Example](image)

Fig. D1: 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 to finding an innermost 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 (SP). 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 SP 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. D2. 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 results in inscrutable sets of terminal vertices, in which case some computations can be avoided. On the other hand, this scheme provides a more accurate estimate of the wirelength and also allows a more accurate area estimate as discussed in Section D3.1.
D 1 Introduction

A well-known strategy for global routing of macrocell layouts consists of two phases [10]. In the first phase, a number of tentative 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 design capacity constraints, and so that some overall criteria, typically area or total interconnect length, is minimized. An advantage of this routing strategy is its independence of net ordering.

Mary [7] and TitanWM [8] are state of the art global routers for macrocell 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 tentative routes for each net. However, due to the time complexity of the applied algorithm, 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 secondary total interconnect length. We 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 genetic phase produces several high quality routes for each net independently of its number of terminals. In the second phase, another GA minimizes the total interconnect length 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 capture and yield densities and the wire length estimate is based on pin locations.

Experimental results show that the layout quality obtained by the router compares favorably to that of TitanWM.

D 2 Phase One of the Router

Before the global routing process itself is initiated a rectilinear \textit{routing graph} is extracted from the given layout. Routing is then performed internally on this graph, i.e., creating a global route for a net is done by
Appendix D

AMacro-Cell Global Router Based on Two Genetic Algorithms

This paper is published as H. Ihnken, "AMacro-Cell Global Router Based on Two Genetic Algorithms" Proc. of The European Design Automation Conference, pp. 4848, 1994.

Abstract

This paper presents an 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 signal interconnect length. Fast demand densities are used for area estimation. The layout quality obtained on NEC benchmarks is comparable to that of Tilt/WMC.


Bibliography


### Table C2: Comparison of solution quality and CPU-time for the graphs in class \( E \)

<table>
<thead>
<tr>
<th>Graph</th>
<th>Cost ( C_{opt} )</th>
<th>Cost ( C_{spb} )</th>
<th>Cost ( \Delta C_{spb} )</th>
<th>Cost ( C_{ga} )</th>
<th>Cost ( \Delta C_{ga} )</th>
<th>CPU-time (secs) ( T_{bc-1} )</th>
<th>CPU-time (secs) ( T_{bc-2} )</th>
<th>CPU-time (secs) ( T_{spb} )</th>
<th>CPU-time (secs) ( T_{ga} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>E-1</td>
<td>111</td>
<td>111</td>
<td>0</td>
<td>111</td>
<td>0</td>
<td>1,150</td>
<td>1,394</td>
<td>7,334</td>
<td>7,395</td>
</tr>
<tr>
<td>E2</td>
<td>214</td>
<td>216</td>
<td>0.93</td>
<td>216</td>
<td>0.93</td>
<td>6,251</td>
<td>1,993</td>
<td>7,355</td>
<td>7,444</td>
</tr>
<tr>
<td>E3</td>
<td>4,013</td>
<td>4,060</td>
<td>1.17</td>
<td>4,013</td>
<td>0</td>
<td>26,468</td>
<td>15,782</td>
<td>50,004</td>
<td>9,449</td>
</tr>
<tr>
<td>E4</td>
<td>5,101</td>
<td>5,113</td>
<td>0.24</td>
<td>5,102</td>
<td>0.02</td>
<td>46,008</td>
<td>1,660</td>
<td>29,921</td>
<td>7,763</td>
</tr>
<tr>
<td>E5</td>
<td>8,128</td>
<td>8,134</td>
<td>0.07</td>
<td>8,128</td>
<td>0</td>
<td>12,564</td>
<td>411</td>
<td>9,318</td>
<td>7,474</td>
</tr>
<tr>
<td>E6</td>
<td>73</td>
<td>73</td>
<td>4.11</td>
<td>73</td>
<td>0</td>
<td>678</td>
<td>-</td>
<td>10,060</td>
<td>10,148</td>
</tr>
<tr>
<td>E7</td>
<td>145</td>
<td>149</td>
<td>2.76</td>
<td>145</td>
<td>0</td>
<td>27,124</td>
<td>-</td>
<td>10,306</td>
<td>10,458</td>
</tr>
<tr>
<td>E8</td>
<td>2,640</td>
<td>2,690</td>
<td>1.89</td>
<td>2,646</td>
<td>0.23</td>
<td>118,618</td>
<td>-</td>
<td>50,013</td>
<td>12,896</td>
</tr>
<tr>
<td>E9</td>
<td>3,604</td>
<td>3,671</td>
<td>1.86</td>
<td>3,611</td>
<td>0.19</td>
<td>24,528</td>
<td>-</td>
<td>50,014</td>
<td>14,933</td>
</tr>
<tr>
<td>E10</td>
<td>5,600</td>
<td>5,624</td>
<td>0.43</td>
<td>5,600</td>
<td>0</td>
<td>39,261</td>
<td>-</td>
<td>50,014</td>
<td>12,976</td>
</tr>
<tr>
<td>E11</td>
<td>34</td>
<td>34</td>
<td>0</td>
<td>34</td>
<td>0</td>
<td>1,901</td>
<td>-</td>
<td>14,472</td>
<td>14,559</td>
</tr>
<tr>
<td>E12</td>
<td>67</td>
<td>68</td>
<td>1.49</td>
<td>68</td>
<td>1.49</td>
<td>7,200</td>
<td>-</td>
<td>14,497</td>
<td>14,588</td>
</tr>
<tr>
<td>E13</td>
<td>1,250</td>
<td>1,317</td>
<td>2.89</td>
<td>1,289</td>
<td>0.70</td>
<td>207,059</td>
<td>-</td>
<td>50,003</td>
<td>21,787</td>
</tr>
<tr>
<td>E14</td>
<td>1,732</td>
<td>1,767</td>
<td>2.02</td>
<td>1,736</td>
<td>0.23</td>
<td>29,263</td>
<td>-</td>
<td>50,030</td>
<td>23,022</td>
</tr>
<tr>
<td>E15</td>
<td>2,784</td>
<td>2,795</td>
<td>0.40</td>
<td>2,784</td>
<td>0</td>
<td>7,666</td>
<td>-</td>
<td>50,020</td>
<td>18,424</td>
</tr>
<tr>
<td>E16</td>
<td>15</td>
<td>15</td>
<td>0</td>
<td>15</td>
<td>0</td>
<td>179</td>
<td>-</td>
<td>14,425</td>
<td>14,586</td>
</tr>
<tr>
<td>E17</td>
<td>25</td>
<td>25</td>
<td>0</td>
<td>25</td>
<td>0</td>
<td>36,040</td>
<td>-</td>
<td>14,458</td>
<td>14,619</td>
</tr>
<tr>
<td>E18</td>
<td>572</td>
<td>625</td>
<td>9.27</td>
<td>583</td>
<td>1.92</td>
<td>-</td>
<td>-</td>
<td>50,017</td>
<td>29,105</td>
</tr>
<tr>
<td>E19</td>
<td>758</td>
<td>802</td>
<td>5.80</td>
<td>766</td>
<td>1.06</td>
<td>6,372</td>
<td>-</td>
<td>50,037</td>
<td>27,319</td>
</tr>
<tr>
<td>E20</td>
<td>1,342</td>
<td>1,357</td>
<td>1.12</td>
<td>1,342</td>
<td>0</td>
<td>272</td>
<td>-</td>
<td>50,055</td>
<td>25,107</td>
</tr>
</tbody>
</table>
### Table C.10: Comparison of CPU time (seconds) for the graphs in class D.

<table>
<thead>
<tr>
<th>Graph</th>
<th>$T_{bc1}$</th>
<th>$T_{bc2}$</th>
<th>$T_{ sph }$</th>
<th>$T_{ avg }$</th>
<th>$T_{\sigma}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>476</td>
<td>200</td>
<td>486</td>
<td>523</td>
<td>8</td>
</tr>
<tr>
<td>D2</td>
<td>284</td>
<td>148</td>
<td>488</td>
<td>537</td>
<td>13</td>
</tr>
<tr>
<td>D3</td>
<td>2,290</td>
<td>106</td>
<td>785</td>
<td>650</td>
<td>39</td>
</tr>
<tr>
<td>D4</td>
<td>3,529</td>
<td>41</td>
<td>689</td>
<td>554</td>
<td>21</td>
</tr>
<tr>
<td>D5</td>
<td>811</td>
<td>37</td>
<td>522</td>
<td>504</td>
<td>8</td>
</tr>
<tr>
<td>D6</td>
<td>2,340</td>
<td>4,148</td>
<td>687</td>
<td>788</td>
<td>44</td>
</tr>
<tr>
<td>D7</td>
<td>100</td>
<td>1,037</td>
<td>681</td>
<td>795</td>
<td>29</td>
</tr>
<tr>
<td>D8</td>
<td>6,985</td>
<td>17,858</td>
<td>13,237</td>
<td>2,101</td>
<td>381</td>
</tr>
<tr>
<td>D9</td>
<td>4,630</td>
<td>16,458</td>
<td>29,354</td>
<td>2,744</td>
<td>624</td>
</tr>
<tr>
<td>D10</td>
<td>1,312</td>
<td>1,678</td>
<td>14,780</td>
<td>1,100</td>
<td>169</td>
</tr>
<tr>
<td>D11</td>
<td>1,374</td>
<td>24,609</td>
<td>949</td>
<td>1,070</td>
<td>59</td>
</tr>
<tr>
<td>D12</td>
<td>305</td>
<td>5,843</td>
<td>961</td>
<td>1,085</td>
<td>20</td>
</tr>
<tr>
<td>D13</td>
<td>1,864</td>
<td>91,718</td>
<td>15,187</td>
<td>2,357</td>
<td>245</td>
</tr>
<tr>
<td>D14</td>
<td>3,538</td>
<td>61,335</td>
<td>41,237</td>
<td>2,601</td>
<td>393</td>
</tr>
<tr>
<td>D15</td>
<td>1,410</td>
<td>16,889</td>
<td>24,828</td>
<td>1,302</td>
<td>102</td>
</tr>
<tr>
<td>D16</td>
<td>871</td>
<td>9,721</td>
<td>956</td>
<td>1,047</td>
<td>21</td>
</tr>
<tr>
<td>D17</td>
<td>6,965</td>
<td>147,598</td>
<td>950</td>
<td>1,068</td>
<td>26</td>
</tr>
<tr>
<td>D18</td>
<td>245,192</td>
<td>227,841</td>
<td>31,015</td>
<td>2,536</td>
<td>491</td>
</tr>
<tr>
<td>D19</td>
<td>878</td>
<td>304,380</td>
<td>50,003</td>
<td>3,441</td>
<td>580</td>
</tr>
<tr>
<td>D20</td>
<td>47</td>
<td>1,276</td>
<td>50,010</td>
<td>2,638</td>
<td>658</td>
</tr>
</tbody>
</table>

### Table C.11: The class E graphs before and after reductions.

<table>
<thead>
<tr>
<th>Graph</th>
<th>Problem size</th>
<th>Reduced size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>n</td>
<td>m</td>
</tr>
<tr>
<td>E1</td>
<td>2,500</td>
<td>5</td>
</tr>
<tr>
<td>E2</td>
<td>2,500</td>
<td>10</td>
</tr>
<tr>
<td>E3</td>
<td>2,500</td>
<td>417</td>
</tr>
<tr>
<td>E4</td>
<td>2,500</td>
<td>625</td>
</tr>
<tr>
<td>E5</td>
<td>2,500</td>
<td>1,250</td>
</tr>
<tr>
<td>E6</td>
<td>2,500</td>
<td>5</td>
</tr>
<tr>
<td>E7</td>
<td>2,500</td>
<td>10</td>
</tr>
<tr>
<td>E8</td>
<td>2,500</td>
<td>417</td>
</tr>
<tr>
<td>E9</td>
<td>2,500</td>
<td>625</td>
</tr>
<tr>
<td>E10</td>
<td>2,500</td>
<td>1,250</td>
</tr>
<tr>
<td>E11</td>
<td>2,500</td>
<td>5</td>
</tr>
<tr>
<td>E12</td>
<td>2,500</td>
<td>10</td>
</tr>
<tr>
<td>E13</td>
<td>2,500</td>
<td>417</td>
</tr>
<tr>
<td>E14</td>
<td>2,500</td>
<td>625</td>
</tr>
<tr>
<td>E15</td>
<td>2,500</td>
<td>1,250</td>
</tr>
<tr>
<td>E16</td>
<td>2,500</td>
<td>5</td>
</tr>
<tr>
<td>E17</td>
<td>2,500</td>
<td>10</td>
</tr>
<tr>
<td>E18</td>
<td>2,500</td>
<td>417</td>
</tr>
<tr>
<td>E19</td>
<td>2,500</td>
<td>625</td>
</tr>
<tr>
<td>E20</td>
<td>2,500</td>
<td>1,250</td>
</tr>
</tbody>
</table>
The class $D$ graphs before and after reductions.

<table>
<thead>
<tr>
<th>Graph</th>
<th>$C_{opt}$</th>
<th>$C_{sph}$</th>
<th>$\Delta C_{sph}$</th>
<th>$C_{best}$</th>
<th>$C_{avg}$</th>
<th>$C_{worst}$</th>
<th>$C_{o}$</th>
<th>$\Delta C_{avg}$</th>
<th>$\Delta C_{worst}$</th>
<th>$N_{ga}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>106</td>
<td>106</td>
<td>0</td>
<td>106</td>
<td>106</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D2</td>
<td>220</td>
<td>220</td>
<td>0</td>
<td>220</td>
<td>220</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D3</td>
<td>1,565</td>
<td>1,570</td>
<td>0.32</td>
<td>1,565</td>
<td>1,565</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D4</td>
<td>1,935</td>
<td>1,940</td>
<td>0.26</td>
<td>1,935</td>
<td>1,935</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D5</td>
<td>3,250</td>
<td>3,254</td>
<td>0.12</td>
<td>3,250</td>
<td>3,250</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D6</td>
<td>67</td>
<td>71</td>
<td>5.97</td>
<td>67</td>
<td>67.1</td>
<td>0.3</td>
<td>0.15</td>
<td>1.49</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>D7</td>
<td>103</td>
<td>103</td>
<td>0</td>
<td>103</td>
<td>103</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D8</td>
<td>1,072</td>
<td>1,095</td>
<td>0.25</td>
<td>1,072</td>
<td>1,072</td>
<td>0.6</td>
<td>0.07</td>
<td>0.19</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>D9</td>
<td>1,448</td>
<td>1,471</td>
<td>0.59</td>
<td>1,448</td>
<td>1,448.4</td>
<td>0.7</td>
<td>0.3</td>
<td>0.14</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>D10</td>
<td>2,110</td>
<td>2,120</td>
<td>0.47</td>
<td>2,110</td>
<td>2,110</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D11</td>
<td>29</td>
<td>29</td>
<td>0</td>
<td>29</td>
<td>29</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D12</td>
<td>42</td>
<td>42</td>
<td>0</td>
<td>42</td>
<td>42</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D13</td>
<td>500</td>
<td>514</td>
<td>2.80</td>
<td>500</td>
<td>500.6</td>
<td>0.7</td>
<td>0.12</td>
<td>0.40</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>D14</td>
<td>667</td>
<td>675</td>
<td>1.20</td>
<td>668</td>
<td>669.7</td>
<td>0.9</td>
<td>0.40</td>
<td>0.60</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>D15</td>
<td>1,116</td>
<td>1,121</td>
<td>0.45</td>
<td>1,116</td>
<td>1,116</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D16</td>
<td>13</td>
<td>13</td>
<td>0</td>
<td>13</td>
<td>13</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D17</td>
<td>23</td>
<td>23</td>
<td>0</td>
<td>23</td>
<td>23</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>D18</td>
<td>223</td>
<td>239</td>
<td>7.17</td>
<td>226</td>
<td>227.7</td>
<td>1.2</td>
<td>2.11</td>
<td>3.14</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>D19</td>
<td>310</td>
<td>335</td>
<td>8.06</td>
<td>312</td>
<td>313.3</td>
<td>0.9</td>
<td>1.06</td>
<td>1.61</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>D20</td>
<td>537</td>
<td>539</td>
<td>0.37</td>
<td>537</td>
<td>537</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Graph</td>
<td>$C_{opt}$</td>
<td>$C_{sph}$</td>
<td>$S_{sph}$</td>
<td>$C_{best}$</td>
<td>$C_{avg}$</td>
<td>$C_{worst}$</td>
<td>$C_{\sigma}$</td>
<td>$S_{avg}$</td>
<td>$S_{worst}$</td>
<td>$N_{ga}$</td>
</tr>
<tr>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>-------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
<td>--------</td>
</tr>
<tr>
<td>C 1</td>
<td>85</td>
<td>85</td>
<td>0</td>
<td>85</td>
<td>85</td>
<td>85</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C 2</td>
<td>144</td>
<td>144</td>
<td>0</td>
<td>144</td>
<td>144</td>
<td>144</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C 3</td>
<td>754</td>
<td>754</td>
<td>0.4</td>
<td>754</td>
<td>754.2</td>
<td>755</td>
<td>0.4</td>
<td>0.03</td>
<td>0.13</td>
<td>2</td>
</tr>
<tr>
<td>C 4</td>
<td>1,079</td>
<td>1,081</td>
<td>0.19</td>
<td>1,079</td>
<td>1,079.1</td>
<td>1,080</td>
<td>0.3</td>
<td>0.01</td>
<td>0.09</td>
<td>1</td>
</tr>
<tr>
<td>C 5</td>
<td>1,579</td>
<td>1,579</td>
<td>0</td>
<td>1,579</td>
<td>1,579</td>
<td>1,579</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C 6</td>
<td>55</td>
<td>55</td>
<td>0</td>
<td>55</td>
<td>55</td>
<td>55</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C 7</td>
<td>102</td>
<td>102</td>
<td>0</td>
<td>102</td>
<td>102</td>
<td>102</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C 8</td>
<td>509</td>
<td>512</td>
<td>0.59</td>
<td>509</td>
<td>509</td>
<td>509</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C 9</td>
<td>707</td>
<td>714</td>
<td>0.99</td>
<td>707</td>
<td>707.4</td>
<td>708</td>
<td>0.5</td>
<td>0.06</td>
<td>0.14</td>
<td>4</td>
</tr>
<tr>
<td>C 10</td>
<td>1,093</td>
<td>1,098</td>
<td>0.46</td>
<td>1,093</td>
<td>1,093</td>
<td>1,093</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C 11</td>
<td>32</td>
<td>32</td>
<td>0</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C 12</td>
<td>46</td>
<td>46</td>
<td>0</td>
<td>46</td>
<td>46</td>
<td>46</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C 13</td>
<td>258</td>
<td>263</td>
<td>1.94</td>
<td>258</td>
<td>259.7</td>
<td>260</td>
<td>0.6</td>
<td>0.66</td>
<td>0.78</td>
<td>9</td>
</tr>
<tr>
<td>C 14</td>
<td>323</td>
<td>327</td>
<td>1.24</td>
<td>323</td>
<td>323.4</td>
<td>324</td>
<td>0.5</td>
<td>0.12</td>
<td>0.31</td>
<td>4</td>
</tr>
<tr>
<td>C 15</td>
<td>556</td>
<td>558</td>
<td>0.36</td>
<td>556</td>
<td>556</td>
<td>556</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C 16</td>
<td>11</td>
<td>11</td>
<td>0</td>
<td>11</td>
<td>11.7</td>
<td>12</td>
<td>0.5</td>
<td>6.36</td>
<td>9.09</td>
<td>7</td>
</tr>
<tr>
<td>C 17</td>
<td>18</td>
<td>18</td>
<td>0</td>
<td>18</td>
<td>18</td>
<td>18</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>C 18</td>
<td>113</td>
<td>121</td>
<td>7.08</td>
<td>113</td>
<td>114.3</td>
<td>115</td>
<td>0.8</td>
<td>1.15</td>
<td>1.77</td>
<td>8</td>
</tr>
<tr>
<td>C 19</td>
<td>146</td>
<td>155</td>
<td>6.16</td>
<td>146</td>
<td>147</td>
<td>148</td>
<td>0.4</td>
<td>0.68</td>
<td>1.37</td>
<td>9</td>
</tr>
<tr>
<td>C 20</td>
<td>267</td>
<td>267</td>
<td>0</td>
<td>267</td>
<td>267</td>
<td>267</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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

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

**Table C7:** Comparison of CPU-time (seconds) for the graphs in class C.
### Table C4

Comparison of CPU time (seconds) for the graphs in class B.

<table>
<thead>
<tr>
<th>Graph</th>
<th>$T_{br}$</th>
<th>$T_{ph}$</th>
<th>$T_{avg}$</th>
<th>$T_\sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>B-1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.0</td>
</tr>
<tr>
<td>B2</td>
<td>0.1</td>
<td>0.1</td>
<td>0.2</td>
<td>0.0</td>
</tr>
<tr>
<td>B3</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.0</td>
</tr>
<tr>
<td>B4</td>
<td>0.6</td>
<td>0.1</td>
<td>1.2</td>
<td>0.6</td>
</tr>
<tr>
<td>B5</td>
<td>1.9</td>
<td>0.1</td>
<td>0.7</td>
<td>0.2</td>
</tr>
<tr>
<td>B6</td>
<td>0.6</td>
<td>0.1</td>
<td>0.7</td>
<td>0.1</td>
</tr>
<tr>
<td>B7</td>
<td>0.2</td>
<td>0.2</td>
<td>0.5</td>
<td>0.1</td>
</tr>
<tr>
<td>B8</td>
<td>0.1</td>
<td>0.2</td>
<td>0.5</td>
<td>0.1</td>
</tr>
<tr>
<td>B9</td>
<td>0.1</td>
<td>0.2</td>
<td>0.2</td>
<td>0.0</td>
</tr>
<tr>
<td>B10</td>
<td>3.1</td>
<td>0.3</td>
<td>1.7</td>
<td>0.5</td>
</tr>
<tr>
<td>B11</td>
<td>1.4</td>
<td>0.3</td>
<td>1.4</td>
<td>0.6</td>
</tr>
<tr>
<td>B12</td>
<td>0.6</td>
<td>0.3</td>
<td>0.6</td>
<td>0.1</td>
</tr>
<tr>
<td>B13</td>
<td>0.7</td>
<td>0.4</td>
<td>1.4</td>
<td>0.4</td>
</tr>
<tr>
<td>B14</td>
<td>1.2</td>
<td>0.5</td>
<td>0.9</td>
<td>0.3</td>
</tr>
<tr>
<td>B15</td>
<td>0.3</td>
<td>0.5</td>
<td>0.8</td>
<td>0.1</td>
</tr>
<tr>
<td>B16</td>
<td>18.4</td>
<td>0.6</td>
<td>4.4</td>
<td>1.9</td>
</tr>
<tr>
<td>B17</td>
<td>3.3</td>
<td>0.6</td>
<td>2.3</td>
<td>0.6</td>
</tr>
<tr>
<td>B18</td>
<td>1.0</td>
<td>0.6</td>
<td>1.5</td>
<td>0.3</td>
</tr>
</tbody>
</table>

### Table C5

The class C graphs before and after reductions.

<table>
<thead>
<tr>
<th>Graph</th>
<th>Problem size</th>
<th>Reduced size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>n  m</td>
<td>$</td>
</tr>
<tr>
<td>C1</td>
<td>500  5</td>
<td>625</td>
</tr>
<tr>
<td>C2</td>
<td>500  10</td>
<td>625</td>
</tr>
<tr>
<td>C3</td>
<td>500  83</td>
<td>625</td>
</tr>
<tr>
<td>C4</td>
<td>500  125</td>
<td>625</td>
</tr>
<tr>
<td>C5</td>
<td>500  250</td>
<td>625</td>
</tr>
<tr>
<td>C6</td>
<td>500  5</td>
<td>1,000</td>
</tr>
<tr>
<td>C7</td>
<td>500  10</td>
<td>1,000</td>
</tr>
<tr>
<td>C8</td>
<td>500  83</td>
<td>1,000</td>
</tr>
<tr>
<td>C9</td>
<td>500  125</td>
<td>1,000</td>
</tr>
<tr>
<td>C10</td>
<td>500  250</td>
<td>1,000</td>
</tr>
<tr>
<td>C11</td>
<td>500  5</td>
<td>2,500</td>
</tr>
<tr>
<td>C12</td>
<td>500  10</td>
<td>2,500</td>
</tr>
<tr>
<td>C13</td>
<td>500  83</td>
<td>2,500</td>
</tr>
<tr>
<td>C14</td>
<td>500  125</td>
<td>2,500</td>
</tr>
<tr>
<td>C15</td>
<td>500  250</td>
<td>2,500</td>
</tr>
<tr>
<td>C16</td>
<td>500  5</td>
<td>12,500</td>
</tr>
<tr>
<td>C17</td>
<td>500  10</td>
<td>12,500</td>
</tr>
<tr>
<td>C18</td>
<td>500  83</td>
<td>12,500</td>
</tr>
<tr>
<td>C19</td>
<td>500  125</td>
<td>12,500</td>
</tr>
<tr>
<td>C20</td>
<td>500  250</td>
<td>12,500</td>
</tr>
</tbody>
</table>
C 7 Computational Results

<table>
<thead>
<tr>
<th>Graph</th>
<th>Problem size</th>
<th>Reduced size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>n</td>
<td>m</td>
</tr>
<tr>
<td>B 1</td>
<td>50</td>
<td>9</td>
</tr>
<tr>
<td>B 2</td>
<td>50</td>
<td>13</td>
</tr>
<tr>
<td>B 3</td>
<td>50</td>
<td>25</td>
</tr>
<tr>
<td>B 4</td>
<td>50</td>
<td>9</td>
</tr>
<tr>
<td>B 5</td>
<td>50</td>
<td>13</td>
</tr>
<tr>
<td>B 6</td>
<td>50</td>
<td>25</td>
</tr>
<tr>
<td>B 7</td>
<td>75</td>
<td>13</td>
</tr>
<tr>
<td>B 8</td>
<td>75</td>
<td>19</td>
</tr>
<tr>
<td>B 9</td>
<td>75</td>
<td>38</td>
</tr>
<tr>
<td>B 10</td>
<td>75</td>
<td>13</td>
</tr>
<tr>
<td>B 11</td>
<td>75</td>
<td>19</td>
</tr>
<tr>
<td>B 12</td>
<td>75</td>
<td>38</td>
</tr>
<tr>
<td>B 13</td>
<td>100</td>
<td>17</td>
</tr>
<tr>
<td>B 14</td>
<td>100</td>
<td>25</td>
</tr>
<tr>
<td>B 15</td>
<td>100</td>
<td>50</td>
</tr>
<tr>
<td>B 16</td>
<td>100</td>
<td>17</td>
</tr>
<tr>
<td>B 17</td>
<td>100</td>
<td>25</td>
</tr>
<tr>
<td>B 18</td>
<td>100</td>
<td>50</td>
</tr>
</tbody>
</table>

Table C2 The class B graphs before and after reductions.

<table>
<thead>
<tr>
<th>Graph</th>
<th>C_{opt}</th>
<th>C_{sph}</th>
<th>ΔC_{sph}</th>
<th>C_{best}</th>
<th>C_{avg}</th>
<th>C_{worst}</th>
<th>C_{σ}</th>
<th>ΔC_{avg}</th>
<th>ΔC_{worst}</th>
<th>N_{ga}</th>
</tr>
</thead>
<tbody>
<tr>
<td>B 1</td>
<td>82</td>
<td>82</td>
<td>0</td>
<td>82</td>
<td>82</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B 2</td>
<td>83</td>
<td>83</td>
<td>0</td>
<td>83</td>
<td>83</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B 3</td>
<td>138</td>
<td>138</td>
<td>0</td>
<td>138</td>
<td>138</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B 4</td>
<td>50</td>
<td>50</td>
<td>0</td>
<td>50</td>
<td>50</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B 5</td>
<td>61</td>
<td>61</td>
<td>0</td>
<td>61</td>
<td>61</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B 6</td>
<td>122</td>
<td>122</td>
<td>0</td>
<td>122</td>
<td>122</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B 7</td>
<td>111</td>
<td>111</td>
<td>0</td>
<td>111</td>
<td>111</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B 8</td>
<td>104</td>
<td>104</td>
<td>0</td>
<td>104</td>
<td>104</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B 9</td>
<td>220</td>
<td>220</td>
<td>0</td>
<td>220</td>
<td>220</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B 10</td>
<td>86</td>
<td>86</td>
<td>0</td>
<td>86</td>
<td>86</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B 11</td>
<td>88</td>
<td>88</td>
<td>0</td>
<td>88</td>
<td>88</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B 12</td>
<td>174</td>
<td>174</td>
<td>0</td>
<td>174</td>
<td>174</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B 13</td>
<td>165</td>
<td>168</td>
<td>1.82</td>
<td>165</td>
<td>165</td>
<td>165</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B 14</td>
<td>235</td>
<td>235</td>
<td>0</td>
<td>235</td>
<td>235</td>
<td>235</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B 15</td>
<td>318</td>
<td>318</td>
<td>0</td>
<td>318</td>
<td>318</td>
<td>318</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B 16</td>
<td>127</td>
<td>127</td>
<td>0</td>
<td>127</td>
<td>127</td>
<td>127</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B 17</td>
<td>131</td>
<td>131</td>
<td>0</td>
<td>131</td>
<td>131</td>
<td>131</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B 18</td>
<td>218</td>
<td>218</td>
<td>0</td>
<td>218</td>
<td>218</td>
<td>218</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table C3 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 (SP) has been presented. The main idea behind the algorithm is the application of the Estimation of Distribution Algorithm for interpretation of bitstrings specifying selected Steiner vertices. This scheme ensures that every bitstring corresponds to a valid solution and minimizes 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 6,500 edges. The experimental results show that in more than 2% of all executions the GA finds a solution within 1% from the global optimum. This performance compares favorably with one of the best deterministic heuristics for SP as well as with an earlier GA by Kralj et al. Performance is also compared to that of branch and cut algorithms by Irani and Reay and by Kralj et al. While the runtime 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 problem instances within a minute amount of time.

Therefore conclude the following. In cases where an exactly optimal solution is absolutely required, the size of the given problem is too big and a runtime limit is imposed, one of the branch and cut algorithms are preferable. If the other hand if an exact solution is sufficient, or the problem is very large or a runtime limit is needed the GA presented here is the best choice of the possibilities considered.

Acknowledgement

The author wishes to thank Periklis Markakis, University of Michigan, who supervised this work in its initial phase when the author was at University of Michigan. Also thanks to Joe Casen and Paul Winter, Georgia Institute of Technology, for pinpointing possible improvements of an earlier version of the algorithm and for suggesting a suitable strategy for performance evaluation. Finally thanks to Per Kjell-Nilsen, Orkaid and Nlger Cup, Athens University for several useful discussions and suggestions concerning this work.
Figure C10  **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 C3 the main idea of the GAs is to apply heuristics for the hill-climber. In contrast, the genetic operators for crossover, mutation and inversion are all standard. They are parameterized by being very simple and blind in the sense that they do not utilize knowledge of the application domain in any way. This is true for the hill-climber. We frequently used GAs to improve the performance of a GA is to apply one algorithmic genetic operators and for operators exploiting application specific knowledge [12]. It is therefore likely that the performance of the GAs presented here can be further improved by applying such techniques.

2) **Other graph types**: An obvious weakness of the test-site used in this work is that all graphs are sparse and randomly generated. It remains to be seen how the performance e.g. dense graphs, rectilinear graphs, minimum 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 mutation operator, the search space reduction or the initial graph reduction?
Moreover, the used stop criteria reflects a priority of solution quality as being more important than runtime.

Fig. C9 shows for each generation the standard deviation of cost in the population. For a value of 0.2 in generation 0, the standard deviation decreases within 10 generations to about 2.0 and then stays at that level throughout the optimization process.

![Graph showing standard deviation of cost as a function of generation number.](image)

Figure C9 Standard deviation of cost as a function of generation number.

As described in Section C31, each generation is initiated by the generation of $M$ offspring individuals. For the total of $2M$ individuals, the best $M$ individuals are taken as values of the new population while the rest are discarded. Fig. C10 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 static around 50. The important thing to note is that the fraction of new individuals does 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 individuals 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 GAIBIS 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 C8, C9 and C10, which stem from a single execution of the GA with graph D15 as input. It should be emphasized that although the graphs stem from a specific single run, the picture they give is very typical.

![Graph showing cost of average and best individual as functions of generation number.](image)

**Figure C8** Cost of average and best individual as functions of generation number.

For each generation, the top graph of Fig. C8 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 1,157 and the best is 1,160. The global optimum is 1,116 is obtained first time in generation 29, and the algorithm terminates after 38 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 C3.1 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 23, corresponding to a 29% reduction of the runtime.
A polynomial increase through the classes B C D E the above
observations remain essentially panned. If only class Braphs are
considered it is difficult to give any definitions regarding performance
of the algorithm. These examples appear to be too simple.

C4.5 Comparison with Kapalis Algorithm

In this Section the Kapalis, EwardSith and Sith [13] is
denoted GAHS. A rational in Section C1, GAHS differs from
the GHS presented here in number of ways. Some other things, neither
animation operator nor allill-dither is applied in GAHS Moder,
the most significant differences concern the decoder and the cost compu-
tation. In GAHS 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
ensure that every tree space \( W \) each genotype is xor'ed with the fixed
strings 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 converted as the subgraph \( G_Z \) of \( G \) induced by \( Z \). In gen-
eral \( G_Z \) is not connected. As \( W \) it consists of \( k \geq 1 \) components. The
cost of a solution is defined as the sum of the cost of a minimal spanning
tree for each component plus a penalty term which grows linearly with \( k \).

Optimised results are given only for the class Braphs from the

Library. The solution quality obtained for each graph is reported
as the best result of the runs. For each graph some parameter setting
of GAHS has been found with which the global optimum is found
in the runs. However, the parameter setting varies with the problem
given. With the parameter setting for all graphs, GAHS finds
the global optimum approximately 70% of all runs and the worst result
obtained is 7.3% above the global optimum.

All experiments with GAHS are run on a Apple Mac IIx. No
total runtimes are given. Instead the time spent until the best solution
found appears first time referred to as last Improved Time (ITT) is
recorded. It is not clear exactly which computer is used i.e., how long
the algorithm takes to terminate beyond ITT for any of the graphs,
the average ITTs is in the range from 0 to 2000 sec. There is a time
limit of 4000 sec. for an complete execution.

GAHS is clearly inferior to each of the other algorithms considered
in this paper, both with respect to solution quality as well as runtime. W
For the class Exampels, SHI finds optimum for 4 of the 20 graphs, while the GA finds the optimum for 11 of these graphs. ΔC

\[ \text{worst} \leq \Delta ' \text{ sph} \]

holds for all bit on graphs in classes B and D in class E where

\[ \Delta ' \text{ ga} \leq \Delta ' \text{ sph} \]

for all graphs. In other words, with a slight exception even the worst results generated by the GA are equal to or better than the result generated by SHI. In turn, for the graphs were both SHI and the average execution of the GA fails to find the global optimum the expected relative error ratio \( \Delta ' \text{ any} \) of the GA is often an order of magnitude better than the error ratio \( \Delta ' \text{ sph} \) of SHI.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Error Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHI</td>
<td>487 667 705</td>
</tr>
<tr>
<td>GA</td>
<td>77.1 86.7 92.6</td>
</tr>
</tbody>
</table>

The C1: Summary of solution qualities obtained by the GA and SHI I.

The C1 summarizes the solution qualities obtained by the GA and the SHI. These figures are based on the results of all 60 executions of the GA and all 78 executions of SHI performed in total. For each algorithm the C1 gives the accumulated percentage of runs which gave a result within the stated relative error from optimum. E.g., 67% of all executions of the SHI 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 Exampels have been weighted by a factor of 10 to compensate for the much higher number of executions for each graph.

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

- The GA is capable of finding a high-quality solution for all graphs considered in a reasonable amount of time. This is not the case for any of the two branch-and-cut algorithms or for SHI.

- In most cases, the runtime of the GA is very similar to that of SHI. In a few cases, the GA is significantly faster than SHI, 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 SHI. As a consequence, the branch-and-cut algorithms are significantly faster than both the GA and SHI for some examples, while they are significantly slower on other examples.
The C12 lists the solution qualities obtained by the GA and the SHI together with the runtimes of all algorithms considered. Due to the extensive runtimes required for the graphs in this class, the GAs evaluated only one for each example. \( C_{ga} \) denotes the cost obtained by the GA, \( \Delta C_{ga} \) is the relative error of the solution found by the GA and the timeliness of algorithmic days denoted by \( T_{ga} \). Here, \( C_{ga} \) and \( T_{ga} \) can be considered estimates of \( C_{agt} \) and \( T_{agt} \), respectively.

It should be noted that the listed value of \( C_{agt} \) for El8 may not be the global optimal but according to the information in the library, it is the best known solution as found by Bailey's algorithm [3]. The optimal for this graph was not found within a quality limit of 2,000,000 seconds on the Gay SX70. Graetz et al. [6] also encountered problems with El8. No runtime is listed for this graph since the algorithm did not terminate within a quality limit of 10 days on the SX800 [2]. However, using the progression of runtime 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.

SHI exceed the optimal limit of 5,000,000 seconds for graphs E3, E8, E9, E10, E13, E14, E15, E18, E19 and E20. The estimated total time required by SHI for these graphs varies from 8,100,000 seconds for E3 to \( 4.3 \times 10^7 \) seconds, or more than 16 months, for E20. Compared to the branch and bound algorithm and SHI, the runtimes of the GA are very similar. Regarding solution quality, SHI finds the global optimum for 4 of the graphs and has a worst relative error ratio exceeding 9% for El8. The Gafitq optimum for 11 graphs and has a worst relative error ratio less than 2%.

### C4.4.5 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 SHI for all graphs in classes B, C and D, the following can be observed. Of a total of 58 graphs, SHI finds the global optimal solution for 34 graphs, while the Gafitq optimum is found in 10 for B graphs and at least one time of 10 for 5 graphs.
vertex degree increases.

Of the class of graphs with $7$ of the graphs, while the $C_{\text{max}}$ is the optimum for at least one, for $17$ graphs and every time for $13$ graphs. $SHI$ has relative errors exceeding $2\%$ for $5$ graphs while the only optimum for the $C_{\text{max}}$ graph $D0$ for all graphs with

\[ C_{\text{max}} \leq C_{\text{shl}} \]

and

\[ C_{\text{worst}} < C_{\text{shl}} \]

for $13$ graphs.

Of this class of problems, the runtime for both branch and cut algorithms varies by three orders of magnitude and are as high as in the $20,000$ seconds range corresponding to $23$ days of computation. The runtime of $SHI$ also varies significantly. For practical reasons, it became necessary to introduce a runtime limit of $5,000$ seconds for this algorithm on graphs from class $D$.

In $5,000$ seconds, $SHI$ did not compute its runtime within this limit, it was terminated and the best solution found so far was used. This happened for graphs $D19$ and $D20$. For these graphs, the total time needed by $SHI$ is estimated to be $5,000$ seconds and $69,000$ seconds, respectively. These estimates can be considered to be quite accurate since there are based on measurements of the runtime for each pair of vertices $x, y \in V$, cf. Fig. C7, which is then scaled with the relative number of vertex pairs not yet considered at the time the runtime limit is exceeded. The average runtime of the $C_{\text{max}}$ varies from $101$ seconds for $D5$ to $3,441$ seconds for $D19$, i.e., by a factor of $7$. This variation is still explained by the variation of the other algorithms considered.

For graphs $D8$ $D9$ $D10$ $D13$ $D14$ $D15$ $D18$ $D19$ and $D20$, the $C_{\text{max}}$ on average an order of magnitude faster than $SHI$ while for the remaining graphs, the runtimes of these algorithms are comparable.

### C4.44 The $E$ Graphs

For the graphs from class $E$, the effect of graph reduction follows a pattern which coincides perfectly with the patterns observed for classes $C$ and $D$.

Even after reduction, the search space sizes for the class $E$ graphs are enormous. Using the bound

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

where $k = \min(m-2, n-m)$ reveals that number of graphs in this class.

For search space exceeding $10^{100}$ graphs. Especially the search space for $E13$ exceeds $10^{231}$ graphs, and for $E18$ it exceeds $10^{242}$ graphs. These bounds are computed after graph reduction have been performed.
However, as the average vertex degree increases, the effect of reductions of types a and b (see Section C3.2) decreases significantly. When $m$ is small, the effect of reductions of type c 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 algorithms in terms of shortest paths, minimum spanning trees, etc., the number of edges are not that important for the performance of the algorithm.

The C6 show that the G9 finds the global optimum least one for all except and even for C2 of the graphs, while SHI finds the optimum for 10 of the graphs. While the average Grit over the SHI finds the global optimum $\Delta_{\text{opt}}$ is often an order of magnitude better than $\Delta'_{\text{opt}}$. This is the case for C3, C4, C9, C14, C18 and C19. For C18 and C19 the solutions produced by SHI are very poor with results in the 6-7% range. The results for C16 are in direct contrast to all other results. While the SHI finds optimum the Grits are 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 $\Delta'_{\text{worst}}$ of 900%.

In Table 1, and subsequent tables, $T_{k1}$ denotes the runtime of the branch-and-cut algorithm by Dupa et al. [5]. Depending on the problem, the runtime for both branch-and-cut algorithms varies extremely. Dupa's algorithm runs from 10 seconds for C16 to over 4,000 seconds for C18, while Iam's algorithm runs from 5 seconds for C5 to over 20,000 seconds for C18. As a consequence, the branch-and-cut algorithms are significantly faster than both the Grit and SHI for some graphs and significantly slower for others. The runtimes of the Grit and the SHI are similar for most graphs, although the Grits significantly faster for graphs C15, C19 and C20. The time variation $T_{\sigma}$ of the Grit is relatively small.

### C4.4.3 The Dgraphs

The effect of graph reductions on the class Dgraphs show a pattern similar to that observed for the Cgraphs, although the pattern is even clearer. Most graphs are reduced significantly more especially Di. 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 qinal solution is found solely by performing graph reduction.

The C3 represents the solution quality obtained by G to the 

globally qinal solution as well as to the solution family by SHI. 

\( C_{opt} \) is the global qimal with \( C_{sph} \) is the solution family by SHI. \( C_{avg} \) and \( C_{worst} \) is the best, average and worst result produced by the 

G in the 10 runs, while \( C_{avg} \) gives the standard deviation of the 10 

cost values. \( \Delta C_{sph} = 100 \frac{C_{sph} - C_{opt}}{C_{opt}} \) is the relative error in percent 

do the solution family by SHI compared to the qimal solution. Simi-

larly \( \Delta C_{avg} = 100 \frac{C_{avg} - C_{opt}}{C_{opt}} \) gives the average error of the solu-

tion family by G and \( \Delta C_{worst} = 100 \frac{C_{worst} - C_{opt}}{C_{opt}} \) is the worst 

error produced by G. Finally, \( N_{opt} \) gives the number of the 10 

runs which did not find the global qimal. This notation is also used 
in the following sections.

A can be seen that C3 finds the global qimal for all examples in 
every execution. SHI performs similarly for all graphs except B13 for 
which it has a 1.8% overhead.

The C4 represents the runtime of the C4 with that of SHI and the 

branch-and-cut algorithm Lann and Bailey [23]. \( T_{sph} \) is the time for SHI. \( T_{avg} \) 

does the runtime of the latter algorithm and \( T_{worst} \) is the time for SHI. \( T_{avg} \) 

does the runtime of the latter algorithm and \( T_{worst} \) is the time for SHI. \( T_{sigma} \) 

does the standard deviation of the time for the 10 runs. Chvatal 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 

conclusions regarding differences in speed for the different algorithms.

The fact that all three algorithms find qimal solutions (except for 

SHI on B13) in a very short time suggests that these examples are 
simply too small to facilitate any distinction of performance of the algo-

rithms. For several of the graphs the search space after graph reduction 

are indeed very small and the largest search space is that of B17 with less 

than 10\(^6\) pints, which is not that much for a combinatorial optimization 

problem.

### C4.4.2 The CGraphs

From the C5 it can be seen that the graph reductions are also very 
effective on net graphs in the Cclass. The especially graph C5 which 
after reductions has a search space size of only approximately 10

\( \times \) 6 pints.
The GA has been executed 10 times for each example in the B Card Ddclasses. 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 parameters' settings are $M = 4$, $S = 5$, $p_{mt} = 0.05$ and $p_{inv} = 0.1$. These values are used for all executions, i.e., no problem-specific tuning has been made. As noted in Section C1, fixed parameter values are of major importance from a practical point of view.

The GA as well as SHI are implemented in the C programming language. For both algorithms, examples from classes B Card Ddclass executed on a Sn SRC II workstation using 2 M MB. These examples require at most 10 Mb memory. For the class Exemplar, the memory requirement is about 58 M. Therefore, for these examples, the GA as well as SHI are executed on a DEC 5000-20 workstation using 28 M MB.

The branch-and-cut algorithm by Luna and Bailey [2] is a further development of the algorithm presented in [3], but instead of using a GA, it is re-executed on a Sn SRC 2 workstation. This runtime is roughly as fast as the Sn SRC II but probably somewhat slower than the DEC 5000-20. Ciparut et al.'s algorithm [5] is executed on a VAX 8600 which is at most as fast as the other runtimes. When comparing absolute runtimes in Section C4.4 the reader should keep these differences regarding the used hardware in mind. However, the runtime variations caused by different instances are insignificant compared to the variations caused by different problem instances when considering a specific algorithm.

### C4.4 Results

In the following sections, the detailed experimental results for all problem classes are presented. The tables referenced can be found in Section C7. A summary and conclusion of the results are given in Section C4.4.5.

#### C4.4.1 The B Graphs

The C2 lists the characteristics of the problem in class B. Before and after the graph reductions of Section C2.2 are performed, the reductions significantly impact all graphs. Especially graphs B1, B3 and B9 are
due to the fact that all distances have been precomputed. Since $O(m)$ candidate solution trees $T_i$ are computed, the total runtime of $SHI$ becomes $O(n^3 + m^4 n^2)$.

```c
graphRelation();
c(TSPH) = \infty;
\forall \ x, \ y \in W, \ x \neq y \ do
    T = G_{xy};
    Q = W \cap V_{xy};
    while W \setminus Q \neq \emptyset do
        find a vertex $z \in W \setminus Q$ closest to a vertex in $T$;
        add to $T$ shortest path from $T$ to $z$;
        $Q = Q \cup \{z\}$;
        if $c(T) < c(T_{SPH})$ then $T_{SPH} = T$;
output $T_{SPH}$;
```

Figure C7: Outline of $SPH-I$.

## C4.3 Experimental Method

The goals are evaluated for kinds of heuristics:

- The solution quality obtained is compared to the global optimum.
- The absolute runtime is compared to that of two distinct branch-and-bound algorithms by Lem and Reaney [2] and Clarke, Gyenes and B"{o} [3].
- Solution quality and absolute runtime is compared to that of $SHI$.
- Comparison with the Caley-K"{a}ralis et al. [18].

The branch-and-bound algorithms are guaranteed to find the global optimum. However, runtime may be unacceptable for some problem instances or even some of the smallest branching speed. It is therefore of interest to investigate if an $a$-approximation solution can be found for all problems using a moderate amount of time.
For a given graph, the size of the search space \( S(n, m) \) to be explored by the GA is

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

where \( k = \min(m - 2, n - m) \), since this is the number of possible distinct degrees of the Steiner vertices. Some of the problem instances considered represent extremely large search spaces, as will be seen in Section C4.4. However, as noted in Section C3.7, the corresponding phenotype spaces are smaller.

### C4.2 Iterated Shortest Path Heuristic (ISH)

A study in Section C3.4 of a comparative study of the deterministic heuristic AD and ISH has been made by Viterbi and Sith[3].

Several variants of these heuristics, especially those of repetitive variants of ISH are also considered in this study. The AD is in general considered to be one of the best deterministic heuristics, which is also confirmed by the investigations in [3]. However, the results also reveal that some of the repetitive variants of ISH perform better than AD with respect to solution quality. Furthermore, by applying initial graph reductions, the runtime of the repetitive ISH variants can be made comparable to that of the other heuristics. One of the specific conclusions in [3] is that the undirected random graphs considered the repetitive ISH variant did not outperform all other heuristics. Therefore, this heuristic has been chosen for comparison with the GA.

Fig. C3.4 outlines one implementation of ISHZ, a detailed ISH. It starts with constructing \( D(G) \) and performing graph reductions as described in Section C3.2. 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 built which spans all vertices in \( W \). \( T \) is initialized with a shortest path between two of the vertices to be sparsal 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 subtree obtained.

As described in Section C3.2, routine \textit{graphReductions} requires time \( O(n^3) \). The construction of each candidate subtree \( T \) takes time \( O(m^2) \).

Since the "while" loop is iterated \( O(m) \) times and it takes time \( O(m^2) \) to find each vertex and extend \( T \) with a shortest path to it. This is
in Step 3 of the decoding process is almost always true, and as a conse-
quence, Step 4 is rarely executed. Therefore, the true bottleneck of the
algorithm is the \(N(2^{m-2})\) computation performed in Step 2 of the decoding
which requires time \(O(m^2)\).

**C 4 Experiments**

This section describes the experimental method applied and the results
drawn. Characteristics of the test examples used are given in Sec-
tion C41. The deterministic heuristic \(S\)-\(1\) used for comparison is de-
scribed in Section C42 and Section C43 describes the done method
for performing the comparative experiments. The results are reported
and discussed in Section C44. A rational in Section C1 an earlier
\(C\) for \(S\) that have been developed by \(K\) and \(d\) and a comparison
to this algorithm is presented in Section C45. Finally, Section C46
describes the typical behavior of the \(C\) during an optimization process.

**C4.1 Test Examples**

The algorithm is tested on all 78 \(S\)-\(1\) instances from the \(S\)-\(1\) library \(\delta\).,

According to their size, these graphs are divided into four classes denoted
by \(C\), \(D\), \(E\), \(A\). Graphs are generated randomly subject only to the
computation constraint, that is, the topology is random and the vertices
to be sampled are selected at random. Every edge cost is a random
integer in the interval \([1, 10]\). In class \(B\) an integer \(n\) is equal to 30, 75
or 100. The value of \(m\) is either \(n / 6\), \(n / 4\), \(n / 2\) and the average vertex
degree is either 25 or 4. Since all combinations exist, class \(B\) consists
of 18 graphs. Classes \(C\), \(D\), \(E\) consists of graphs with \(n\) equal to 50,
100, and 200, respectively. \(m\) equals 5, 10, \(n / 6\), \(n / 4\), \(n / 2\) and the
average vertex degree is 25, 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 facili-
tates comparison with the global optimum solution. The global optimum
was first computed by J. E. Bailey who developed a branch-and-
algorithm which was executed on a Cray \(X\)-1 supercomputer \(\delta\).
Apart of the ring is then selected at random and reversed. More specifically, two parts \( x, \ y \in \{ 0, 1, \ldots, r-1 \} \), \( x \neq y \), are selected at random. The operator then defines the reworking \( \pi' \) of \( g \) as

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

for all \( i = 0, 1, \ldots, r - 1 \). The inversion operator is illustrated in Fig C6.

**Example inversion**

\[ \alpha : \{ (2, 0), (3, 1), (0, 1), (4, 0), (1, 0) \} \]

After inversion with \( x = 2 \), \( y = 0 \):

\[ \alpha : \{ (0, 1), (3, 1), (2, 0), (1, 0), (4, 0) \} \]

**Figure C6 Illustration of the inversion operator with \( r = 5 \).**

### C3.8 Time Complexity

The filter routine described in Section C34, the generation of each of the initial individuals, and the genetic operators crossover, mutate and insert each requires time \( O(r) = O(n - m) \). The repeated decoding using DH is the most expensive operation of the GA. Some knowledge of shortest paths is also required when preparing some of the initial graph reduction. \( D(G) \) is computed once and for all as mentioned in Section C32. This reduces the time of Step 1 of DH to \( O(1) \) and as a consequence, the decoding's time can now be termed \( O(mn \log(nm)) \).

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 \( O(M[n m \log(nm) + \log M]) \).

Measurements reveal that the vast majority of the total runtime is spent on decoding. It also turns out that in practice the graph feeded

\[ \text{The definition of } \pi' \text{ relies on the mathematical definition of modulo, in which the remainder is always non-negative.} \]
Given individuals:

\[
\alpha : \{ (2, 1), (0, 1), (1, 0), (4, 0), (3, 0) \} \\
\beta : \{ (1, 0), (2, 1), (4, 1), (3, 1), (0, 0) \}
\]

Step 1. Random \( \beta \):

\[
\gamma : \{ (2, 1), (0, 0), (1, 0), (4, 1), (3, 1) \}
\]

Step 2. Crossover with \( x = 2 \):

\[
\phi : \{ (2, 1), (0, 1), (1, 0), (4, 1), (3, 1) \} \\
\psi : \{ (2, 1), (0, 0), (1, 0), (4, 0), (3, 0) \}
\]

Step 3. Gly \( \phi \) is subjected to the filter:

\[
\phi : \{ (2, 0), (0, 1), (1, 0), (4, 1), (3, 1) \}
\]

Figure C5: Illustration of the crossover operator with \( m = r = 5 \).

C3.7 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 \( p \).

The schema 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 on internal genotypes, the fitness of produced offspring depends on which of the possible genotypes are used as coding 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 \( p \), the inversion operator reads the tuples of a given genotype \( g \) by altering its ordering \( \pi \). This does not change the phenotype corresponding to \( g \). To obtain a uniform probability of mutation of all tuples, we consider the genotype to form a ring.
C3.5 Fitness Measure

Given an application \( P = \{ p_0, p, \ldots, p_m \} \), the relative evaluation of Eq C2 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 \( C(p_0) \geq C(p_1) \geq \ldots \geq C(p_m) \). The fitness \( F \) of \( p_i \) is then computed as

\[
F(p_i) = \frac{2^i}{M - 1}, \quad i = 0, 1, \ldots, M - 1.
\]

This fitness computation scheme is called ranking and is discussed in [2].

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 does not necessarily give the best individual twice the probability of the random individual of being selected for crossover.

C3.6 Crossover Operator

Given two parent genotypes \( \alpha \) and \( \beta \), the crossover operator generates two offspring \( \phi \) and \( \psi \). The parent genotypes are not altered by the operator. An example of crossover is shown in Fig C5. In this section, a superscript specifies which individual the marked property is part of.

Crossover consists of three steps:

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

2) Each string is given the same ordering as their parents, i.e., \( \pi^\phi = \pi \alpha \). 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 \( \phi \) and \( \psi \) is then defined

\[
\hat{\phi}_{x}(k) = \begin{cases} 
\check{\phi}_{x}(k) & \text{if } k \leq x \\
\check{\gamma}_{x}(k) & \text{if } k > x
\end{cases}
\]

and

\[
\hat{\psi}_{x}(k) = \begin{cases} 
\check{\psi}_{x}(k) & \text{if } k \leq x \\
\check{\gamma}_{x}(k) & \text{if } k > x
\end{cases}
\]

where \( \pi = \pi \alpha \).

3) Finally, both \( \phi \) and \( \psi \) are subjected to the filter routine, if necessary.
For a given instance of $\Sigma G$, assume that indexing $0, 1, \ldots, r - 1$ of the vertices in $V \setminus W$. Let $\pi : \{ 0, 1, \ldots, r - 1 \} \rightarrow \{ 0, 1, \ldots, r - 1 \}$ be a bijective mapping. A genotype is then a set of $r$ tuples:

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

where $i, k \in \{ 0, 1 \}, k = 0, 1, \ldots, r - 1$. The Seirer vertices $S \subseteq V \setminus W$ specified by the genotype is $S = \{ v \mid k \in V, \hat{s}(k) = 1 \}$. The Seirer tree $T(G)$ corresponding to the genotype is the tree obtained by deleting the set $S \cup W$ as the vertices to be removed. In Step 5 of the algorithm, every vertex $v \not\in W$ of degree 1 is deleted. Note that the Seirer tree is independent of $\pi$. In other words, the Seirer tree constituting the phenotype of an individual does not change if the tuples in its genotype are reordered.

A set of values of the $i$'s in a genotype corresponds to a valid phenotype. However, later [2] has shown that a MT in $D(G)$ exists, which has at most $m - 2$ Seirer vertices. This result relies on the fact that regardless of the edge cost function, the edge costs in $D(G)$ always satisfy the triangle inequality. Here, 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 $g$ ensures the satisfaction of $|S| \leq \min(m - 2, r)$ by randomly selecting and deleting the necessary number of set bits.

Win the initial random population has been generated, the filter is applied to each of the individuals. Further 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 DH is not used for use as decoder because it is a specially good heuristic in terms of real quality. In [3], the performance of DH is compared to that of two other well-known phylum tinal heuristics for the $\Sigma G$: The Shortest Path Heuristic (SH) by Hikami and Masuyama [27] and the Average Distance Heuristic (AD) by Edward Sith and Oak [28]. With respect to real quality, the DH is already outperformed by both these heuristics. The reason to use DH for decoding is that it provides a way to interpret any set of selected vertices as a valid Seirer tree and severely that it is relatively fast. To implement advantage of considering valid Seirer trees only is that it eliminates the need for subtlety in the cost measure, and this avoids potential problems of assigning a suitable cost value to an invalid or incomplete solution.
C3.4 Genotype and Decoder

The basic idea of the genotype and the associated decoder is the following: The genotype specifies a set of selected Seirer vertices. The decoder computes the corresponding phenotype by executing the DH using the union of the selected Seirer vertices and $W$ as the set of vertices to be spamed. The selected Seirer 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 C3.7, 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:

**Figure C4** The steps of DH given the input graph from Fig. C.1.
C3.3 Distance Network Heuristic (DNH)

The key point in designing a Gis the design of a suitable genotype of
an individual together with its interpretation by the decoder. The genetic
encoded decoder is based on the Distance Network Heuristic
(DNH), a deterministic heuristic for the SG developed by Kat et al. [2].
Therefore, before proceeding by presenting the genotype and the decoder,
the DNHs described.

Given a graph \(G=(V,E)\), a cost function \(c\) and a subset of vertices
\(W\) in accordance with the definition of SG in Section C2, 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 C4, given as input the graph \(G\) of
Fig C1 and the subset \(W = \{v_0, v_1, v_2, v_3\}\).

If \(D(G)\) is not known, Step 1 of DNH requires time \(O(m^2)\) to com-
pute 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 \(T_1\) in \(G_2\) corresponds 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 Keld's algorithm[1]. The final step is due
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 \textit{graphReductions} terminates when no reduction of any type succeeded for a complete iteration, i.e., when no reduction can reduce $G$ further.

\begin{verbatim}
count $D(G)$;
repeat
  reduction(c);
  reduction(b);
  reduction(d);
  reduction(a);
until no important in an iteration;
\end{verbatim}

Figure C3 \textit{Outline of routine graphReductions.}

To deduce the worst case time complexity of \textit{graphReductions}, start by considering the minimum time spent 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 spent on type $d$ reduction is $O(n^3)$. Each execution of $\textit{reductions(}x\text{)}$ requires at most time $O(n^{2})$ when either $x /=d$ 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 deterministically in the sense that after executing $\textit{reductions(}c\text{)}$ nodes 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 \textit{graphReductions}. Hence, there can be no more than $O(n)$ iterations. In total this gives routine \textit{graphReductions} the time complexity $O(n^3)$.

Although it is not difficult to construct a graph for which one of the reductions performed by \textit{graphReductions} applies, the routine has been observed to be very effective on many graphs, as will be seen in Section C4.4. When applied to the graph of Fig. C1, the result is the degenerate graph consisting of one vertex only, implying that a M
eric

...
Assume that \( v \in W \) and that 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 \( W \setminus \{v\} \) with the closest \( G \) to \( u \). If \( c(e_{uw}) \leq c(e_{wv}) \), then any MST includes \( e_{uw} \). Therefore, \( G \) can be contracted along this edge. Note that \( u \in W \Rightarrow z = u \Rightarrow c(s(u, z)) = 0 \). 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 when a reduction of type \( c \) or \( d \) is performed. Shortest paths are also repeatedly needed by the \( G \)s as will be apparent in Section C3.4. Therefore, the distance graph \( D(G) \) is computed initially using Floyd's algorithm [1] and requires time \( O(n^3) \).

When one of the above reductions is performed, \( D(G) \) has to be (re)computed. 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. When a contraction is performed, \( D(G) \) is updated using an \( O(n^2) \) algorithm by Dine and Flanagan [6].

In [31], the following reduction is also suggested along with the reductions described above. If \( \max\{c(s(v, w)), c(s(v, u))\} < c(e_{uw}) \), \( e_{uw} \in E \) and \( v \in W \), then any MST includes \( e_{uw} \), 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 Dine and Flanagan'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 at. Furthermore, reductions of distinct types are mutually dependent in the sense that performing all possible reductions of some type may allow or disallow 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]. To arbitrarily choose among reducing reductions in routine graphReductions is shown in Fig C3. Routine reductions(\( x \)) performs a single traversal of all vertices (or edges in the case of type c reduction) of \( G \) in an unspecified order and carries out a reduction of type \( x \) whenever possible.
2. Retain 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 genotypes, or genes, of its argument, as described in Section C.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 position of the genes of $p$ with respect to the crossover operator. The inversion operator will be described in Section C.7. Retain optimize($s$) performs a simple 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 is made to reduce the size of the given problem via formal, using standard graph reduction techniques. Retain graph Reductions of Fig. C.2 perform a kind 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 $P(v, w) \subseteq E$ denote the shortest path between $v$ and $w$. To form reductions used are:

a) Assume $\deg(v) = 1$ and $e_{vw} \in E$. If $v \in W$ any MST can induce $e_{vw}$. Here, $v$ 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 induce $e_{vw}$, i.e., in this case $v$ and $e_{vw}$ can also be deleted.

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

c) If $e_{uw} \in E$ and $c(e_{uw}) > c(P(v, w))$ then no MST can induce $e_{uw}$, which therefore can be deleted.
Fig. C2 Outline of the algorithm

Computes the fitness of each of the given individuals, while best G finds the individual with the highest fitness. The execution of the outer "repeat" loop corresponds to the simulation of one generation. Through the simulation the number of individuals \( M = \left| P_C \right| \) 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 function of a set of diploids \( P \) \( N \) 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 C3.6 generates two diploids.
C.3 Description of the Algorithm

In this section the decodred algorithm is described in detail. First an overview of the algorithm is given in Section C3.1. Initially an attempt to reduce the size of a given problem is made by applying some graph reduction techniques described in Section C3.2. The main idea of the GA is the application of the Fitness Noval Heuristic for interpretation of the representation manipulated by the genetic operators. This is discussed in Sections C3.3 and C3.4. Other aspects of the algorithm are described in Sections C3.5, C3.6 and C3.7. Finally the time complexity of the algorithm is discussed in Section C3.8.

C3.1 Overview

The concept of genetic algorithms, introduced by John Holland [16], is based on natural selection. 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. 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 simulates recombination, and mutation, which simulates 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.

An phenotype is the physical appearance of an individual, while a genotype is the corresponding representation of a 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 obtained by a decoder. An introduction to genetic algorithms is given in [12].

Fig. C2 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 C3.2. The initial current population $P$ is constructed from randomly generated individuals by routine generate. Routine evaluate described in Section C3.5 can
**The Steiner Problem in a Graph (SPG):** Given a connected undirected graph \( G = (V, E) \), a positive edge cost function \( c : E \rightarrow \mathbb{R}_+ \), 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 \( G \). An acyclic subgraph \( G' \) with minimal cost is called a Minimal Steiner tree (MST) for \( W \) in \( G \). A 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.

![Diagram of a graph with edge labels]

**Figure C1:** An example instance of the SPG. The highlighted vertices constitutes \( W \).

Though this problem, 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 \( [2] \) 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 show the following:

- The G algorithm clearly outperforms the GA in [18] with respect to solution quality as well as runtime.

- The solution quality obtained by the G algorithm is at least as good as that obtained by SHS, 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 G algorithm is significantly faster.

- As opposed to the branch-and-cut algorithm, the GA is not guaranteed to find a global optimum solution. However, the experiment reveals that the GA finds the global optimum more than 77% of all runs and is within 1% from the minimum time 2% of all runs. While the GA is capable of finding near-optimum solutions for all test examples in a moderate amount of time, the runtime of the branch-and-cut algorithm varies extremely and can preset some of the largest problem instances from being solved.

The paper is organized as follows. A problem definition is given in Section C2. Section C3 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 C4, and in Section C5 possible directions for future work are suggested. Finally, Section C6 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) $(u, v) \in E' \Rightarrow u, v \in V'$, and 3) $\forall u, v \in V' \setminus (u, v) \in E \Rightarrow (u, v) \in E'$. 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 $(u, v)$ equals the cost of the shortest path in $G$ from $u$ to $v$. For a given edge cost function $c : E \rightarrow \mathbb{R}$, the cost of a graph $G$ is the sum of the cost of all edges of $G$, and is denoted $\sum (G)$. The problem considered in this paper is defined as: 
C.1 Introduction

The Steiner Problem in a Graph (SP) 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 subgraph spanning the designated vertices. The SP arises in a large variety of diverse optimization problems such as network design, multiprocessor scheduling and integrated circuit design [10, 2].

Numerous algorithms of various kinds have been developed for the SP. Best algorithms can be found in e.g. [2, 3, 5, 8, 13, 23, 2]. However, since the SP is NP-complete [1] these algorithms require exponential worst case time complexities. Therefore, a significant research effort has been directed towards polynomial time heuristics, e.g. [2, 20, 24, 25, 27, 31]. Suggested algorithms have also been applied to SC [7].

The shortest Steiner Path (SSP) is an important special case of SP [14], which is still NP-complete [11]. While at least two genetic algorithms for SSP have been published [15, 17], we are aware of only one previous genetic algorithm (GA) for the SP developed by Kipalis, Pappal D and Sith [18].

The contribution of this paper is a new GA for the SP which differs significantly from the approach of Kipalis et al. [18] in number of ways. While individual solutions are allowed to be used 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 the use of an inversion operator.

The performance evaluation strategy also differs significantly. While the parameter settings used in [18] vary from problem to problem a fixed set of parameter values has been used for all results reported in this paper. From a practical point of view, stochastic algorithms 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 algorithms tested on all SP instances from the DIMAP [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 CLIBay [1], an iterative version of the Shortest Path heuristic called SHL, which is one of the very best deterministic heuristics [3], and two recent benchmark algorithms human and Bailey [2] and Giclas, Gues and Ro [5].
Abstract

A new Genetic Algorithm (GA) for the Steiner Pollenina Graph (SPG) is presented. The algorithm is based on a bitstring encoding. A bitstring specifies selected Steiner vertices and the corresponding Steiner tree is created using the Extreme 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 ORlibary of which the largest graphs have 2500 vertices and 62,500 edges. We 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 77% of all runs executions and is within 1% from the global optimum 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 (SPH). For all test examples but one, even the worst result ever found by the GA is equal to or better than the result of SPH. In many cases, the average error ratio of the GA is an order of magnitude better than that of SPH. The runtime of the GA is minute for all test examples. This is in contrast to SPH as well as the branch-and-cut algorithms for which the runtimes in some cases are extremely high.
Appendix C

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


Bibliography


B. 7 Conclusion

This paper has presented a stochastic optimization algorithm called SGA that combines the genetic algorithm with simulated annealing. The approach is application independent and adaptive. The performance of the unified algorithm on the microcell placement problems has been investigated. It is empirically shown that a mixture of GAA and SGA performs better than a pure GAA on this problem. Furthermore, on MC placement benchmarks, we obtain better results or comparable to previously published results by using a GAA/mixture. The current implementation is not yet complete; however, significant improvements can be made. Therefore, conclude that the approach presented is a very promising approach to microcell placement.
quality of the best computed results are compared to the best published results. The absolute area is one area in m^2. To ease comparison, relative areas have also been computed by assigning the best result for each benchmark the relative area 1. The total intercost length in mm and the total number of vis is also given.

<table>
<thead>
<tr>
<th>Architecture</th>
<th>System</th>
<th>Area</th>
<th>Rate</th>
<th>Vis</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>relative</td>
<td>absolute</td>
<td>length</td>
</tr>
<tr>
<td>4xe</td>
<td>SGA</td>
<td>1.00</td>
<td>53.58</td>
<td>49</td>
</tr>
<tr>
<td></td>
<td>BB[8]</td>
<td>1.09</td>
<td>54.05</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>Settle Silicon [11]</td>
<td>1.02</td>
<td>54.77</td>
<td>30</td>
</tr>
<tr>
<td>Xerox</td>
<td>Settle Silicon [11]</td>
<td>1.00</td>
<td>25.79</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>BB[8]</td>
<td>1.05</td>
<td>26.17</td>
<td>68</td>
</tr>
<tr>
<td></td>
<td>SGA</td>
<td>1.03</td>
<td>27.15</td>
<td>69</td>
</tr>
<tr>
<td></td>
<td>BB[4]</td>
<td>1.10</td>
<td>28.47</td>
<td>63</td>
</tr>
<tr>
<td></td>
<td>NECO</td>
<td>1.125</td>
<td>29.01</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>VIL</td>
<td>1.230</td>
<td>31.17</td>
<td>86</td>
</tr>
<tr>
<td>16p</td>
<td>SGA</td>
<td>1.00</td>
<td>11.81</td>
<td>21</td>
</tr>
<tr>
<td></td>
<td>Settle Silicon [11]</td>
<td>1.08</td>
<td>11.81</td>
<td>20</td>
</tr>
<tr>
<td></td>
<td>BB[8]</td>
<td>1.09</td>
<td>12.15</td>
<td>28</td>
</tr>
</tbody>
</table>

The B3 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 SGA is highly captive with respect to solution quality, the actual implementation of the algorithm requires significantly more runtime than the other systems listed in Table B3. The approach [8] is about 23 times faster and the Settle Silicon approach [11] is about 1020 times faster. However, there is a number of reasons why we expect this runtime to be improved significantly. First of all, in the actual implementation, the majority of the runtime is spent computing dead entities in a way inefficient manner. All dead entities are computed by a brute force search. Instead, by using a suitable data structure, not dead entities could be computed much faster by a quick query from a previous data structure computation. Furthermore, the inherent parallelism of this kind of algorithm allows a parallel version of the unified algorithm to be expected for any MM architecture.

\[\text{Referenced here as found in [4, 11].}\]
are quite close to the optimum suggesting that the run for `inherent` over the GA is small. However, for the `A` and `Xerox` benchmarks, the values of $A_{test}^\text{test}$, $A_{avg}^\text{avg}$ and $A_{\sigma}^\text{sigma}$ are significantly improved by the mixed strategy, while for the `Ip` 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. B18. The results of each of the 4 runs of `SA` are grouped so that the $i^{th}$ group (left to right on Fig. B18) corresponds to area (in $m^2$) in the interval $[27.5+0.3i, 27.5+0.3(i+1)]$. The height of a bar indicates the number of the 4 runs belonging to the group.

![Performance comparison on Xerox](image)

At average the `GSA` is about 40% slower than the `GA` for the `Xerox` benchmark while for `Ate` and `Ip` 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. 15 consecutive generations without any other improvements, the search is continued for at least another 20 generations, no matter how insignificant the improvement might be been.

### B6.3 Comparison with other Systems

The of the 4 parameters generated for each benchmark by `SA`, using the mixed `GSA` strategy, has been rated and compared using `Music` [4], which is part of the `Cattel` `CA` framework. In `B6.3` the
B6.2 Comparing the GA with a Mig 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 $M = 5$, $S = 20$, $p_r = 0.05$, and $p_{mv} = 0.05$ were used for both strategies. For the GA, $\beta = 1 - 10^{-4}$, while for the GA/SA, $\beta = 0.7$, $\gamma = 1.5$, $R = 80$, $P = 0.91$, $\alpha = 0.6$ and $\lambda = 1$.

The same parameter setting has been used for all three benchmarks, i.e., no problem-specific tuning has been made. For each benchmark and each set of parameters, SA/SA was executed 40 times. Table B2 summarizes the results. $A_{\text{best}}$ and $A_{\text{any}}$ denote the best and the average estimated minimum, respectively; $A_{\sigma}$ denotes the standard deviation. Since SA 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. $T_{\text{any}}$ denotes the average CPU time in seconds, and $T_{\sigma}$ is the standard deviation of the CPU time.

<table>
<thead>
<tr>
<th>Bench</th>
<th>Quality</th>
<th>GA</th>
<th>GA/SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ate</td>
<td>$A_{\text{best}}$</td>
<td>5.39</td>
<td>5.47</td>
</tr>
<tr>
<td></td>
<td>$A_{\text{any}}$</td>
<td>5.64</td>
<td>5.20</td>
</tr>
<tr>
<td></td>
<td>$A_{\sigma}$</td>
<td>0.86</td>
<td>0.77</td>
</tr>
<tr>
<td></td>
<td>$T_{\text{any}}$</td>
<td>3.13</td>
<td>3.38</td>
</tr>
<tr>
<td></td>
<td>$T_{\sigma}$</td>
<td>1.53</td>
<td>2.20</td>
</tr>
<tr>
<td>Xerox</td>
<td>$A_{\text{best}}$</td>
<td>7.81</td>
<td>7.54</td>
</tr>
<tr>
<td></td>
<td>$A_{\text{any}}$</td>
<td>8.91</td>
<td>8.21</td>
</tr>
<tr>
<td></td>
<td>$A_{\sigma}$</td>
<td>0.53</td>
<td>0.46</td>
</tr>
<tr>
<td></td>
<td>$T_{\text{any}}$</td>
<td>9.54</td>
<td>13.12</td>
</tr>
<tr>
<td></td>
<td>$T_{\sigma}$</td>
<td>3.08</td>
<td>2.56</td>
</tr>
<tr>
<td>Hb</td>
<td>$A_{\text{best}}$</td>
<td>12.85</td>
<td>12.83</td>
</tr>
<tr>
<td></td>
<td>$A_{\text{any}}$</td>
<td>13.30</td>
<td>13.24</td>
</tr>
<tr>
<td></td>
<td>$A_{\sigma}$</td>
<td>0.37</td>
<td>0.39</td>
</tr>
<tr>
<td></td>
<td>$T_{\text{any}}$</td>
<td>3.31</td>
<td>3.07</td>
</tr>
<tr>
<td></td>
<td>$T_{\sigma}$</td>
<td>1.53</td>
<td>1.49</td>
</tr>
</tbody>
</table>

**Table B2** Comparison of the pure GA with a GA/SA mixture.

The results reported in [5] approximately correspond to what is here called the pure GA, and they are comparable to the best results published. It is therefore likely that the area shown in Table B2 obtained by the GA.
Figure B16 *Ratio of attempted mutations which are actually accepted.*

\(\alpha = \gamma = 1\). Q and the remaining parameters as before. In other words, it is the SA controlled mutations that allow the population size to be reduced while at the same time maintaining coverage. On the other hand, fixing the population size (\(\beta = 1\)) while increasing an increasing number of SAtuned mutations leads to an infeasible 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 B17: *Effect of reducing the population size in a otherwise pure GA process.*
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 pre-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%.

Average acceptance probability.

The decrease of the population size and the increase of the number of SA control ed mutations are both important contracts of the optimization process. Reducing the population size in an otherwise pre-GA process will cause divergence, as illustrated in Fig. B17. These graphs stem from a simple execution of SA with the parameters $P = 1 - 10^k$.
Figure B12 Population size as a function of generation number.

Figure B13 Estimated areas of the average and best individuals as functions of computational work.

tion B3.2). The quality is approximately proportional to the actual effort required.

For each generation, the average number of attempted mutations per individual is shown in Fig. B14. 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 B4.2. Therefore, the average value of the acceptance probabilities $P_{sp}$ (as defined in Section B4.2) in the population decreases with time as illustrated in Fig. B15. The ratio of attempted mutations, which are accepted and performed, is shown in Fig. B16 as a function of generation number.
mixed runs. The graphs are extracted from the same sample execution of the algorithm using the parameter values $M_0 = 5$, $S = 20$, $p_{in} = 0.05$, $\beta = 0$, $\gamma = 1$, $R = 80$, $\mathcal{P} = 0$, $\Omega$, $\alpha = 0.6$, and $\lambda = 1$.

![Graph](image)

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

Fig. B.11 show for each generation the average of the estimated area of all individuals and the estimated area of the best individual. Both quantities improve 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 behavior is typical for a pre GA in case of which no further improvement should be expected. However, due to the SA protocol of this algorithm new significant improvements are obtained from generation 800 to 1,000. The very best individual emerges in generation 1,000 and the process terminates after 1,200 generations.

The population size $M$ decreases as shown in Fig. B.12. For generation 1,000 it equals 1 and the process becomes pre 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 estimated area as function of computational workload for practical reasons, work is measured here as the number of channel duties measured during decoding (see Sec-
\textbf{B.6 Experimental Results}

In this section, experimental results obtained with an implementation of the application of SGA to a real-world problem are reported. In Section B6.1, the behavior of SGA when executed in a real GSA role is investigated. Section B6.2 compares the performance of the pre-GA role with the real GSA role, and in Section B6.3, the performance of SGA is compared to the best results obtained by other placement algorithms found in the literature.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Gls</th>
<th>Ns</th>
<th>Thms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ave</td>
<td>9</td>
<td>F</td>
<td>2F</td>
</tr>
<tr>
<td>Nsix</td>
<td>10</td>
<td>2B</td>
<td>6B</td>
</tr>
<tr>
<td>Tp</td>
<td>11</td>
<td>8</td>
<td>3D</td>
</tr>
</tbody>
</table>

\textit{Table B1: 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 DEC VAX 5000 workstation. Performance is measured using three benchmarks from the \textit{ISI Intentional Wish} on Placement and Routing. Table B1 lists the main characteristics of these examples.

\textbf{B6.1 Behavior in Mixed Mode}

The most interesting execution mode of SGA is the mixed GSA role, which also covers the most complex behavior. Figures B11 through B16 illustrate the typical optimization process obtained in the
and updated as described in Section B42 using the corresponding state.

deviation of area and internode length $\tilde{\sigma}$ and $\tilde{\sigma}$, respectively.

The parameter $P_{s}$ (and $\mathcal{P}_{s}$) is used for both procedure optimisation.

The probability $P_{\text{acc}}(s, r)$ of accepting the mutation $r$ is given as

$$P_{\text{acc}}(s, r) = \begin{cases} 
\exp\left( \frac{C_{a}(r) - C_{a}(s)}{T_{x}} \right) & \text{if } C_{a}(r) > C_{a}(s) \\
\exp\left( \frac{C_{w}(r) - C_{w}(s)}{T_{x}} \right) & \text{if } C_{a}(r) = C_{a}(s) \land C_{w}(r) > C_{w}(s) \\
1 & \text{otherwise}
\end{cases}$$

The implementation of the local hillclimb routine $\text{optimize}(t)$ 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 $\text{optimize}(t)$ 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 phenotype. As a result in Section B2, the purpose of inversion is to optimise 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 nodes are violated and so that the corresponding phenotype is still the same. An example of this is shown in Fig B10. The genotype tree is generated by moving the subtree rooted at $b$ in the genotype shown in Fig B7.
2. After 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 from being exchanged due to the constraint that any node always has a higher priority than its predecessor. An example is shown in Fig B9.

3. After $\pi$ 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 deleted 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 deleted, while mutations 2 and 3 require deleting from priority $\min(\pi(b_i), \pi(b_j))$. Mutation is only performed if it does not cause any constraint violations.

Before mutation

After mutation

**Figure B9** 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 B5.3, the cost of an individual $s$ can be suitably expressed in a single number $C(s)$. Therefore, the acceptance criterion for mutations shown in Fig B6 has to be modified slightly for this application. Let $C_a(s) = a(s)$ and $C_w(s) = w(s)$ denote the estimated area and total interconnect length of $s$, respectively. Each individual has two separate parameters, $T_a$ corresponding to area and $T_w$ corresponding to interconnect length. These are defined.
$E_\theta$ is constructed as follows. For the cell tree of $\phi$, a rooted subtree $T' = (V', E)$, $V' \subseteq V$, $E \subseteq E_\phi$ is drawn $T'$ is drawn at random subject to the constraint that decoding $T'$ in the order defined by $\pi$, i.e., using $b \in V'$ \ $\forall b \in V \setminus \{ b \} : \pi(b) < \pi(b')$ as root, causes no constraint violation. The size of $V'$ is obtained by a normal distributed random variable having mean $n/2$ and standard deviation 1.

In Fig. B8 the drawn $T'$ is indicated by the dashed line. Initially $E_\phi$ is diffused to be $E_\phi'$. Here, $\theta$ has inherited all cells in $V'$ from $\phi$. The remaining cells $V - V'$ are then inherited from $\theta$ by extension of $E_\phi'$. The cell tree of $\psi$ is traced in accordance to $\pi$ according to $\pi_\psi$. Any node it is decided if the corresponding cell $b$ belongs to $V'$, that is, whether it has been placed in $\theta$ already. If so, the cell is skipped. Otherwise, $b$ is added to the cell tree of $\theta$ by extending $E_\phi'$. 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. $\pi_\theta$ is uniquely defined so that it corresponds to the order in which the cells were placed when creating $E_\phi'$.

### B5.5 Mutation Operator and Hillclimber

The implementation of the qeator mut at e of Fig. B3 performs for different types of randomizations 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. After 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 $o : V \to \{0, 1, 2, \ldots, 7\}$.

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

### 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 $\Phi$ be the set of all possible individuals for a given instance of the problem having $n$ cells $b_1, \ldots, b_n$ and let $F : \Phi \to \mathbb{R}$ denote the fitness measure. Since the objective is to minimize the area, initially $F$ is defined as

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

where $B_s$ is the outer rectangle of the individual $s$ and $A(x)$ is the area of rectangle $x$. That is, $F(s)$ is the inverse of the total estimated nesting area in $s$. All individuals having equal area will now have equal fitness. By minimizing the total area, a placement, the probability of a 100% nesting completion within the estimated area is likely to increase as the total interconnect length decreases. To minimize 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 ensures that area is still the primary optimization criterion i.e., smaller area will always receive higher fitness. Finally, the adjusted fitness values are scaled linearly as described in [4] in order to control the variance in the population. For a detailed description of the fitness computation the reader is referred to [5].

### 5.4 Crossover Operator

Given two individuals $\phi$ and $\psi$, the crossover operator generates a feasible offspring $\theta$. This operation is illustrated in Fig. B8. Throughout this section, a subscript specifies which individual the plotted property is a part of.
The genotype of an individual having \( n \) cells \( b_1, \ldots, b_n \) is described. An example genotype with \( n = 7 \) cells is shown in Fig. B7 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 \( b_i \). 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.

\[
\begin{align*}
&b_6 \quad b_5 \quad 6 \\
&b_6 \quad b_2 \quad 4 \\
&b_4 \quad b_1 \quad 2 \\
&b_7 \quad 7
\end{align*}
\]

**Figure B7:** An example genotype and the corresponding phenotype.

Let \( e_{ij} \in E \) denote an edge from \( b_i \) to \( b_j \), and let \((b_{1i}^d, \beta_{1i}^u)\) and \((b_{2i}^d, \beta_{2i}^u)\) 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. It is,

\[
\forall e_{ij} \in E : e_{ij} \in E_t \Rightarrow b_{ij}^d \geq b_{ij}^m, \quad e_{ij} \in E_r \Rightarrow b_{ij}^d \geq b_{ij}^m.
\]

The tree is drawn as follows. The cells are placed one at a time in a rectangular array 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 rating area estimate, which is computed as each cell is placed. The estimate is based on the computation of density estimates, and is described in detail in [5]. The cells are placed in ascending order according to their priorities, which are defined by the mapping

\[
\pi : V \rightarrow \{ 1, \ldots, n \}.
\]

Any node has higher priority than its predecessor in the tree. In Fig. B7, the priorities are indicated at the top right hand side.
B5.1 Problem Definition

The macrocell 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 list specifying the interconnections of all terminals.
- An approximate horizontal length \( W \) of the dipole construction.

Given

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

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.

From the last constraint, the necessary routing area is estimated during the placement. The estimate is based on the assumption 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.

B5.2 Generic Encoding

The generic encoding of a macrocell placement is based on a generalization of the two-dimensional binpacking problem. The standard bin-packing algorithm places the bins in the linear at a time at the bottom 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 the routing area estimate. The solution space considered by the algorithm is restricted to the set of all possible \( BL \)-placements.
Fig. B6 Structure of the routine $SA\text{mutate}(s)$.

B4.3 GA and SA as Special Cases

$SA$ reduces to a pre $GA$ when $M_{_{0}} > 1$, $R = S$, $\alpha = 1$, $0$ and $P$ is close to 1.0. The $SA$s 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($x$, $y$) always yields the offspring $x$, in which case the reproduction step becomes equivalent to the empty statement 1.

B5 Application to Macro-Cell Placement

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

1Alternatively, the generation of $\Pi_n$ can be conditioned by $M > 1$, as can the invocation of the inversion operator, if desired.
The population size $M$ after $c_R$ iterations is

$$M = \min(\text{rand}(\beta^{c_R M_0}), 1, 0)$$

where $M_0$ is the initial population size, $0 \leq \beta \leq 1$ is a real-valued parameter, and $\text{rand}(x)$ returns a random number to the nearest integer value of $x$.

Initially when $M = 1$, corresponding to a pre-SA process.

When $M$ is decreased, the $M$ fittest individuals are kept, while the rest are discarded. Furthermore, the mutation rate $p_{\text{mut}}^{c_R}$ is increased so that after $c_R$ increases, it is given by

$$p_{\text{mut}}^{c_R} = \min(\gamma^{c_R p_{\text{mut}}^{c_R}}, 1, 0)$$

where $p_{\text{mut}}^{c_R}$ is the initial mutation rate, and $\gamma > 1$ is a real-valued parameter. Finally, notice that mutations are now performed by the routine $SAmutate$, which will be discussed in the following section.

### B4.2 SAmutate

Mutation of individual $s$ is performed as illustrated in Fig B6. The routine $mutate$ of Fig B3 is used to generate a random value of $s$. If this is the first mutation of $s$, its variables $P_s$, $T_s$ and $c_s$ controlling its coding schedule are defined. Then the mutation is performed with a temperature dependent probability as in SA. It may be noted that this schedule replaces the SA outline of Fig B4. However, the temperature decrease is not computed in a slightly different way which will be explained below.

The absolute value of the 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.

Specifically let $P_s$ be the probability of accepting a mutation on $s$, which increases the cost of $s$ by $\sigma$, the standard deviation of the cost of all solutions in the search space. From initial value $P_s$, $0 < P_s < 1$, $P_s$ is then reduced by a factor $\alpha$, $0 < \alpha < 1$, whenever quasi-equilibrium is maintained. For a given value of $P_s$, the corresponding temperature $T_s$ is computed as

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

where $\sigma$ is an estimate of $\sigma$ computed during generation of the initial population.
\texttt{genrate}(\Pi_c) ; \\
\forall s \in \Pi: T_s = \bot ; \\
\texttt{evalute}(\Pi_c) ; \\
q = \texttt{best}(\Pi_c) ; \\
c_R = \emptyset \\
\texttt{repeat until stop?}() : \\
\quad \text{if important for } R \text{ genrate } \emptyset : \\
\quad \quad c_R = c + 1 ; \\
\quad \quad M = \texttt{rand}(\beta, c R M, 1) ; \\
\quad \quad \Pi_c = \texttt{red} \left( \Pi_c, M \right) ; \\
\quad \quad p_{\text{nt}} = \texttt{inv}(\gamma, p_{\text{nt}}, 1, 0) ; \\
\quad \text{end} \\
\Pi_n = \emptyset ; \\
\texttt{repeat } M \text{ times} : \\
\quad \texttt{select } s \in \Pi_c, t \in \Pi_t ; \\
\quad \quad v = \texttt{cross} (s, t) ; \\
\quad \quad T_v = \bot ; \\
\quad \quad \Pi_n = \Pi_n \cup \{v\} ; \\
\quad \text{end} \\
\texttt{evalute}(\Pi_{c \cup \Pi_n}) ; \\
\Pi_c = \texttt{red} \left( \Pi_{c \cup \Pi_n}, M \right) ; \\
\forall s \in \Pi: s = \texttt{Shake}(s) ; \\
\forall s \in \Pi: \text{ with prob } p_{\text{inv }} \emptyset : \\
\quad s = \texttt{inert}(s) ; \\
\texttt{evalute}(\Pi_{c}) ; \\
\quad q = \texttt{best}(\Pi_{c \cup \{q\}}) \\
\text{end} \\
\forall t \in \Pi \cup \{q\} : t = \texttt{optimize}(t) ; \\
r = \texttt{best}(\Pi_{c \cup \{q\}}) ;

Figure B3 Outline of SAGA
B 4 The Unified Algorithm

The unified algorithm SQA can be presented. It can be viewed as a GA which has been unified into two major parts, 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 cooling schedule.

2. Initially, SQA executes as a pure GA like the Estagrates, as illustrated in Fig. B1. SQA gradually switches over to SA. The speed of this switch is adaptive, since it is determined by the progress of the optimization itself.

SQA 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 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 Bards SA

Fig. B5 gives an overview of SQA. By copying it to Fig. B2, it can be seen that only few things have changed. The temperature of individual $s$ is denoted $T^c_s$, and $\bot$ denotes the unified value. This, every new individual, generated in the initial population or as a result of crossover, has an unified temperature.

The switch towards SAs is handled by the if-statement which initiates each generation. A step towards SAs 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 $\pi/m$. In other words, one SA-controlled mutation 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].

```plaintext
start();
T = T;
repeat until stop criterion:
    c_s = 0
    while c_s < λ - k):
        q = mutate();
        with probability exp(\( \frac{\Delta s(q)}{T} \)), accept q;
        c_s = c_s + 1;
    end
    T = α T;
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 \( s \). The function `mutate` of Fig. B3 is used for this, assuming that \( p_m \) is sufficiently high. Each change is accepted or rejected with a probability that depends on the temperature \( T \). The temperature is gradually decreased according to the formula \( T_{n+1} = α T_n \), until it starts from an initial temperature \( T_0 \). At each fixed temperature, a sequence of changes are performed on \( s \) until a quasi equilibrium state is reached. In Fig. B4, the temperature is reduced each time \( λ \) changes. If all changes 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 \( \exp(\frac{\Delta s(q)}{T}) \), which decreases with \( T \).
internal genotypes, while fitness is defined in terms of phenotypes. For a given genotype, the corresponding phenotype is computed by a decoder. An introduction to genetic algorithms is given in [6].

Fig B2 shows a template for the genetic algorithm. Initially, the current population \( \Pi \) is constructed from randomly generated individuals. Routine \textit{evaluate} computes the fitness of each of the given individuals, while \textit{best Q} finds the individual with the highest fitness. The creation of the outer "repeat" loop corresponds to the simulation of one generation. Throughout the simulation, \( M = \Pi \) is kept constant. Keep track of the best individual \( q \) ever seen. Routine \textit{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 \( \Pi' \) of size \( M \). Two mates \( s \) and \( t \) are selected independently of each other, and each mate is selected with a probability proportional to its fitness. Routine \textit{reduce} \((\Pi, k)\) returns the \( k \) fittest individuals from \( \Pi \) thereby keeping the population size constant.

\begin{verbatim}
\forall \text{genotypes } g \ 1, \ g, \ldots, \ t \ \text{of } \Pi:
\text{with prob } p \ \text{mut} \ \text{do:}
\text{alter } g \ k \ \text{randomly}
\end{verbatim}

Fig. B3 Structure of the routine \textit{mutate}(t).

As illustrated in Fig. B3, the mutation operator \textit{performs pointwise mutation with a given probability on each of the genotypes, or genes, of its argument. The genetic operator \textit{invert}(t) alters the genotype of } t \text{ without altering the corresponding phenotype. As described in [6], the purpose of this operator is to optimize the relative position of the genes of } t \text{ with respect to the crossover operator. Finally, local hillclimbing is performed on all existing individuals by routine \textit{optimize}(t). It is common practice to apply this hillclimbing in a GA in an attempt to slightly improve the final solution [6]. If the solution is the output of the algorithm.}
less fit individuals tend to die at. This survival-of-the-fittest Darwinian principle is the basic idea behind the GA.

```
generate(\Pi \cup_c);  
edulate(\Pi \cup c);  
q =best(\Pi \cup_c);  
repeat until stagnation():
  \Pi_n =\emptyset ;
  repeat M times:
    select s \in \Pi \cup c, t \in \Pi;
    v =crossover(s , t );
    \Pi_n =\Pi_n \cup \{ v \};
  end
  edulate(\Pi \cup c \cup \Pi_n);  
  \Pi_c =reduce(\Pi \cup c \cup \Pi_n, M);
  \forall t \in \Pi: t =mate(t );
  \forall t \in \Pi: \text{with prob p inv ch}:  
    t =inert(t );
  edulate(\Pi \cup c);  
  q =best(\Pi \cup c \cup \{ q \} )
end
\forall t \in \Pi \cup \{ q \} : t =optimize(t );
r =best(\Pi \cup c \cup \{ q \} );
```

Figure B2 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 operators of this process are crossover, with minor propagation, and mutation, with minor the random changes occurring in nature. After a number of generations, the fittest 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 EP. One of the strategies is called life cycle. Application of individuals exist. Mutations are accepted with a certain probability as in SA but individuals go through a life cycle. As it gets older, its probability of being mutated decreases while the probability of mating increases. Beauk and Helling report that the life cycle strategy is superior to pure SA on the EP.

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

- To provide a novel algorithm that 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 microcell placement problem. It is experimentally shown that a mixed strategy performs better than a pure GA. Furthermore, using the mixed strategy on NCM microcell placement benchmarks, we obtain results capable of or better than previously published results.

The rest of this paper is organized as follows. In Sections B2 and B3, the concepts of GA and SA are briefly introduced. The unified algorithm is then discussed in detail in Section B4. The discussion in Sections B2, B3, and B4 is application independent. Section B5 describes the application of the unified algorithm to the microcell placement problem. This includes a brief description of the application-specific genetic encoding and corresponding operators. Finally, experimental results are described in Section B6, and a conclusion is given in Section B7.

### B.2 The Genetic Algorithm

The concept of genetic algorithms introduced by John Holland [7] at 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 number, 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. B1. Initially, the cost of the solution 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. One approach is to unify the GA with the simulated annealing algorithm (SA), another well-known, high-performance optimization technique. While SA is generally able to obtain improvements also into the later phase of the process, it does not converge as fast as the GA in the initial phase. The unified algorithm called SGA is designed in such a way that the advantages of the GA as well as the SA are utilized.

![Figure B1: The typical convergence of a GA](image_url)

Earlier attempts to combine GA and SA have been made. With the GA as the starting point, Srag and Waser have incorporated elements from SA with the objective to improve control of population variance [10]. This is achieved by a so-called dynamic quencher, in which the number of destroyed/preserved states are controlled by a temperature-dependent stochastic variable. The degree of destruction increases as the global temperature is decreased. However, this approach is limited to ordering problems like the Traveling Salesman Problem (TSP). An alternative general approach is presented in [3].
Appendix B

SAA: Miro-Cell Placement by a Unification of the Genetic Algorithm with Simulated Annealing


Abstract

In this paper, a stochastic optimization algorithm called SAA is presented, which is a generalization of the genetic algorithm and the simulated annealing algorithm. Depending on the settings of its control parameters, SAA acts as a genetic algorithm, a simulated annealing algorithm, or a controllable mixture of these. SAA represents an application-independent approach to optimization and the resulting search process is highly adaptive. The performance of the approach on the miro-cell placement problem is evaluated. It is empirically shown that a mixture of the genetic algorithm with simulated annealing yields better layout qualities than a pure genetic algorithm. Furthermore, layout qualities obtained by SAA on MCNC benchmarks have been observed to be comparable to or better than previously published results.
[10] Paolo Micalle, Ian East, “An Ingot of Several Parallel Genetic Algorithms, Tools and Techniques for Transputer Applica-
tions”, pp 606, ICPS Res, 1990

a Network of Workstation”, IEEE Transactions on Computer-Aided
Design, Vol. 12, No. 9, pp 1324-1336, Sept. 1993

[12] H Mablin, MG Geys-Siller, O Leu, “Efficient algo-
rithm in abitacal minimization” Parallel Computing, Vol. 7,
pp 60-85, 1988

Bond Placement for Echard Bok Late”, Proceedings of The

lem”, Proceedings of The 17th Design Automation Conference,
pp 42-44, 1990

Cell Placement using Genetic Particle Optimization,” IEEE
Transactions on Computer-Aided Design, Vol. 9, No. 5, pp 509-511,
May 1990


Standard Cell Placement”, Proceedings of The European Design Au-
tomation Conference, pp 609-614, Mch 1990

for Med Miro Cell and Standard Cell Layouts”, Proceedings of The
Bibliography


A 5 Conclusion

In this paper a genetic algorithm for the macrocell placement problem has been presented. The algorithm is based on a different approach and 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 has been avoided.

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

The main factor setting the limits of the applicability of the algorithm is the time consumption. Average, the current implementation requires 52 CPU-ints for the atea benchmark, 55 CPU-ints for lp and 156 CPU-ints for xbn. The computation times require an added effort for experiments with larger benchmarks like xbb and xn. Furthermore, in order to rule the runtime captive to that of other systems, it needs to be reduced by a factor of 3 to 10. However, there are two reasons why it is believed that such a significant reduction is indeed attainable.

Firstly, in the current experimental implementation of the algorithm, the majority of the total runtime is spent running dual subroutines during decoding. For an example such as the lp benchmark, the algorithm takes between 10 and 30 billion dual subroutines. When an immediate fix is needed, it is expected that running an information monitor in such a large problem would entail updating the individual cells to the previous state. Therefore, a data structure which allows a dual subroutines 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. Of the characteristics of GA in general is that a high speedup can be expected on any
<table>
<thead>
<tr>
<th>Method</th>
<th>System</th>
<th>Area</th>
<th>Width</th>
<th>Vas</th>
</tr>
</thead>
<tbody>
<tr>
<td>qte</td>
<td>Tis work</td>
<td>53.9</td>
<td>563</td>
<td>7.0</td>
</tr>
<tr>
<td></td>
<td>B[13]</td>
<td>54.0</td>
<td>49</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Settle Sli on [18]</td>
<td>54.7</td>
<td>3.0</td>
<td>-</td>
</tr>
<tr>
<td>lp</td>
<td>Settle Sli on [18]</td>
<td>11.3</td>
<td>20.0</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Tis work</td>
<td>11.5</td>
<td>20.2</td>
<td>6.7</td>
</tr>
<tr>
<td></td>
<td>B[13]</td>
<td>12.15</td>
<td>27.8</td>
<td>-</td>
</tr>
<tr>
<td>eceo</td>
<td>Settle Sli on [18]</td>
<td>23.7</td>
<td>61</td>
<td>110</td>
</tr>
<tr>
<td></td>
<td>B[13]</td>
<td>23.17</td>
<td>68</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>Tis work</td>
<td>23.58</td>
<td>55.6</td>
<td>137</td>
</tr>
<tr>
<td></td>
<td>B[13]</td>
<td>24.4</td>
<td>63.2</td>
<td>8.7</td>
</tr>
<tr>
<td></td>
<td>MADO 1</td>
<td>29.1</td>
<td>60</td>
<td>117</td>
</tr>
<tr>
<td></td>
<td>VAL 1</td>
<td>31.17</td>
<td>86</td>
<td>109</td>
</tr>
</tbody>
</table>

Table A4: Comparison of quality with other systems. A hyphen indicates that the value is not available.

### A4.4 Gene Pool Rate

Fig. A8 show 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 solution. This, when a large number of individuals in the population are relatively close to the optimum solution, further progress becomes very slow. This is the typical behavior 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 runs 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.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>$A_{best}$</th>
<th>$A_{avg}$</th>
<th>$A_{\sigma}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>qte</td>
<td>53.9</td>
<td>51.9</td>
<td>1.2</td>
</tr>
<tr>
<td>xerox</td>
<td>26.8</td>
<td>21.1</td>
<td>1.5</td>
</tr>
<tr>
<td>lp</td>
<td>11.9</td>
<td>12.8</td>
<td>0.4</td>
</tr>
</tbody>
</table>

Table A3 Variation in result quality

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

![Figure A7: The lp benchmark before and after routing.](image)

In Table A4 the layout quality obtained is compared to the best published results. Again, the absolute area is one area of the completed layout in mm$^2$. The total wire length in mm and the total number of vias in each layout is also listed. The results referenced should be copied with some caution due to minor variations in the definitions used.
A first sight this may seem surprising since this particular configuration represents the highest degree of randomization among all strategies tested. However, as described in Section A3, 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 outperforms this partially degenerate 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 $\beta_2a$ and $\beta_2b$. Consequently, instead of improving the efficiency of the search, the greedy strategies actually prevent really good solutions from being found.

### A4.2 Pranter Settings

For all examples considered the same set of pranter values has been used to control the GA, i.e., no problem-specific tuning has been performed towards each benchmark. The values used are $M = |P|$, a mutation probability of 0.05 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 $p \in P$ 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 A3.1.2 depend on the characteristics of the interconnections to be routed, i.e., these parameters are problem-dependent. Table A2 shows the values used.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>afe</th>
<th>oco</th>
<th>lcp</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>0.0</td>
<td>0.2</td>
<td>0.0</td>
</tr>
<tr>
<td>$b$</td>
<td>10.0</td>
<td>100.0</td>
<td>7.0</td>
</tr>
</tbody>
</table>

**Table A2**: Values of parameters for routing area estimate
\[ \alpha_1: \mid V \mid \text{ is chosen uniformly and at random so that } V_{\text{min}} \leq \mid V \mid \leq V_{\text{max}}, \]

where \( V_{\text{min}} \) and \( V_{\text{max}} \) are user-defined limits. These limits are chosen so that the interval \([V_{\text{min}}, V_{\text{max}}]\) 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.

\[ \alpha_2: \mid V \mid \text{ is determined as the rounded value of a normal distributed statistic variable having mean } n/2 \text{ and standard deviation 1.} \]

Therefore, in the large majority of all cases, \( |V| \leq \sqrt{s} \) 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 \( \beta \) must be added to the tree of \( \gamma \) by extending \( E \rightarrow V \). Three different strategies have been tried for determining the position at which to add the node.

**\( \beta \textbf{1} \):** Choose a free position at random among all free locations.

**\( \beta \textbf{2} \):** Add \( c \) at a position which will probably lead to a high fitness of \( \gamma \).

Determine this position by evaluating all free positions according to some quality measure, add \( c \) at the best scoring position by applying an iterative search. Three different quality measures have been tried.

**\( \beta \textbf{2a} \):** Pick \( c \) at the position corresponding to the lowest possible position at phenotype level.

**\( \beta \textbf{2b} \):** Pick \( c \) at a position which 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. With \( \beta_2a \) or \( \beta_2b \) is used monotonic charge 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 improved 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 algorithms has been implemented in the C
programming language and runs on a DEC 1000-20 workstation.
Approximate size of the source code is 14,000 lines. The performer
has benchmarked on three machines from the 1984 NCITS and
WISP at Princess and Rating. The AI lists the main characteristics of these examples. Rating and execution of the layouts have
been performed by using the Mcintosh set [1] which is part of the
Codiads CPFinneck.

<table>
<thead>
<tr>
<th>Pathname</th>
<th>Cells</th>
<th>Nts</th>
<th>Trinds</th>
<th>I/O rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>ape</td>
<td>9</td>
<td>27</td>
<td>27</td>
<td>73</td>
</tr>
<tr>
<td>xerox</td>
<td>10</td>
<td>20</td>
<td>68</td>
<td>2</td>
</tr>
<tr>
<td>lp</td>
<td>11</td>
<td>85</td>
<td>39</td>
<td>4</td>
</tr>
</tbody>
</table>

Table AI: Benchmark characteristics. The number of terminals includes
the io-terminals.

A 4.1 Experiments with the Crossover Operator

In most GNP implementations crossover is a random operation in the sense
that the parent from which a given feature is inherited is always deterministically. However, in some GNP implementations, a kind of local
optimization is performed as an integrated part of the crossover operation. Instead of just copying the features of two given individuals in
a completely randomized fashion, an attempt is made to assimilate the
features in such a way that highly fit offspring emerge. 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-aggregated operators can be found in Chapter 5 of Gldberg's
book [6].

Experiments with six different variants of the crossover operator de-
scribed in Section A3.3 have been performed. These represent various
degrees of local optimization. To determine the size of the connected sub-
set $V_{s}$, inherited from the first parent $\alpha$, two strategies have been tried
is not controllable. This can usually be obtained easily \([t]\), and here the
decoder already has this property as outlined in section A3.1.

![Diagram](image_url)

**Figure A6** Another genotype for the phenotype in Fig. A2.

The insertion 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. A6. The genotype tree is generated by moving the
subtree rooted at \(c_2\) in the genotype shown in Fig. A2.

### A3.6 Stop Criterion

The evolution process is terminated when no improvement has been
made for a certain number of consecutive generations, denoted
by \(s\). When determining if improvement has occurred, we consider
the best existing individual as well as the average individual, and we
compare individuals in accordance with an predefined criterion of
minimum area and secondarily minimum total interconnect length.

The specific, denote by \(A_{\text{avg}}(t)\) and \(A_{\text{best}}(t)\) the average and best area
of the individuals present in generation \(t\), respectively. Similarly, let \(L_{\text{avg}}(t)\)
and \(L_{\text{best}}(t)\) denote the average and best total estimated interconnect
length present in generation \(t\). Thus by definition, an improvement has
occurred in generation \(k\), if and only if for some \(x \in \{a,v,g,b,s,t\}\),

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

or

\[
A_x(k) = \min \{ A_x(t) \mid k - \delta \leq t < k \} \
\wedge (k \leq \min \{ L_x(t) \mid k - \delta \leq t < k \})
\]

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

A3.5 Inversion Operator

In a typical GA as presented in [5], 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 $\alpha$ (see Fig. A4). The smaller the distance between two given nodes is, the more likely it is that both nodes will belong to the connected subtree $T_s = (V_s, E_s)$, and hence that both nodes will be inherited by the offspring. Since $T_s$ is not altered during crossover, the two nodes will outline to be close in the offspring.

If the closer nodes represent a good placement and contribute significantly to a high fitness of the individual, then the above property of the crossover operator is beneficial. 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 creates a new genotype by rearranging the components in such a way that their mutual distances change, 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...
$p^\gamma$ should correspond to the order in which the cells were placed when creating $E^\gamma$. Since $p$ is a bijection, the following constraints uniquely determine $p^\gamma$:

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

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

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

### A3.4 Motion Quotas

For different motion quotas exist. Each of these prefers some random device in the given genotype. Let $c_i$ and $c_j$ denote two randomly chosen cells, $i \neq j$. The four quotas 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 chosen as part of the move.

2. Alter the set of edges $E$ by editing $c_i$ and $c_j$. The priorities of the cells are edited simultaneously so that no pair of cells are replaced a priori from having been deleted due to the constraint that any cell always has a higher priority than its predecessor. An example is shown in Fig A5.

3. Alter $p$ by editing $p(c_i)$ and $p(c_j)$.

4. Change the transformation of a cell by altering the value of $t(c_i)$.

When performing each of these motions, a part of the genotype has to be decided to check if the mutated individual satisfies all constraints. Motions 1 and 4 require that all cells having priority $p(c_i)$ or higher are decided while motions 2 and 3 require deciding from priority $\min(p(c_i), p(c_j))$. A mutation is only performed if it does not cause any constraint violations.
A3.3 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 A41. Given two individuals $\alpha$ and $\beta$, the crossover operator generates an offspring individual $\gamma$, the descendent of $\alpha$ and $\beta$. This operation is illustrated in Fig. A4. Throughout this section, a superscript specific to individual the marked property is a part of.

![Figure A4: Combining $\alpha$ and $\beta$.](image_url)

$E^\gamma$ is constructed as follows. First the cell tree $\alpha$ is counted subset $T_s = (V_s, E_s), V_s \subseteq V, E_s \subseteq E^\alpha$ is drawn $T_s$ is drawn at random but subject to the constraint that drawing $T_s$ in the order defined by $p^\alpha$, i.e. using $c \in V_s \setminus \{c\}$, $\forall c \in V \setminus \{c\}$, $\Pr(c) < p^\alpha(c')$ as root, causes no constraint violation. To different schemes have been tried for the determination of the size of $V_s$, as will be described in section A41. In Fig. A4, the drawn $T_s$ is indicated by the dashed line. Initially $E^\gamma$ is defined to be $E_s$. Here, $\gamma$ has inherited all cells in $V_s$ from $\alpha$. The remaining cells $V - V_s$ are then inherited from $\beta$ by extension of $E$. $E^\gamma$.

The cell tree of $\beta$ is traced in ascending order according to $p$, and if it is dashed if the corresponding cell $c$ belongs to $V_s$, then it is placed in $\gamma$ already. If so, the cell is skipped otherwise, $c$ is added to the cell tree of $\gamma$ by extension of $E$. Various schemes for determining the position at which to add cell $c$ has been tried, see section A41. The transformation of any cell is inherited unaltered together with the cell itself. That is,

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

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

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 $F'(p_i) = F'(p_{i+1}) = \ldots = F(p_j)$, $i < j$. Thus the fitness of $p_i, \ldots, j$ may be adjusted according to interconnect length.

In order to ensure that area always predetermines interconnect length, this is done as follows. Sort $p_i, \ldots, j$ into decreasing order according to interconnect length, i.e., let us assume $L(p_{i+1}) \geq L(p_i) \geq \ldots \geq L_j(p)$.

Define $F_{ij}$ 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

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

Since the fitness values modified can be very small, they are normalized. Finally, the values are scaled linearly as described in [6]. The purpose of scaling is twofold: A the initial phase of the evolution, a few individuals having very high fitness compared to the average will be very daunting. 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. More selection becomes almost random thereby reducing the chances of further improvement. At this stage, scaling counteracts this effect by increasing the standard deviation.
then apted as

\[ D_s = \begin{cases} 
\lambda \left[ d_s + r \cdot \text{und} \left( \frac{1}{\lambda} \right) + b \right] & \text{if } d_s > 0 \\
0 & \text{if } d_s \leq 0 
\end{cases} \]

where \( h_s \) is the length of side \( s \) of \( c \), \( \lambda \) is the spring in the rating grid \( r \) of \( d(x) \) is the rounded value of \( x \) and \( a \) and \( b \) are user-defined parameters. The area inside the solid rectangle shown in Fig. A3 is uniquely determined by \( D_s \). All \( c \) can be placed at the given position if and only if this area contains no (pairs of) cells apart from \( \text{space of } i \) itself.

When \( a = b = 0 \) the estimated rating area is a lower limit of the area needed by any rotor regardless of the domain definition. If \( a > 0 \) the corresponding term \( D_s \) increases with \( h_s \). The argument for this definition is that the larger the domain, the more likely its will pass through it [18]. Namely using \( c \) in any direction may affect the value of \( D_s \) for all values of \( s \).

In summary, given \( V \), the genotype of an individual consists of the relation \( E \) and the function \( p \) and \( t \). The genotype (and the decoder) has the important property of implicitly representing not constraints of the problem. This simplifies the design of genetic operators which ensures the satisfaction of all constraints at all times.

### A3.2 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 plant \( A \) is the area of a rectangular cell and \( R_p \) is the rectangle \( R \) of the individual \( p \). This is, \( F'(p) \) is the inverse of the total estimated rating area in \( p \). All individuals having equal area will receive equal fitness. If defining the total area of a plant, the probability of a 100% rating completion within the estimated area is likely to increase as the total interconnect length decreases. To minimize of the total interconnect length is therefore introduced as a secondary optimization criterion. Al
Any $H$-plane can be represented at least one genotype, i.e., the decoding mapping is not one-to-one. Therefore, since a $H$-plane can only be represented by a genotype if it is a $H$-plane, the search space explored by the algorithm is exactly the set of all $H$-planes. This is one of the significant differences to the approach in [3], in which the search space is restricted to sliding structures.

### A.3.1.2 Rating Area Estimation

When dealing the binary tree, the rating area need be estimated as each cell is placed. When placing the $i$-th cell, the distance within each direction $s \in S = \{\text{north, east, south, west}\}$ 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 strategy as close to the previously placed cells as allowed by $D_s$. Figure A3 illustrates how $D_s$ is computed.

![Figure A3: Estimation of routing area.](image)

When testing if cell $c_i$ can be placed at some given position $(x_i, y_i)$, the four areas indicated by dashed squares are considered $D_s$ depends on all terminals at side $s$ of $c_i$ and 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 an invalid cell. Given this set of terminals, the demand density $d_s$ is computed as if the square were the routing demand. In order to account for global routing $D_s$ is
\( E_t \cap E_r = \emptyset \). Each node has at most one outgoing 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 \( (c^d_i, p^d_i) \) and \( (c^m_i, p^m_i) \) denote the coordinates of the lower left and upper right corners of \( c_i \), respectively. The edge \( e_{ij} \in E_t \) \((E_r)\) means that cell \( c_j \) is placed above (to the right of) cell \( c_i \) in the phenotype. That is,

\[
\forall e_{ij} \in E : e_{ij} \in E_t \Rightarrow c^d_j \geq c^m_i, \quad e_{ij} \in E_r \Rightarrow c^d_j \geq c^m_i
\]

The tree is divided 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 made as far down and then as far left as possible without violating the rooting area estimate described in section A3.1.2. The cells are placed in ascending order according to their priorities defined by the one-to-one function \( p : V \rightarrow \{1, \ldots, n\} \). A node has higher priority than its predecessor in the tree. In Fig A2 the priorities are indicated at the top right hand side of each node. The transformation of each cell is defined by the function \( t : V \rightarrow \{0, 1, \ldots, 7\} \), which is also part of the genotype.

\[ \text{Figure A2: An example genotype and the corresponding phenotype.} \]

All cells are placed in the smallest rectangle, \( R \), enclosing all cells, is cut by the decoder. Each of the four sets of i/outputs are then unisonly positioned along the corresponding edge of \( R \), so that the ordering within each set is preserved. Finally the rectangle \( R \) is constructed by extending \( R \) until the rooting area estimate is satisfied along all edges of \( R \). Note that the genotype itself contains no explicit representation of the i/outputs and no absolute coordinates of the cells. At the time of computation of a phenotype from a given genotype the decoder determines these quantities.
The drawback of the second strategy is that it is more complex in nature and its associated genetic operators are slowing to the fact that they ensure 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 adopted in which some of the constraints are implicitly represented and therefore need not be considered by the genetic operators.

### A3.1 Genetic Encoding

In GA 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. *Adaptive* is used to compute the phenotype corresponding to a given genotype. Estimation of the rotating 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 boxes into a bin having fixed width and infinite height in such a way that the distance from the top edge of the largest placed block to the bottom edge of the bin is minimum. The standard algorithm for this problem places the blocks one at a time at the bottom and then at the leftmost position. The placement algorithm is based on a generalization of this scheme. For a given instance of the rare cell placement problem let a *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 rotating area estimate. The solution space considered by the algorithm is restricted to the set of all possible placements.

#### A3.1.1 Genotype and Decoder

Assume the given problem has $n$ cells $c_1, \ldots, c_n$. An example genotype with $n = 7$ cells is shown in Fig. A2 together with the corresponding phenotype. An 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 $E = E' \cup E_r$. 
simply defined to be the $M$ fittest individuals of $P$

Whose probability each individual in $P$

$c \cup P_N$. $c$ is not affected to a mutation. The main purpose of mutation is to ensure that information lost by reproduction can be recovered. In Section A34 for 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 purely converge to a local minimum. A mutation each individual is subject to a mutation query with a small probability. The purpose and the operating principle of this query is explained in Section A35. Each mutation is completely executed on all individuals as the basis for the selection to take place in the next generation. Initially, the best individual ever seen is updated.

With the last generation also simulated an attempt to optimize each of the individuals. $P \quad c \cup \{q\}$ a little further is made using routine optimize$(p)$, which creates a sequence of fitness prooptimizations on each individual. An intuitive strategy is used so that the optimize$(p)$ has been performed no single mutation can improve $p$ further. The best individual following the optimization then constitutes the resulting next mutation.

With 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. Ensuring 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 methods are the satisfaction of all constraints by the produced individuals. However, the fitness measure becomes 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 contribution regarding constraint violations are appropriately balanced at all times. On the other hand, the second strategy allows a single fitness definition, since no penalty terms are needed. This, we are guaranteed that the measure always expresses something meaningful.
\textbf{Figure A1: Outline of the algorithm}

Repeted selection and mating of individuals from $P$ and 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 \cite{1}, the scheme used here is stochastic sampling with replacement. That is, the individual $p_i \in P^C$ is selected with probability

$$\frac{F(p_i)}{\sum_{i \in P^C} F(p)}$$

where $F$ is the fitness measure, cf. Section A3.2. The two steps 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. A change to the selection scheme used for crossover, the selection pattern here is deterministic. The new $P^C$ is
A 3 Description of the Algorithm

The concept of the CA, which was introduced by Holland [8], is based on the idea of optimizing by simulating biological evolution. In nature, the individuals of 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, since their reproduction rate is high and their characteristics are inherited by their descendants. On the other hand, the less fit individuals tend to die out. This Darwinian principle of “survival of the fittest” can be used in optimization. Given a specific optimization problem, e.g., the Knapsack Placing 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 concepts of this process are reproduction and mutation, for which specific criteria have to be defined. 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 CA is given in e.g. [6].

Many variants of CA can be found in the literature [2, 4, 6, 7]. An outline of the CA used here is shown in Figure A1. After briefly presenting an overview of the algorithm, its various concepts are discussed in detail in the following sections.

Initially, the entire population $P_c$ is constructed from randomly generated individuals by random generation ($P_c$). The fitness of each individual is computed by evaluate ($P_c$), described in Section A3.2. Since the quality of an individual is relative to the rest of the population, computation of fitness requires the complete population as input. Runtime best ($P_c$) selects the best of the given individuals and is used throughout the process to keep track of the best individual 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 A3.6.

Reproduction initiates each generation. $M$ is simulated by the crossover operator described in Section A3.3. Given a pair of individuals, the crossover operator produces one offspring. The overall purpose of crossover is to ensure exploration of the promising parts of the search space. More the offspring produced here to resemble its parents. By
The more-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.

Apart from this 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 allowed by the technology.

The more-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 $O(n^4)$ 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 produce high-quality placements at the cost of extensive runtime.

Ales presented type of placement algorithms the genetic algorithm (GA). In [4, 5, 17] GA 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 runtime. To our knowledge only two papers have been published in which a GA for macrocell placement are presented [3, 4]. A will be accounted for in Section A2, both algorithms significantly differ from the approach presented here.

For the two-dimensional bin packing problem has been developed by Kier et al. [9]. The two-dimensional bin packing problem be seen as the hypothetical special case of the macrocell placement problem without nets exists, i.e. no routing will be performed. In this paper a GA for the macrocell placement problem is developed based on a comprehensive extension of the genetic encoding for genetic algorithms for bin [9]. The resulting algorithm is capable of placing placements having a quality comparable to the best published results.

A 2 Problem Definition

In the literature, the definition of the macrocell 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 diprimer construction. These terminals constitute the interface of the diprimer.

- A list specifying the interconnections between all terminals.

- An approximate horizontal length $W$ of the diprimer construction.
Appendix A

A Genetic Algorithm for MesoCell Placement


Abstract

A new genetic algorithm for mesocell placement is presented. The algorithm is based on a generalization of the two-dimensional bin-packing problem. The genetic encoding and the genetic operators ensure that all constraints of the problem are always satisfied. Consequently, the potential problem of achieving a quality target to the cost function is eliminated. The algorithms have been tested on real benchmarks and the layout quality obtained is comparable to the best published results.
Proc. of The Fourth International Conference on

[Wall 88] N H E Wall, K Hraghian Principles of CMOS
VLSI Design, Addison-Wesley Reading 1988


[Wit 92] P Witz, J MSith “Path-Distance Heuristics
for the Sein Ieferin Iidedued Novels,”


[Scott 89] S Scott, R N Ma, G Hadi, J K Qatari, (Ed.), “Introduction VLSI: Still Me Vis,” *by the Gengal Artists,” Report No WPI/SD 87/22, Center Science Division (HES), University of California, Berkeley, 1987

[Mann 81] S. Mann, P. Mandel, "Weins Sadad
Gel Parent on a Nerve of Wetstone," 
IEEE Transactions on Computer-Aided Design, 
Vol. 12, No. 9, pp. 132-136, Sept. 1983

[Mahin 8] H. Mahin, M. Grage-Slater, O. Fär, 
"Dition algorithma in ahlantaal optimiza-
tion" Parallel Computing, Vol. 7, pp. 6, 8, 1988

ization of a Usable NP-Complete," IEEE Transac-

[Nizska 8] Y. Nizska, M. Naia, A. Sajitski-Viartelli, 
"Vary: A New Approach to Micel Gel 
Design" Proceedings of the IFIP 10/WG10.5 In-
ternational Conference on VLSI, Mld, 1989

[Nusa 8] V. Nusa, "Dition y Algoritma in Mag-
netic Siani," Papers on Economics and Evolution 
#9303, Interdisiplinere Gildedakleg Universitats Gottingen, Germany, July 1988

[No 8] A E. No, M D. Ves, "Ming Cric Ar-
githma with Miv Gien," Annals of Math-
ematics and Artificial Intelligence, Vol. 5, No. 1, 
pp. 7988, 192

Engineering and Computer Sciences, University 
of California, Berkeley, 1988

[Odra 8] H. Odra, Y. Oga, K. Thau, "Bask 
and Bok Parent for Bask Bok Bok," 
Proc. of the 28th Design Automation Conference, 
pp. 43-43, 1981

P. Pat, "VIL Elly Atomic Parent 
Strategies for Vy Large Simulant Design," 
International Conference on Computer Design, 
pp. 43-43, 1988


[May 94] B A May, G Jinks, "A Test Rating"


[Ehren 91c] H Ehren, "Opting Non-Optim Shrines to the Seiner Pedersen a Graph using a Genetic Algorithm" Ethical report Iain B148, Aarhus University, Denmark, Feb 1991

dam "Evaluation of Genetic Algorithms in IBM Transactions on Computer-Aided
Design", pp 135-139, Vol. 8, No. 12, 1999

Sala, "Midjective Search in VLSI Design."
Proc. of The 7th International Conference on VLSI

[Laich 8] Y Laich, *Epistasis Variance - Suitability of a
Representation to Genetic Algorithms*, Thrid
Report 595-2, Te Wzam Institute of Sci-

[Laich 9] Y Laich, *An Intuitive Introduction to Genetic Al-
gorithms as Adaptive Optimizing Procedures*, Thrid
Report 596-7, Te Wzam Institute of Sci-
ence, Israel, April 1993.

[Laich 9] Y Laich, "A genetic algorithm applied to
robot trajectory generation" in L Laich, ed-
tor, *Handbook of Genetic Algorithms*, Chapter 12,

[Laish 8] L Laish, M Steenstrup "Genetic Algorithms
and Simulated Annealing: A Overview" in L Laish,
editor, *Genetic Algorithms and Simulated Anneal-

[Laish 9] L Laish, "Alighting Genetic Algorithms in Ge-
etic Algorithms", Proc. of the Third International

Nostad Reinhold 1993.

[T E Laish 9] T E Laish, J C Rupke, "Simulated Anneal-
ing: A New Approach for the Single Genetic
Algorithm", Proc. of the Fourth International Con-


[D Bailey 88] D Bailey, D R Hill, R R Mitin, "Rid-
ning
qstais in ahoioal poolon by onpine
ning" Proc. of the Fifth International Conference
on Genetic Algorithms, pp 49-57, 1989

for the
Steiner Problem in Graphs," Networks, Vol. 19

[J. E Bailey 90] J. E Bailey, "Ofllay: distributeing test pat-
entiallyelectrical, Journal of the Operational

[Baeiuk 91] T Baeiuk, Weding "Bltzam, Diva,
Hied-Strategies in Optimiztion Rllens"
Proc. of the 1st Workshop on Parallel Problem
Solving from Nature, pp 49-54, Springer-Verlag

[Baletter 91] M E Baletter, "Initialzation, rotation and se-
lection methods in genetic algorithms for function
optimization" Proc. of the Fourth International
Conference on Genetic Algorithms, pp 10-17,

[Bi 93] T N Bi, B R Min, "Near Approx-
sis for Genetic Algorithms," Proc. of the Fifth
International Conference on Genetic Algorithms,
pp 10-17, 1993.

[Krestin 82] M Krestin, "An placement/raging appro-
ch to ation of VSL las design" Proc. of the
1982 International Symposium on Circuits and Sys-

of Trilhie Clock Enen C6S Circuits," In
L Arell, M Tingen (Eds.), Nordic Transputer
Bibliography


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

Important topics of future work include a need to update the opti-
mization criteria and the way they are defined. Furthermore, the
accuracy of the aerial estimations should be improved which ultimately
leads to the consideration of simultaneous plant and global routing.
Judging from the characteristics of the problem, the general charac-
teristics 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, assuming that the main objective is to obtain the best possible layout quality. For specific questions of interest, these are formulated in Section 1.1, and in the following, the answers are summarized.

When comparing the performance of the developed algorithm to the best existing approaches using any other method, the GA is competitive with respect to layout quality. Consequently, the main conclusion of this thesis is that the GA is a very promising approach to the problem considered. This conclusion is further emphasized by the fact that this work shows that many ways only a single large effort has been needed in the use of, e.g., simulated annealing for these applications. Hence, significant future improvements are likely.

The performance of the developed GA is clearly superior to that of a previous GA for macro-cell placement and a previous GA for the Steiner path 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 improved performance. Here, for providing strong characteristics similar to those considered here, traditional binary encodings and the standard genetic operators should be studied to ensure that the efficient enforcement of constraint satisfaction.

A number of problems have been identified, among which the runtime requirements is the most serious. The current implementation of the developed algorithm is 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 data, 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 reinitialization of the population 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 only distributed among the processors. This has not yet been applied to parallel any of the classical GAs. The main reason is that although the current sequential implementation is not runtime efficient, the runtime requirements are not an obstacle for the practical experimentation either.

The typical hardware available at the design sites is a set of interconnected workstations. Consequently, from an application point of view parallel implementation 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 connection is very slow compared to that of other parallel architectures. This is demonstrated in [Min 96] which presents a parallel implementation of the GA for standard cell placement introduced in [Shindar 90]. A number of systems exist, which supports the development of parallel algorithms to be executed on a set of heterogeneous machines interconnected by a network. One such system is PM (Parallel Virtual Machine) [Sunblat 90].

Although 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.
or subpopulation to another. The question is which state selection should be distributed as described and investigated experimentally. An example of a parallel GA is depicted which is illustrated in Figure 61. The model includes as special cases each of the models showing diffusion and migration discussed in [Mital 91, Dool91].

Figure 61: 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.

The types of processes exist: processing elements, local controllers, and a single global controller. A subpopulation consists of one local controller and a number of processing elements. The local controller performs the selection within the subpopulation and then allocates tasks to the processing elements. Each task corresponds to one or more executions of a genetic operator on individuals of which copies are given. Occasionally, (copies of) identical 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 over a generation due to the selection.

This model has been implemented on a transputer network.
related accordingly

Introducing droppings of solutions when an upper bound on cost is exceeded could also potently improve runtimes. As a for example a steady-state GA in which a generated offspring always replaces the current best solution. If, during a drooping process initiated by the crossover operator, it can be determined that the cost of the newly generated offspring will exceed the cost of the current best solution, then the drooping of that offspring can be interrupted as it will not be inserted in the population anyway.

Although the improvements made above are expected to be very effective if implemented, it is still an open question if the present algorithm will be able to handle the larger benchmarks. AIS and AID in a reasonable amount of time. Some kind of hierarchical partitioning, as used by e.g. the B-algorithm [Gudra 91] and the BRR system [Ar 81] 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, large segments of the population can be passed by the genetic operators are independent, i.e., they can be processed in any order. The tests rely on local information only (the input individuals), and they require much shorter times because of the different types of tests and because of the 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. Note, e.g., the work on parallel GA for combinatorial optimization is due, e.g., Mathon et al. [Mathon 81].

The key issue when implementing a parallel GA is how to handle selection. The 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 restricting selection so that parents are always drawn from the same subpopulation. To allow information to spread through the entire population, it should also be possible for an individual to move from
6.3.1 Improvements of the Sequential Algorithm

The main reason for the excessive runtime requirements of the current implementation of the sequential algorithm and the global router is the repeated computation of channel densities, cf. Sections 5.22 and 5.41. A variable position for a block is saved by the sequential algorithm, channel densities involving the same set of instances are computed once 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 whenever a new position is tried out. Significant amounts of computation could 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 sets of terminals using a channel. Here, computation time could be reduced significantly by using data structures which allow the density of a channel to be dynamically updated as the sets of terminals using 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 needed.

Altering 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 once per generation, and the generated offspring is inserted immediately into the population for example, by replacement of the current worst solution(s). This contrasts the scheme used in the developed algorithm, in which all of the offspring is generated in each generation, assuming a population size of n. By eliminating the syndrome 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 crossovers. In other words, the algorithm presented here probably spends much time in each generation producing new 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
ay limitations, and all nets must be restricted to patterns of a certain type.

The BDD layout system allows floorplanning and global routing to be deeply interleaved. As described in Section 4.14 in [Bra 93], during the tree construction, the floorplan is then performed during a top-down traversal of the tree. Global routing can be incorporated into this traversal as described in [Fli 85]. A each level of the hierarchy, a global routing graph is extracted from the floorplan and then 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 internally of the new hierarchy. 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. At each level of the global nets, corresponding to the global net routes on a given level of the hierarchy is required. For this purpose, the data of floorplanning is used so that the extracted global routing graphs always have a certain freedom with a minimum tree can be created in linear time. Here, this scheme of simultaneous placement and global routing is not used in the earlier version of [Bra 93].

Another approach which only integrates placement and global routing is the SHP developed by Bpt and Gom [Bpt 93, Bpt 94]. The basic idea is recursive partitioning combined with a collection of pre-empted quads. Similar trees can be used at each level of the hierarchy. Global routing is refined as lower levels of the hierarchy is considered. However, the overall spirit, the basic ideas are quite similar to those of [Fli 85]. In [Bpt 92] the performance of SHP is compared to that of Tiram and found to be inferior with respect to solution quality but about the same faster.

6.3 Runtime Problems

It is expected that the runtime of the presented algorithm can be reduced significantly by improving the sequential algorithm as described in Section 6.3.1. Another and independent approach to speed up computation is to parallelize the algorithm as outlined in Section 6.3.2. The fact that the Gis parallel by nature and have very well suited for parallel
the Gmopter a key issue then become that of determining the relative
fitness of solutions while at the same time avoiding expressing solution
quality by a single figure of merit. In [Besca 9] it is described how this
can be done so that the Gmopter 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. With (samples of) this set known, the search can be
focused on certain interesting subsets by incorporating constraints for
some of the criteria. As pointed out in [Besca 9] the Gmopter is very well
suited for this kind of optimization/expansion because of its built-in
simultaneous investigation of many alternative solutions.

6.2 Simultaneous Placement and Global Routing

The inherent problem of the artificial separation of the initially depen-
dent placement and global routing tasks has been addressed through
Chapters 4 and 5. Form the synthesis process tangible, the tasks
were separated when this field emerged and the separation has been a
standard assumption ever since. During the years a better understand-
ing of the problem has developed and it seems natural to investigate
the possibilities of a reunification of the problem. Clearly,
at least in principle solution quality should improve when a problem is
solved as a whole rather than by solving subproblems of dependent sub-
problems. 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 ex-
tremely limited. For the simple design style of gate arrays, a scheme
for simultaneous placement and routing was proposed as early as 1982
in [Iseme 83]. A approach for building block layouts restricted to
sliding structures is presented in [Shea 83]. More, a number of
strict assumptions preclude the practical applicability of the algorithm.
For example, routing can be performed on 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.
nts are still trees, but the criterion minimizes the radius. My and Minz abandon the implicit assumption used throughout the literature for years, that a net is a tree \cite{My9}. Using the so-called three rule of delay, they show that network ratings my significantly improve the delay at a relatively small cost in terms of wavelength.

Matter which delay rule is used to adapt the SCL algorithm to delay minimization, it is necessary and sufficient to replace the decoder (and later the cost computation). Necessity follows from the fact that by creating an SCL circuit, the error decoder relies on knowledge of the criterion minimized. Sufficiency is a consequence of problem-specific knowledge being exploited by the decoder only. Here, a new decoder should implement a heuristic which given a set of selected vertices generates a feasible, spanning graph of reasonable lowest cost in terms of the delay rule used. A good performance valid be expected of the fifth conjecture of Section 5.5.

Turning to the question of the balancing of various possibly conflicting objectives, recall that in all presented algorithms, area is given higher priority than wavelength. Since both quantities are measured using estimates rather than exact values, the strict prioritization my in some cases deteriorate result quality as pointed out in Section 5.2. Here, from a practical point of view, even more important issue is the actual needs of the designer. When doing the layout synthesis phase, the designer needs 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.
Chapter 6

Future Work

Many aspects of the work presented in this thesis warrant 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 discussion from Sections 5.1.1 and 5.2.1 on the drive of optimization criteria and how they should be balanced. The consequences of the strong mutual dependence of the placement and global routing tasks have 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 problem of 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 balanced?

A point is in Section 5.1.1 the drive of minimizing 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 drive depends on the extent to which the algorithm exploits problem-specific knowledge. For the placement algorithm, the drive 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 S6 algorithm repeatedly used in the first place should minimize the delay of an edge rather than the net length. Various kinds of delay exist.

In [Fig 2] delay is defined from the radius of an edge. This is the minimum length from the source terminal to any sink terminal. Here,
A priori from this viewpoint it is in fact quite easy to find settings of the control parameters which yield 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 (unbinary) 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. Since experiments with previous algorithms of a similar nature, one has a (rather small) interval of feasible values for each parameter. Finding a solution which will can then be performed in a 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 for all whenever a new algorithm is developed.
- **Bitwise quantization** of area and interconnect length should be accurate, since as already mentioned, the estimates effectively amount to adjusting the cost functions. If the rise margin is not maintained, the work performed by the GA in the late phase, in which most of the time is spent and only relative small improvements obtained may be pointless. This situation suggests the use of estimates of which the accuracy is increased dramatically 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 in addition to other techniques. Section 3.4.1 and [9, 91, 92]. A rational in Section 5.1.3 all GA developed in this work apply inversion operators, and all experiments which addressed inversion turned out in favor of the operator. Here, the inversion operators are one of the contributors to the obtained performance.

- The SGA algorithm performs better than the other GA developed in the same sense that it is also runin sequence. It is conjectured that by generalizing the principles of the SGA algorithm, high-performing GA for a larger class of graph algorithms can be designed. A simple bitstring can specify the vertices and/or edges which should be (part of) a solution to a given problem. By using a fast, deterministic heuristic for the problem or some other repair algorithm the decoder can ensure that any bitstring is interpreted as a feasible (or possibly randomly good) solution. Standard genetic operators can be applied. Examples of problems which particularly are well suited for this approach are the minimum vertex cover and graph coloring.

---

10. In the case of graph coloring, 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 planar and global rating algorithms, this was at the cost of excessive computation time. In contrast, the SI 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 years. Significantly more time has probably been spent on a system such as ThrWMT.

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

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

- Due to the nature of the problem, constraints should be handled by eficacient, avoiding penalty terms in the cost function, cf. the discussion of Section 3.4.3. An additional reason is that the details of penalty terms have even more pronounced for the problem considered here because of the individual strings, which effectively amounts to aligning the cost function. Experimental indications support this claim. There is a significant difference between the GA for the SI presented in Appendix C and the GA in [Kapalis 92] is the constraint handling strategies, enforced versus penalty. The obtained results are very clearly in favor of the GA using enforcement. Similarly comparing the GA for planar to SI, one of the major differences is the constraint handling strategies. Again the algorithm using enforcement clearly performed 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 wire length obtained by the GA is exactly as the estimate used in Thrive which is clearly more accurate than that of Mary due to the different strategies for positioning terminal vertices.

Copying the routing area estimate of the GA to that of Mary, the GA performs the most accurate computation. As explained in Section 4.22, in Mary, 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, Mary is capable of adjusting the placement of certain blocks, while the GA relies on the computer 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 computed to their minimum width by the computer. If that is not the case, the GA will underestimate the total area.

Another problem of the area estimation of Mary and the GA is that they both rely on the routing graph topology to be preserved. As 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 require 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 Thrive, Section D.4.3, page 216, includes examples on what may happen when that is not the case. The issue of mirroring the gap between placement and global routing is brought up in Section 6.2.
Since there is no stochastic variation when using ThrWMC only one layout was generated using this router. The 5.9 column the total area the routing area (i.e., the total area minus the sum of the cell areas) and the total wire length of the completed layouts. Each entry is computed as $10(GA_{res}/TW_{res}) - 1$, where $GA_{res}$ is the result using the GA and $TW_{res}$ is the result using ThrWMC alone, a negative value is an improvement in percent of using the CA. The CA is clearly superior to ThrWMC with respect to layout quality for Xerox the area reduction is obtained by increasing the wire length while for the other examples, area as well as wire length is reduced. However, the quality improvement comes at a high price. While ThrWMC spends about 30 seconds rating each of Xerox and AiiB and about 5 minutes rating AiiD on average the GA requires about 22 minutes for Xerox, 12 minutes for AiiB and 30 minutes for AiiD. All values are elapsed time on a Sun Sparc 1 IX. ThrWMC spends most of its time computing dand densities, which are computed for all edges of the routing graph. Whereas an averaging solution is evaluated by keeping track of the actual need for computing dand densities it is expected that the runtime would be significantly improved.

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Solution</th>
<th>Total area</th>
<th>Rating area</th>
<th>Wire length</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xerox</td>
<td>best</td>
<td>-1.9</td>
<td>-4.7</td>
<td>-9.0</td>
</tr>
<tr>
<td></td>
<td>avg</td>
<td>-1.4</td>
<td>-3.5</td>
<td>-9.8</td>
</tr>
<tr>
<td>AiiB</td>
<td>best</td>
<td>-3.0</td>
<td>-4.7</td>
<td>-1.5</td>
</tr>
<tr>
<td></td>
<td>avg</td>
<td>-1.1</td>
<td>-1.7</td>
<td>-0.2</td>
</tr>
<tr>
<td>AiiD</td>
<td>best</td>
<td>-4.2</td>
<td>-7.3</td>
<td>-4.0</td>
</tr>
<tr>
<td></td>
<td>avg</td>
<td>-3.7</td>
<td>-6.3</td>
<td>-2.9</td>
</tr>
</tbody>
</table>

The 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.

---

8The circuits listed as Xerox, AiiB33 and AiiD49 corresponds to xerox-M, a33-2-M and a49-2-M respectively, in Table D2, 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 contained in Section 5.4.2. Though these sections, the term GA refers to the global router presented in Appendix D T mask WfM. M refers to the global router of the system [Ref 88, Ref 88] (Section 4.2.1), and Mercur is the approach of [Nakazaki 88] (Section 4.2.2).

5.4.1 Reference

Since the GAs are an extension of the interconnect length. Mercury minimizes the length of critical nets, and secondary area, and ThrWfM minimizes total interconnect length. The performance of the GAs is preferable to that of Mercury. Unfortunately, for purely technical reasons that was not possible.

In other respects the comparison of the two routers is prejudicial as far as this kind of comparison will ever be. All input placements are generated by Flex, a SCA-based tool interfaced to Ottools and both routers are also interfaced to Mercury. 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 time reasonably comparable.

Parents of the Microwind Xerox, Aix3, and Aix3 were used for the experiments. Due to a technical problem it was necessary to rename all input terminals (pins) from the examples. The completed layouts were generated using the GA for global routing.

---

6 Mercur is included in Ottools version 5.2 installed at Aarhus University, but our version of the programs is not functioning.

7 In Ottools 5.2, the placement of pads produced by the programPadplace cannot be handled by the channel definition programAtl as. This is the same problem that prevents the use of theGA-based router on placements generated by SAGA, as mentioned in Section 5.2.1.
The 5.8 compares the absolute computation times required by SHI, B1, B2 and the GA. For each algorithm and each class of graphs, the table gives the minimum computation time required to solve one problem. I.e., considering B1 and class C the easiest (for B1) of the 2 problems was solved in 10 seconds while the hardest (for B1) required 4.88 seconds. For the GA the values listed are the minimum of the average values obtained for each graph. We omit reporting the absolute values of Table 5.8 as should be seen in and the different tables used B1 are run on a VAX 800 and B2 is run on a Sun Sparc 2. For SHI and the GA a Sun Sparc 10 was used for graphs from classes B, C and D while a DEC 5002 was used for the graphs of class E.

<table>
<thead>
<tr>
<th>Cas</th>
<th>B1</th>
<th></th>
<th>B2</th>
<th></th>
<th>SHI</th>
<th></th>
<th>GA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>min</td>
<td>in</td>
<td>min</td>
<td>in</td>
<td>min</td>
<td>in</td>
<td>min</td>
</tr>
<tr>
<td>B</td>
<td>-</td>
<td>-</td>
<td>0</td>
<td>18</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>C</td>
<td>10</td>
<td>5</td>
<td>20</td>
<td>26</td>
<td>61</td>
<td>11,34</td>
<td>79</td>
</tr>
<tr>
<td>D</td>
<td>47</td>
<td>24</td>
<td>34</td>
<td>100</td>
<td>69,00</td>
<td>501</td>
<td>341</td>
</tr>
<tr>
<td>E</td>
<td>179</td>
<td>34</td>
<td>7,334</td>
<td>3×10^7</td>
<td>7,335</td>
<td>21,105</td>
<td></td>
</tr>
</tbody>
</table>

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 SHI for classes D and E are estimates.

Despite the use of different ratios it is clear that the GA ratios are always very close to any of the other algorithms. Both B1 and B2 are able to solve some problems extremely fast, although faster than the GA. B1 for other problems, the nature of the branch and bound algorithm explains these improvements over the ratio of running time solved. The minimums for class E are not available for these algorithms, since B1 failed to solve any of the problems within a 10 day limit, while for B2 [from Table 2] only lists minimums for 5 of the 20 graphs, presumably because of runtime problems. Considering the estimated time of SHI 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.
Chap. Text

- Abrams' approach by Chandra et al. [Chandra 9] detailed here by [12].
- Te btramadat approach by Irem and Bailey [Irem 9] detailed in Section 4.2.3 and detailed here by [12].

The benchmark data used are from the Ordinatry [J. E Bailey 9] and consists of 78 randomly generated instances, which are divided into four classes B, C, D, and E according to size. Class B has at most 100 vertices, while in classes C, D, and E each graph has 500, 1000, and 2000 vertices, respectively. The number of vertices to be spamed varies from to half of the vertices of the graph and average vertex degree varies from 2.5 to 50. The largest graphs have 62500 edges. Class B contains 18 graphs while each of classes C, D, and E contains 20 graphs. The solutions are known for all examples, which are initially found by a branch and bound algorithm and a Greedy sparer [J. E Bailey 9].

<table>
<thead>
<tr>
<th>Class</th>
<th>SHI &lt;1%</th>
<th>GA &lt;1%</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>94.4</td>
<td>100</td>
</tr>
<tr>
<td>C</td>
<td>600</td>
<td>78.0</td>
</tr>
<tr>
<td>D</td>
<td>20.0</td>
<td>55.0</td>
</tr>
<tr>
<td>E</td>
<td>81.0</td>
<td>81.0</td>
</tr>
</tbody>
</table>

Table 5.7: Comparison of the solution quality obtained by the SHL and the GA.

As is expected for Section C45, page 100, the genetic algorithms presented in [Koosis 9] is clearly inferior to all other algorithms considered here, and are not discussed further in this Section. While [12] and [12] find a global optimum solution for all graphs within the bounds of the table, this is of course not the case for SHL and the GA. Table 5.7 compares the solution quality obtained by the latter algorithms. For each class of graphs and each algorithm, the table gives the percentage of all performed executions which gave a solution within 0 % respectively 1 % from the global optimum. Since SHL 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, D, and one per graph in class E. So, as an example, 25 % of the 10 × 20 = 200 executions of the GA on the graphs of class D gave a solution within 1 % from the global optimum, while in 7.5 % of these runs a global optimum was found.
easily be constructed for which neither MP nor GSAA can ever find a globally optimum solution. Third location can be assigned to the parent of Fig. 51 (c) so that this parent corresponds to the global optimum for the complete layout.

![Diagram](image)

**Figure 51:** (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.

The potential problem of GSAA and MP 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 GSAA than for MP since for a given problem space of slicing structures is generally larger than the space of BL-placements. Furthermore, the relation to slicing structures has some advantages with respect to rating with the BL-placements do not, cf. Section 2.3

### 5.3 Evaluation of the Steiner Tree Algorithm

In this section the term refers to the algorithm for the SC presented in Appendix C. Its performance has been compared to that of five other approaches:

- Deterministic heuristic denoted here by SHI, described in Section C.2, page 182. According to a comparative study presented in [Wier 92], SHI is among the very best deterministic heuristics, and is superior to e.g. a parallel algorithm [Bandyopadhyay & Singh 88].

- A genetic algorithm by Kipnis et al. [Kipnis 92], with the knowledge is the only genetic algorithm for the SP referenced prior to this work.
than the other systems. However, the WM includes a third factor which depends on the location of the dam in the layout, so that dams close to the outer of the circuit will be wider, cf. Section 41.5.

There is no doubt that this is a good idea. Cogestion at the outer region of a circuit may significantly affect the size of dams everywhere, and this phenomena is neither captured by NP nor the GA.

Finally, some comments on the area estimation used in the GA concerning the relation between the two terms measuring local and global cogestion. It should be noted that independent of the setting of the parameters $\alpha$ and $\beta$, the global rating factor is not equivalent to simple block cogestion, since if the dam density is zero nothing can 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 local density estimation can be estimated for example by assuming that all terminals are uniformly distributed over all blocks. This gives bounds on the possible dam densities, i.e., the magnitude of the local cogestion term. The desired magnitude of the global rating term can then be determined considering block lengths and the total number of nets. Nevertheless, this issue requires further investigation aiming at diminishing the parameters.

Another concern is the imbalance of the two terms with respect to their accuracy. The local cogestion estimate is very accurate but also extremely time consuming and the main reason for the extensive computation time of the GA Gaining it with a less accurate term raises the question whether the timespendcomputing dam densities is fully justified although the accurate estimation presumes one of the reasons for the layout quality obtainable by the GA.

5.2.3 Search Space Relations

GA and WM differs significantly from the other approaches discussed by the way the search space is reduced. The restriction to B-placements followed from the viewpoint that placement problems a generalization of a binpacking problem. In contrast, NP uses the common reduction of the search space to sliding structures, while neither B|A|NP nor WM reduces the space explored. As illustrated in Fig. 5.1, placements exist which are contained in the search space explored by NP but not by G|A and vice versa. Furthermore, problems can
5.2.2 Estimation of Wiring Area and Interconnect Length

As stated previously, the issue of estimation 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 SA) are discussed and compared to those used in the other system presented in Chapter 4. However, since the nature of the estimation of the EEs is significantly different from those used in the other systems, comparison with EE is difficult and has been omitted.

Starting with wirelength, $NP$, $CM$, and $ThrWM$ all estimate total wirelength by summing the half-contacts of all nets, and implicitly use the half-contacts in the cost function estimating total area. In contrast, the Gestimate the length of each net by sum of the interconnect distance from each terminal to a constructed center-point of the net, as described in Section 5.1.2. The half-contact 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 more than two terminal nets, 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.

Trying 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 simple which is applied to $Bard$-$CM$ can be characterized as being inaccurate but very fast to compute. However, in $Bard$-area is also indirect affected by the estimated wirelength through the cost-function. The area estimates of $NP$, $ThrWM$, and the $CM$ are some common features. In all three systems, the estimated width of a demand includes two (artificially separated) contributions, one accounting for local congestion and one accounting for global routing, i.e., nets passing through the demand. In $NP$ and $ThrWM$, the local congestion is estimated by a function of the number of induced terminals, while in the $CM$, the exact demand density is counted. In all three systems, the global routing contribution is a function of the length of the demand. Hence, the accuracy of the global routing contribution is about the same for the three systems, while the Gestimate local congestion are accurately
<table>
<thead>
<tr>
<th>Benchmark</th>
<th>System</th>
<th>Area</th>
<th>Width</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>absolute</td>
<td>relative</td>
</tr>
<tr>
<td>Alpha</td>
<td>SKA</td>
<td>53.58</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>GA</td>
<td>53.90</td>
<td>1.10</td>
</tr>
<tr>
<td></td>
<td>BB</td>
<td>54.05</td>
<td>1.01</td>
</tr>
<tr>
<td></td>
<td>NP</td>
<td>54.77</td>
<td>1.02</td>
</tr>
<tr>
<td></td>
<td>GMP</td>
<td>61.86</td>
<td>1.15</td>
</tr>
<tr>
<td>Neox</td>
<td>NP</td>
<td>2.79</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>BB</td>
<td>3.17</td>
<td>1.01</td>
</tr>
<tr>
<td></td>
<td>GA</td>
<td>23.58</td>
<td>1.08</td>
</tr>
<tr>
<td></td>
<td>SKA</td>
<td>27.55</td>
<td>1.05</td>
</tr>
<tr>
<td></td>
<td>BEB</td>
<td>28.47</td>
<td>1.10</td>
</tr>
<tr>
<td></td>
<td>MACO</td>
<td>2.01</td>
<td>1.12</td>
</tr>
<tr>
<td></td>
<td>VIL</td>
<td>3.17</td>
<td>1.21</td>
</tr>
<tr>
<td></td>
<td>GMP</td>
<td>3.09</td>
<td>1.26</td>
</tr>
<tr>
<td>HPhi</td>
<td>SKA</td>
<td>11.81</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>NP</td>
<td>11.55</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>GA</td>
<td>11.55</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>BB</td>
<td>12.15</td>
<td>1.08</td>
</tr>
<tr>
<td></td>
<td>GMP</td>
<td>12.15</td>
<td>1.08</td>
</tr>
<tr>
<td>Athena</td>
<td>BB</td>
<td>22.4</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>NP</td>
<td>2.42</td>
<td>1.08</td>
</tr>
<tr>
<td></td>
<td>BEB</td>
<td>2.83</td>
<td>1.26</td>
</tr>
<tr>
<td></td>
<td>VIL</td>
<td>3.12</td>
<td>1.39</td>
</tr>
<tr>
<td></td>
<td>MACO</td>
<td>3.16</td>
<td>1.41</td>
</tr>
<tr>
<td></td>
<td>GMP</td>
<td>3.16</td>
<td>1.41</td>
</tr>
<tr>
<td>Athena0</td>
<td>NP</td>
<td>4.79</td>
<td>1.00</td>
</tr>
<tr>
<td></td>
<td>BB</td>
<td>5.49</td>
<td>1.06</td>
</tr>
<tr>
<td></td>
<td>GMP</td>
<td>5.49</td>
<td>1.06</td>
</tr>
</tbody>
</table>

The 5G 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 GMP for HPhi and Athena, 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 vield in obtains by the
GSGA to those of MP Wile on Nixx there are no significant
differences, for Ares and B1 the GSGAs are 31 - 61% higher
than those of MP Wile. Factors can contribute to these significant dif-
terences:

- To some extent there is a tradeoff between area and vield
  in using it is likely that a very small area is desired because of
  larger tolerances for some. Lower, this can imply a cost for
  much of the observable difference.

- The fitness function of the GSGA always gives higher pri-
  ority to area than vield. In solution A it is a much larger
  estimated total vield than B but just a slightly smaller esti-
  mated area, this Ans is considered better than B. Here, this
  priority relies heavily on a very accurate rating area estimation.
  This issue is discussed further in Section 61.

- The global rating of the GSGA plants could be inferior to that of MP's plants. As mentioned previously the
  GSGA plants are input to Meica which uses the global
  rating of Thir WMC described in Section 421. As will be
  described in Section 541, the GSHAD router presented in Ap-
  pendix D is clearly superior to Thir WMC both with respect
  to area and vield. Consequently it is very likely that by using
  the GSHAD global router rather than Thir WMC the areas
  and vields reported for the GSGA will improve further.

Regarding the reservations concerning the fairness of copying completed
layouts, it can be noted from the last point above that the GSGA
probably is disadualaged with respect to the influence of global rating

---

5 The GSHAD global router is interfaced to Ottools version 5.2, installed at Arhus University. In this Ottools version some bugs not present in version 5.1 prevent i/o terminals (pads) from being handled. Specifically, the channel definition program Atlas can not handle the placement of pads generated by Redface. The global router could then be interfaced to Ottools 54 installed at University of Michigan. But because of other differences between the two Ottools 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 rating and compilation etc. The time for Br is measured on a DEC 1100 while NBr uses an Apollo DN400. For both the times listed are dependent on the CPU and are measured on a VAX 880 with wall clock 105-125. Finally, GAMP is run on a Sun Sparc 1+ and the GA and SGA on a DEC MPs 500-20. For the latter two algorithms the listed runtimes are average values.

The remaining of this Section consists on the results listed in Table 5.6, keeping in mind the above reservations. Both SGA and GA perform very well with respect to area, the main layout quality criterion. For A4e and B3 the best published results are obtained by SGA, although differences between the best results are small. For A5e, the two algorithms are inferior to NBr and Br but are still doing better than e.g. B3. Thinking to runtime, we see that the GA and SGA require about the same amount of time and that they are significantly slower than all other algorithms. The runtime requirements of the algorithms present A4e and A5e, which being placed. Very sufficient results are available, GAMP are about 2-3 times slower than Br and GAMP the other GA and approach Gapped to NBr the algorithms are 10-20 times slower. The runtime used for the GSA is presumably at least as fast as the other runtimes, which further qualifies these differences. The other hard sequential programs can often be speeded up significantly by a mere programming effort. Only very limited time has been spent on this for the GSA and SGA.

It is possible that to some extent the small improvements obtained by using significantly more computation time may reflect the nature of the problem. 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 GSA and SGA are not satisfactory and at least the algorithms should be able to handle all benchmarks. In Section 6.3 and 6.4 possibilities of improving runtime are discussed and I believe that significant improvements can be obtained.
system [Gatto 80] and the VH [system [Ratnam 88] are referred
here as f.o.d [L 80, Tipton 90]. No results for Tipton W are
given since we were found for the MNC benchmark.

<table>
<thead>
<tr>
<th>Problem</th>
<th>Gls</th>
<th>Nts</th>
<th>Thirds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ace</td>
<td>9</td>
<td>9F</td>
<td>27</td>
</tr>
<tr>
<td>Nc</td>
<td>10</td>
<td>2B</td>
<td>68</td>
</tr>
<tr>
<td>H4</td>
<td>11</td>
<td>83</td>
<td>39</td>
</tr>
<tr>
<td>AitB</td>
<td>33</td>
<td>123</td>
<td>52</td>
</tr>
<tr>
<td>AitD</td>
<td>49</td>
<td>48</td>
<td>93</td>
</tr>
</tbody>
</table>

Table 5.5 Characteristics of the MNC benchmark circuits. The number of terminals includes the number of i/o-terminal 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 rating and compaction, etc. In other work, since it is not possible to capture planarly directly, the table copies the performance of complete layout systems including rates and compactions, etc. A least three other factors explicate the comparison

- The exact problem definition used varies slightly. For example, the B system defined 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 cases. However, the results for MP GNP and possibly other approaches are "best of a small number of runs". A caveat for this is that a slightly better result will be attainable using the stochastic algorithm a few times, which is useful if layout quality is the top priority. Of course the die area effect on capacitance may then be taken into account. To make the comparison to the other stochastic approaches 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 rating and compaction, etc. has been done using MetaMax of Section 5.1.

---

3 Personal communication with authors of [Chan 91, Tipton 90].
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.
A can be seen from the table the practial values used are quite similar. Variations in population size and step 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 example, if \( r \) and mutation rate for \( n = k \) is 0.005, i.e. similar to the mutation rate of GivG. The mutation rate for \( r \) of 0.001 is significantly higher, which coincides with the view of [Teo 92] who states that "...problems requiring more binary encodings may benefit from higher mutation rates than those generally used with binary encodings." Extensive experiments have shown that the algorithms are quite robust to changes of the parameters setting. 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 convey the placement algorithms with other approaches. Section 5.2.1 contains quality and computation time, Section 5.2.2 compares the area and wirelength estimation and Section 5.2.3 discusses the differences regarding search space reduction. Throughout these Sections, GA will refer to the algorithms presented in Appendix A. SAGA refers to inclusive variants of the algorithms presented in Appendix B. MP is the approach of [Yoon 90] (Section 4.1.1), B3 refers to Orad [Giso 91] (Section 4.1.2), G3P is the approach of [Chen 90] (Section 4.1.3), B3 refers to [Ed 81] (Section 4.1.4) and ThiW refers to the placement algorithm used in the ThiW system [Sedm 88] (Section 4.1.5).

5.2.1 Performance

The main characteristics of the MChenalis [Kimka 91] used for performance evaluation are listed in Table 5.5.

1. The default results are compared to other approaches in Table 5.6, page 76. All to the best of our knowledge include all the best published results. The M3ACO

---

1 As 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 Ortoools [Ortools 93]. We will use capital letters for the placement algorithms described in [Gasotto 86].
In QAe, not 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 GA, a constraint satisfaction is enforced by the decoder exclusively.

<table>
<thead>
<tr>
<th></th>
<th>QAe</th>
<th>GA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Representation</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Tolerate</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Genetic operators</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>

Table 5.3 Applied constraint enforcement methods.

Selection of Parameter Values

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

<table>
<thead>
<tr>
<th></th>
<th>QAe</th>
<th>GA</th>
<th>GAk</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population size</td>
<td>25</td>
<td>40</td>
<td>40</td>
</tr>
<tr>
<td>Stop criterion</td>
<td>20</td>
<td>50</td>
<td>10</td>
</tr>
<tr>
<td>Mutation rate</td>
<td>.005</td>
<td>.005</td>
<td>.002(r^k - 1)</td>
</tr>
<tr>
<td>Inversion rate</td>
<td>.05</td>
<td>.01</td>
<td>.01</td>
</tr>
</tbody>
</table>

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 improvement has been observed upon termination of the algorithm. Since all mutation operators perform “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 GAk, \( r_k \) is the number of alternative rates of the \( k \)th run. Here, the integer value identifying a specific rate for run \( k \) 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 3.42 are listed and the algorithms classified accordingly.

<table>
<thead>
<tr>
<th>Ways</th>
<th>GGA</th>
<th>GP</th>
<th>GA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idiomatic code</td>
<td>ID</td>
<td>ID</td>
<td>ID</td>
</tr>
<tr>
<td>Idiomatic quanta</td>
<td>ID</td>
<td>ID</td>
<td>ID</td>
</tr>
<tr>
<td>New quanta added</td>
<td>ID</td>
<td>ID</td>
<td>ID</td>
</tr>
<tr>
<td>Setting initial population</td>
<td>ID</td>
<td>ID</td>
<td>YES</td>
</tr>
</tbody>
</table>

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

GGA do not exploit any problem-specific knowledge. Although rather complex, the decoder as well as the genetic quanta merely ensure feasibility of the generated individuals and do not attempt to discard poor solutions. Section 4.4.1, page 128 describes experiments on exploiting problem-specific knowledge in the crossover quanta of GGA. These attempts did not result in any improvement of the performance of the algorithm.

A natural way of exploiting the CRO approach applied to all algorithms uses hillclimbs, i.e. they have added quanta. However, these are not problem-specific but simply execute a sequence of fitness-improving mutations. Simulating it is evident that the use of problem-specific knowledge in the algorithm is very limited and here leaves a large potential for improvement of the algorithm of Section 3.42 and [Milewicz 9].

Constraint Handling

In a sense any problem has some constraints which need to be satisfied. For example, a solution may need 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 (if used) non-trivial constraints. As stated in the GA context description all algorithms apply the strategy of enforcing constraint satisfaction at all times. Table 5.3 lists the three ways of enforcing constraints introduced in Section 3.43. The problem definition for GGA contains no constraints, but algorithms exclude...
to a great extent the performance of a GA is the result of the complex interaction of a number of design rules and. For example, a statement like "selection schemes A or crossover is better than selection scheme B rarely holds true in practice 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 parent algorithms described in Appendix A and B are identical as far as their GA parts are concerned, and here are treated here as an algorithm referred to as GpR ace. The GA for the SC (Appendix C) is referred to as GApg and GpR out e denotes the GA for global routing (Appendix D), i.e., the GpR out e in the second phase of the global router.

**Encoding and Search Space Reductions**

Table 5.1 lists the genetic encoding used and indicates whether the search space is reduced or not. The search space considered by GpLace is limited to Placements only. Section 5.1.2 while the search space reduction in GpR out e is from an upper bound on the number of Seiner vertices in a minimum Seiner tree, defined by [Dake 76]. Hence, the search space reduction has the form of an upper limit on the number of Seiner vertices selected as an 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 GpR out e.

<table>
<thead>
<tr>
<th>Encoding</th>
<th>GpLace</th>
<th>GpR out e</th>
<th>GpR ace</th>
</tr>
</thead>
<tbody>
<tr>
<td>Search space reduction</td>
<td>binary tree</td>
<td>yes</td>
<td>binary string</td>
</tr>
</tbody>
</table>

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

This is not the case for GpLace, since the global optimum may be a Placement, as will be illustrated in Section 5.2.2. Another major difference is the way the search space reduction is incorporated into the algorithm. In GpLace the representation and the decoder enforce the reduction, while in GpR out e it is limited exclusively by the genetic operators.
5.1.3 Crossover and Design Decisions

All GA developed in this work applies to the same plate by having
the following properties in common:

1. Let $n$ be the population size. In each generation, a pool of $n \cdot d$ off-
spring is genetically recombined of the crossover operator. From
the resulting total of $2n$ individuals, the $n$ best is then chosen de-
terministically for survival into the next generation.

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

3. A mutation 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. In each generation, the best individual with the lowest fitness
is archived. This scheme should not be confused with the comm
scheme assuming the best individual in a generation is new-
detected. It is included to control for premature stagnation in the
search for the best individual as a function of time is monotonically increasing
while the best does not.

6. After the last generation, the best individual ever seen (possibly
zero) are optimized further using simple hillclimbers.

7. All constraints are handled exclusively by each parent. One-
wide auxiliary fitness function for each constraint.

This template was initially written as the template for not 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 version
of all algorithms. When copying specific items of the template to
different possibilities from the literature, one should keep in mind tht


rank data including random graphs of up to 2,500 vertices and 62,500 edges. Furthermore, I can to knowf the highperformance SGap graphs described in Section 4.2.3. The algorithms improved mainly by adding reduction tests, and the implementation was improved reducing runtime as well as many requirements. The new version of the algorithms copied among others to the hand and a set of approaches of Section 4.2.3 and results were published in [Bensen94] which is the paper reproduced in Appendix C.

Global Rating Based On To GA

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 SGs. Terminal nets are handled by Later's algorithm [Later 78] while nets with three or more terminals are handled by the GAs described above. Trial vertices are added at exact locations as in Tiger-WM [Sahn 88].

In 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. Rating area is estimated using planar graphs as in Mary [Nelizki 87]. However, as opposed to Mary, the estimate is based on approximation of the exact channel density for each edge of the rating 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 be low and relatively small area. This seeding does not improve the final layout quality obtained but speeds up convergence. The global router has been published as [Bensen94] 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 TinhWM [Scha88] and May [Nelz88] are both two-phase routers. Since the optimization criterion involves total wirelength, this leads to consideration of the SG A discussed in Sections 4.21 and 4.22. TinhWM and May generate only a single route for each net, having one or 5 terminals respectively due to the runtime requirements of the indirect algorithm. Partially this limits the overall quality obtainable, as noted in [Scha88]. Since a GA provides a number of distinct solutions in each run it could potentially overcome this problem. This feature of the GAs 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" away.

The basic idea of the developed GA for the SGAs is to represent a solution as a list of lengths 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 SGAs decoder, any liststring 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. Rating 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 SG algorithm. The interface to Maico 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 [Elban94] 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 rating application would probably be limited. In the meantime, because of the Odd Library [J. E. Bailey 90], a database containing challenging benchmark...
search process. Like any other G, 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 converge much slower than a GA. Initially, but later on the SA may do better. Here, the idea was to combine the GA with SA mining at calibrating 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 Beaulieu and Ebbing [Beaulieu 9], 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 SAA. Both the GA and SA are special cases of SAA when appropriate settings of the control parameters. The interesting part is of course evolution in mixed GSA and SA. Like the algorithm starts out as a pure GA, the performance of the algorithm decreases, SAA gradually and adaptively switches toward SA. Motions 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 toward SA is taken by decreasing the population size and increasing the number of attempted mutations. Ultimately, the population size may become zero, in which case the process is pure SA. To provide representation, genetic operators, etc. are unlinked i.e., they are as described in Appendix A.

Since SAA 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, SAA 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 of Section 1.2, the latter property is the one emphasized when presenting the algorithm SAA was published as [Beaulieu 9]. An extended version of this paper is presented in Appendix B. For example, the extended paper illustrates in detail what happens during a mixed evolution.

AGA for the SAA

As discussed in Section 2.3, following placement/bounding/ global routing is the layout synthesis step which should be expected to influence overall layout quality the most. When placing the layouts generated using GA and SA, one might think while the obtained areas are
where \( \lambda \) is the spring of the rating grid \( l \) and \( s \) is the length of side \( s \), \( \alpha \) and \( \beta \) are used for practices and the rating function returns the nearest integer of its argument. \( d \) is the exact distance of the grid from each node. However, the rating factor is computed considering all appropriate termini within the grid on either sides of \( l \) and \( s \) as illustrated in Fig. A3 on page 121. The total rating is computed without considering global rating i.e., this term expresses local cognition only. The two other terms \( \alpha = \sqrt{l_s^3 / \lambda} \) and \( \beta \), are used to account for global rating. The first one term grows with the length of the rating region similar to what is due in \([\text{John 98}]\), \( d \).

Section 41.1. Note that since \( d \) = 0 implies \( D \) = 0, the grid can be denoted. The wavelength is estimated as in \([\text{John 98}]\). Let \( M \) denote the number of nets, \( m \) the number of terminals of net \( k \) and \( t \) the coordinates of the \( i \)-th terminal of net \( k \). The center of gravity \( T \) of net \( k \) is then defined by

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

and the total wavelength estimated by

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

where \( || \) is the usual Euclidean vector norm. For the estimated total area and the estimated wavelength fitness is computed in such a way that smaller area always means higher fitness.

Initially this algorithm was interfaced with the layout system Mic [Scott 86] and published as [Isenberg 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 MicroSoft tools, which offers more and better tools than Mic. Appendix A describes the resulting version of the algorithm which performs significantly better than the original version described in [Isenberg 92].

A Unification of the Grad SA Applied to Microcell Placement

While the layout quality attained by the algorithm described above was very pleasing, the computation time required was still very large. Consequently, it was natural to look for some improvement of the
relevant, despite the fact that it is a frequently used criterion in the literature. Although short total wire length will often result in small layout area and short delay (the length of the longest path through the circuit), this will not always be the case. For high-performance circuits, explicit minimization of delay will be more adequate, but will require changes of the algorithm as will be discussed in Section 6.1. The first assumption prevents routing of all blocks. This will rarely be realistic unless the design is small and area is not constrained. Therefore, the first assumption is the one that encompasses the practical applicability of the algorithm. The net. However, incorporating over-the-cell routing in the algorithm will require significant alteration.

5.1.2 Wt and W− the Development History

AGAfor Mrose Placem

The first algorithm developed was AGAfor mrose placem. It is inspired by a GA for the two-dimensional bin-packing problem [Kag 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 mrose placem problem can be seen as the bin-packing problem generalized in two ways. Firstly, the placement of each rectangle (mrose) is directly a function of the minimum space 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 Te (min 

9) 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 captured 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 previous placed blocks is estimated as

$$
D_s = \begin{cases} 
\lambda \left[ d_s + \text{ran}(\alpha \sqrt{d_s + \beta}) \right] & \text{if } d_s > 0 \\
0 & \text{if } d_s = 0 
\end{cases}
$$
2. Two types 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 control 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 execution of the algorithm. For the reason discussed in Section 1.2, the aim is to evaluate performance of the algorithm 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 MC2Gater for Microarchitects, Ninth Generation [Koisti 91]. To be meaningful, comparisons have to be done in terms of complete layouts, cf. Section 2.3 and therefore the developed tools have been integrated with a complete microcell layout system called Maelo which is part of the G2Tools CADnak [G2Tools 9]. Since the above assumptions are compatible with the MC2Gater benchmarks as well as the assumptions of the Maelo toolset (routers, capacitor, etc.) they provide a feasible basis for the kind of comparisons desired.

The practical relevance of the assumptions is another issue. Poor nets have to be wider than signal nets, and hence are often routed by dedicated algorithms which also take care of sizing of the wires. Similarly, clock nets often require special treatment to avoid various timing problems. Here, 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 provide 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 become adequate. In any case, since the second and third assumptions are used only in the estimates of routing area and wirelength, and therefore concern isolated parts of the algorithm only, it would not be too difficult to adapt the algorithm to other versions of these assumptions.

While minimization of total layout area is highly relevant, cf. the forth assumption, the minimization of total wirelength is only indirectly
Chapter 5

Summary and Evaluation of Developed Algorithms

The appendix presents two approaches for macrocell placement (Appendix A and B), an algorithm for the Series Parallel a Graph (Appendix C) and a global router for macrocell layouts (Appendix D). This work is summarized in Section 5.1 and evaluated in Section 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 rely on the same set of basic assumptions about the problem 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 at major similarities and differences. Further considerations that lead from one piece of work to the next are included. Section 5.1.3 summarizes key design decisions taken for the OA and relates these to the discussion of practical issues of Section 3.4.

5.1.1 Basic Assumptions

The algorithms presented in this thesis all conform to 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 netlength 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.
4.3 GAs for Related Problems

References to other GA-based approaches for more advanced place-and-route using artificial neural networks are given below. The listing below is not meant to be exhaustive, but is intended to highlight significant GA-based approaches known to the author.

Ghon et al. present a distributed GA for floorplanning of building block layouts, which minimizes area and total wirelength [Ghon 94].

Glykis and St. John present an algorithm for automatic layout and routing using the genetic algorithm [Glykis 89].

Glykis et al. present a GA for VLSI layout synthesis, as Genie, developed by Glykis at University of Virginia [Glykis 89]. Later, Sivakumar et al. presented an algorithm for floorplanning called GASP [Sivakumar 90, Sivakumar 91], which at the time of publication outperformed IBM's GA-based approach called Ti mber Wolf SC. In [Melin 91] a parallel implementation of GAs presented.

A GA-based partitioning algorithm is presented in [Sivakumar 91, Sivakumar 92]. It can do bi-partitioning, i.e., recursively partition the layout in two parts, as well as multiway partitioning. Ahead of time 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 Fujita and Miyasaka [Fujita 88].

Finally a GA for detail routing by Leaig et al. should be mentioned [Leaig 93, Leaig 94]. The algorithm minimizes the number...
the fact that the original solution was already min. Consequently, given an \( \tilde{G} \) for \( G \) satisfying Equation 41, the \( \tilde{G} \) for the \( G \) is simply \( \tilde{G} \), and the cost of the \( \tilde{G} \) is the cost of the MT. Let \( P_i \subseteq \tilde{E} \) be the set of edges which connect to vertex \( i \). The problem of finding an MT for \( G \) is then formulated in [J. E. Baily 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

\[
\sum_{(\mu, \eta) \in \tilde{E}} x_{\mu \eta} = |V| - 1 \tag{42}
\]

\[
\forall T \subseteq \tilde{V} : \sum_{\mu \eta : (\mu, \eta) \in \tilde{E}} x_{\mu \eta} \leq |T| - 1 \tag{43}
\]

\[
\forall i \in V - W, \forall (p, q) \in \tilde{E}, x_{p q} \leq 1 \tag{44}
\]

\[
\forall (i, j) \in \tilde{E} : x_{ij} \in \{0, 1\} \tag{45}
\]

Equations 42 and 43 assure that the solution is a spanning tree, and Equation 44 is equivalent to Equation 41. The constraints are added in [J. E. Baily 89].

Before starting the branch-and-cut algorithm an attempt to reduce the size of the problem is performed by applying various \textit{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 MT. If the MT is a tree, all the tests used in [J. E. Baily 89] are also used by the algorithm presented in Appendix C. The theory will be discussed in Section C3.2, page 172.

The search is a depth-first traversal of a binary tree. An initial guess is obtained by the heuristic presented in [Fawad-Sith 88]. A exhaustive search is carried out so that only very few nodes left to be visited. Allow bounds are cut off by solving a linear program (LP) obtained by replacing Equation 45 and initially ignoring most of the constraints. The bound is then iteratively strengthened by
at Imperial College, London, UK, J. E. Bailey [8] and later imposed
by Hearn and Bailey [9, 2].

The basic idea of this backtracking approach is to transform the
SC into an equivalent Minimum Tree (MPT) problem.

Jet to an additional constraint. Assume \( V = \{ 1, 2, \ldots, n \} \), \( \forall (i, j) \in E : i < j \) and without loss of generality assume \( 1 \in W \). Furthermore, let

\[ c_{ij} \text{ denote the cost of edge } (i, j). \]

Then an extended graph \( \tilde{G} = (V, \tilde{E}) \) is constructed as illustrated in Fig. 42 (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 MPT for \( \tilde{G} \) which satisfies the additional constraint

\[ \forall i \in V - W : (0, i) \tilde{E} \Rightarrow \deg(i) = 1 \] (41)

will have the form illustrated in Fig. 43.

![Diagram](image_url)

**Figure 43** The structure of any MPT for \( \tilde{G} \).

If the edge \((0, 1)\) of the MPT we 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 vertex of \( \tilde{G}_1 \) includes \( W \), and in fact \( \tilde{G}_1 \) is a MPT for the original

SC. As in this is not the case. Then \( \tilde{G}_1 \) could be replaced by the true

MPT for the original SC at any left cut vertices could be added to

that is, they could be connected to vertex 0 using an edge of cost 0. The

resulting tree would be a feasible MPT for \( \tilde{G} \) satisfying Equation 41.

It would also be of lower cost than the original solution, contradicting
42.3 The Steiner Pointer a Graph: An Algorithm

The Steiner Problem in a Graph (SPG) is the following: Given a connected, undirected graph \( G = (V, E) \), an edge cost function \( c : E \to \mathbb{R} \), \( c \geq 0 \) 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) for \( W \) in \( G \). For \( 2 < |W| < |V| \) the SG is complete [Karp 72]. An example problem instance is shown in Fig. 42 (a).

![Figure 42 (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.](image)

The SG is surveyed in [Winter 87], which includes presentations of several deterministic heuristics for the problem. One of the most popular is an algorithmly by Reed-Sith and Cesc [Reed-Sith 88].

Many recent other deterministic heuristics of superior performance have been reported [Winter 89]. One of these is the Iterated Shortest Path Heuristic presented in Section C42, page 122. The best of knowledge, the only GA for the SPG published prior to this thesis is a recent algorithmly by Ipsi [Ipsi 99], which is discussed in Section C45, page 109.

In [Ipsi 92, J. E. Bailey 93, Imai 93] state-of-the-art approaches to the SPG based on branch and cut are presented. These algorithms solve problems up to \( |V| = 250 \) vertices to optimality. Terminating of this section summarises the algorithm initially developed by Bailey...
capacity violation is decreased or the amount of violation is increased but the total wirelength is decreased.

### 4.2.2 An Integral Polygonizing Approach

Mary is a dual router developed by Niziali, Iggo and Sajoami-Viratelli at University of California Berkeley [Niziali 8]. Mary 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, larger routes may be selected in phase two in order to minimize layout area. As opposed to the unrestricted positioning of terminal vertices used by ThrWM, in Mary 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 need to be considered only once. This makes the algorithm a less accurate net length estimate.

The phase one algorithm is the same as that of ThrWM, although the search performed by Schur's algorithm for multi-terminal nets has been removed, thereby improving the time complexity of the algorithm to $O(M^{k/(k+1)} n^3)$. By default, $M = 5$ is used when $k \leq 5$, and $M = 1$ otherwise.

In phase two, the algorithm minimizes layout area, which is estimated using polygonal graphs as described in Section D31, page 210. The factors contribute 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 channel is considered for such nets, their contribution to the width of each channel can be computed. 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 each 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 adjacency of the placement blocks. The adjacency constraint 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 D31. 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. These uprisins, the twostate-of-the-art global roters
selected for presentation in Sections 421 and 422, are also both graph
based phases, and with similar criteria.

To remedy repeatedly solved in the first phase, a two-phase rater
van rating an individual net is known as the Steiner Problem in a
Graph (SPG), assuming that the criteria minimize the net length. This
thesis also presents a new algorithm for the SPG and to facilitate com-
parison, Section 423 summarizes a state-of-the-art approach to this
algorithm.

42.1 The WM

The TMWM got systems introduced in Section 41.5. Chap-
ter 8 of [Schen 88] and [Schen 88] describes the global rater of Tim
WMW. It minimizes the total wirelength subject to the capacity
constraints of the rating regions. Electorically equivalent pins can be
rounded that is, if two or more terminals of a block are connected
the block, only one of the terminals will be externally connected by the
rater. These alternative rates for a specific net are placed in the rating
terms and are added to the rating graph at positions accurately reflecting
the position of the terminals. All terminal vertices are removed again
before the net is considered.

In phase one, the M shortest rates for each terminal net are com-
pited by an exact algorithm to later [Laker 71]. This algorithm
requires time \( O(Mn^3) \), where \( n \) is the number of vertices of the rating
graph. For nets with more than two terminals, Schen has developed a
heuristic generalization of Laker’s algorithm that attempts to find the
M shortest rates. Since this algorithm requires time \( O(M^{k+2}n^3) \) for a
net with \( k \) terminals,

\[ M \text{ has to be reduced for nets with many terminals. For two-terminal nets } M=2 \text{ is typically used while } M=1 \text{ for nets with 2 or more terminals.} \]

In phase two, a randomized algorithm is used. The initial
state consists of the shortest rate for each net. If no constraints are
violated the rater terminates. Otherwise, a new state is generated
by randomly selecting a net passing through a drum, the capacity of
which is not exceeded. An rater for the net is chosen at random
among the alternatives such that it minimizes the total amount of capacity
violations. The new rater is accepted if and only if the total amount of

\[ ^{1} \text{More precisely, } k \text{ is the number of terminals of the net, which are not electrically equivalent.} \]
4.2 Global Routing

Mt global routers for mac ell layouts perform routing intear of a retilinser global routing graph or channel graph, which is obtained from the given placement. The edges of the graph correspond to the routing regions while the vertices correspond to the intersections of routing regions. The routing graph of a single placement is shown in Fig. D1, page 37.

To compute a global route for a specific net, vertices representing the terminals are added at appropriate locations, as illustrated in Fig. D2, page 38. Finding a global route resembles finding a slice in the routing graph which spans the terminal vertices. Each 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 dmem 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. A each net is routed, current dmem congestion, 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 adopted by the two-phase routers, which generates a routing solution into distinct phases. In the first phase, several alternative routes are computed for each net. The nets are treated independently 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 dmem capacity constraints or any other constraints.

A main drawback of sequential routers is that the result quality is highly dependent on the order in which nets are routed, and in general it is difficult to devise a good ordering policy. This problem is solved by the two-phase router, within this respect treats all nets equally. Other hard 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 net graph-based. Some of global routing can be found in [Suri, 93, Vlcek 89].

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 traffic is meant to account for global routing passing through the channel and is proportional to the TIL divided by the estimated channel length. Since net nets will be implemented at very short rates, the closer a channel is to the outer of the layout, the more congested it is likely to be. The second factor accounts for this phenomenon by assuming that slices placed at the outer of the layout are allocated more surrounding space for routing than slices 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 congestion is calculated (Chapter 6 of [Sidhu 88]) is added to a description of the routing area estimate.

The placement algorithm is based on simulated annealing and the cost function minimizes consists of three terms. The first term \( C_{teic} \) is the TIL computed as a weighted sum of the half-prints 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 \( w C_{overlap} = \xi C_{teic} \) holds at the initial temperature \( T_0 \). A value of 0.5 were successfully used for \( \xi \). The first term of the cost function has to do with the positioning of pins of flexible blocks.

Two types of nets exist. All block can be moved to another position or a pair of blocks can exchange positions. Block orientation 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. Range limiter function decreases with temperature, so that initially a block can be moved any distance while at low temperatures, only short distances are generated. An heuristic similarly assures that short distance moves are not generated at high temperatures, since they are likely to be insignificant at the 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 \( 0 \leq \alpha(T_k) \leq 0.2 \). Temperature decreases the fastest towards the end of the process. From block problem Oh move are attempted at each temperature.
tivity between clusters $i$ and $j$. The need space $s_{kl}$ between adjacent nodes $k$ and $l$ is then estimated by

$$
s_{kl} = \lambda t_k \sum_{i,j} p_{ij} c_{ij}
$$

where $\lambda$ is the rating grid spring, $t_k$ is a parameter accounting for distinct nets sharing the same track in a dual, and $p_{ij}$ is the probability that a connection between nodes $i$ and $j$ passes through the region between nodes $k$ and $l$. The probability $p_{ij}$ is obtained by considering the shortest paths from block $i$ to block $j$, relying on the two-terminal representation of the nets.

### 4.1.5 ThrWM

ThrWM is an integrated system for planning/placement and global routing of multilevel block layouts developed by Schm at Yale University [Schm 88, Schm 88]. The system has been continually improved and refined for many years and now offers many useful facilities. One characteristic feature of ThrWM is that all min algorithms are based on simulated annealing.

The problem definition used is very general. Fixed as well as flexible blocks are allowed, allowing any retiming steps, and the search space is not restricted to e.g., sliding 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 TEIC, the Total Estimated Interconnect Length.

ThrWM consists of two main phases, initial placement and placement refinement. The latter consists of repeated execution of three steps: dual definition, global routing, and adjustment of the placement. Like the placement, the first two steps are repeated several times, and only three iterations of the second phase are 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 refinement 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 blockly an amount which depends on the position of the block. I.e., as opposed to the static block expansion strategy applied in [Chu 91] and [Ghara 93] 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 plant, i.e., the relative positions of all blocks, as well as the orientation of each block, is determined. 2) Global rating is performed. 3) The plant at the global rating is adapted to each other through a sequence of internal alterations of the plant at as well as the global rating. 4) Rating regions are defined and ordered. 5) Local rating is performed. At each region is rated, the plant is locally adjusted according to the final region width. At the third and the fifth step, which integrates the optimization of the floorplanning plant with the global and detailed routing respectively, fitting the third step during updates of the plant as well as the global rating are made possible by the use of sophisticated data structures described in [11, 87]. An attempt is made to preserve the topology of the plant at the block orientation are not altered.

The remaining of this Section focuses 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. To partitioning heuristic considers block sizes and connectivity. A plant is generated by a top-down traversal of the resulting cluster tree. The search space is not restricted in any way. A each level of the hierarchy, a cost function is minimized by an exhaustive search of all possible topological fragments of the individual blocks/clusters. Furthermore, block orientation are determined at the leaf level. The cost function is a weighted sum of geometry cost and connection cost. The latter temporalizes connections between adjacent clusters, while the first term minimizes total area and measures the relationship between actual shape and target shape of the plant. When using a specific topology, target shapes for each of the individual blocks/clusters are passed on level above the hierarchy. As the root of the tree, which corresponds to the complete plant, 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 which the cost significantly exceed the minimum obtained at the current level, are unlikely to lead to better plant. Consequently, such topologies are pruned from the search tree.

When evaluating a specific topology, the rating area is estimated as follows. An n-terminal net is represented as \( m(m - 1)/2 \) two-terminal connections between all pairs of terminals. A n-levels of the hierarchy, all connections are measured from vertex to center of the individual clusters. A connectivity matrix is created in which \( c_{ij} \) is the number...
3. $A$ and $B$ are divided into submatrices each according to the cross-point:

$$
A_{mn} = \begin{pmatrix}
A_{p1}^{11} & A_{p1}^{12} \\
A_{p2}^{21} & A_{p2}^{22}
\end{pmatrix}
$$

$$
B_{mn} = \begin{pmatrix}
B_{p1}^{11} & B_{p1}^{12} \\
B_{p2}^{21} & B_{p2}^{22}
\end{pmatrix}
$$

4. The splitting are defined as

$$
C_{mn} = \begin{pmatrix}
A^{11} & B^{12} \\
B^{21} & A^{22}
\end{pmatrix}
$$

$$
D_{mn} = \begin{pmatrix}
B^{11} & A^{21} \\
A^{12} & B^{22}
\end{pmatrix}
$$

Notice that the first step is needed to assure that each feature is copied into each offspring exactly once. Any step of this type is moderately expensive by applying a random generator. In this specific case, since inversion prevents any row and column of the bit map from as opposed to individual entries, the sorting required in step one is done in time $O((n + m) \log(n + m))$ instead of $O(n m \log(n m))$.

Given the offspring is immediately initialized in the new population as in the simple GA, Section 3.2. Instead of all offspring is generated and the new population is determined stochastically defines the best individuals of the offspring plus 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 HDR System

A diversity of California, the Kd-tree, is an integrated system for floorplanning, placement, and routing of high-speed block layouts called HDR (High-speed Block Routing). This system has been developed by L. Edman, M. I. and Redman [L.E. 89, Edman 88]. Given a set of rectangular boxes, which may be flexible at a fixed number of layouts is minimized area and total wire length while considering given target values for the height, width, or aspect ratio of the layout. The characteristic feature of HDR 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 the steps.
4.1.3 ACG Approach

To our knowledge only two GA for nano-cell placement have been published prior to the algorithm presented in this thesis. GNP developed by Chin, Schrader and Mindor at University of Michigan [Chin 94, Schrader 94], is the subject of this Section, while the other approach is discussed in Section 4.3. GNP initiates nano-cells of any rectangular shape and the search space is not restricted in any way. Two criteria are minimized: area total width and violation of given bonds on each dimension of the first layout. Rating area is estimated by initial opinion of all blocks by a certain amount similar to what is done in [Gebra 91]. Total width is estimated by the sum of the half-printers of all nets.

Given \( n \) blocks, the genotype of a solution is an \( n \times m \) boolean matrix referred to as a bitmap chromosome. Each row represents the placement of a specific block as a row of 1's and 0's. Each item of its \( x \) and \( y \) coordinates and three additional bits select one of the eight possible rotations of the block. Infeasible solutions are allowed and penalized by the cost measure, which is a weighted sum of terms. The total area, i.e., the smallest rectangle enclosing all blocks, the estimated total width, 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.

Grosser, mutation and inversion are the three genetic operators used. Baldwinian, all entries of the bitmap chromosome are tagged with identifiers of the features they encode. Section 4.4.1. 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 follow:

1. A copy of \( B \), denoted \( \overline{B} \), is made homologous to \( A \), that is, it is reached by permuting columns and rows so that each entry of \( \overline{B} \) 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 a partial placement, using the smallest side lengths of any holes
which are not yet oriented. The decision tree is traversed depth-first to
reduce storage requirements.

For bounding questions are used

1. As an upper bound can be computed just described. Initially,
a good solution is obtained by solving an approximation of the cost
function subject to the constraints using linear programming.

2. An acute 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 infeasibility 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 bound-
ing, while lower bounds are of no use.

To use constraints the user specifies, the more effective the search
becomes. When additional constraints are specified, equations like
show that at most 6 blocks can be placed by the algorithm in a
reasonable amount of 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 consists of more than 36 blocks, the partitioning is li-
eral. The batch and bank algorithm is then repeatedly applied
on each cluster of the hierarchy in a bottom-up order. The partitioning
algorithm needs to be relatively simple and considers connectivity only.

A first thought one might think that since this approach is based on
batch and bank, it should always produce an optimal placement. Now
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
down 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 algorithms need
for layouts of more than 6 blocks inherently lead to suboptimal results.
41.2 Abandoned Grid Approach

Abandoned grid algorithms have been developed by Odra at University of Gifunma, Hiddle and Tugden at Kyoto University, Japan [Odra 9]. It places rectangular cells with minimum layout area. The search space considered is unrestricted i.e., not limited to e.g., sliding structures. The cost function minimized

\[(W_x + \frac{\lambda_x}{W_y} \sum_i L_{i,y}) \times (W_y + \frac{\lambda_y}{W_x} \sum_i L_{i,x})\]

where \(W_x \) (\(W_y\)) is the width (height) of the smallest rectangle containing all blocks, \(L_{i,x} \) (\(L_{i,y}\)) is the width (height) of the smallest rectangle containing all pins of net \(i\), and \(\lambda_x \) (\(\lambda_y\)) is the routing grid spacing in the horizontal (vertical) direction. Here, the function optimized is the area of the smallest rectangle containing all blocks, with \(\lambda_y\) equal to \(1\) for routing. The expansion in the y dimension is \(\lambda_y \) times \(\frac{1}{W_x} \sum_i L_{i,x}\), i.e.,

The cost function is minimized subject to specified constraints on the size of the layout and on critical nets. The shape constraints can be a target aspect ratio, bounds on aspect ratio, or upper bounds on area or both dimensions of the layout. Constraints on critical nets can be upper and/or lower bounds on the length of these nets.

A layout is described by specifying an orientation of each block and the topological relationship between each 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 an 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 bundling schedule, larger blocks and/or blocks related to critical nets are considered before smaller blocks. Furthermore, the topological relationship of the blocks constituting a partial placement is always determined before the orientation of the blocks. Allow bundling on cost can then be obtained by solving the cost function described
NP applies a classical estimate of the total netlength 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. The netlength of the layout is estimated as the sum of all nets of the half-perimeters.

![Diagram](image.png)

**Figure 41.** 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 NP is that the two cost factors, area cost and netlength cost, are not combined into a single cost measure using a weighted sum as is done in many algorithms. As pointed out in [Iyengar 92, Iyengar 93], a weighted sum often introduces blurring problems caused by the different nature of the individual factors. These problems are of the very same nature as the problems of purity tests discussed in Section 3.4.3. Instead, in NP the two criteria are evaluated independently. Two heuristics are maintained one for each criteria, using the same coding 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 trees of the slicing tree can be extended, as a subtree can be moved and the orientation of a subtree can be altered. An subtree can consist of any number of blocks. The fourth move type is alteration of the aspect ratio of a bilevel block.
4.1.1 SImulated Annealing Approach

The Micro-Bock Planner program (MB), developed by Toshi, Sint and Sjye Sint, at the Singapore University of Technology, is described in [Toshi 98, Sint 99]. MB takes as input a layout, which is a mixture of standard cells and macro cells. This capability is highly relevant for real-world design, as Section 22.1.2. A planner is generated in three main steps:

1. The standard cells of the layout are partitioned into a number of feasible blocks. The partitions are the sizes of 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 floorplanning. An aspect ratio is to be determined for all feasible blocks just created, and all blocks are to be placed and oriented. The area, total wire length, and the deviation from the target aspect ratio is minimized using simulated annealing.

3. With all aspect ratios determined and all blocks 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 MB. The search is restricted to sliding structures, and the algorithm operates in terms of the slicing tree, a binary tree representing the slicing structure as illustrated in Fig. 41.

Three factors contribute to the estimated area cost: 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. A cut intermediate node the width of the channel separating the two subtrees is estimated. The width of the channel between blocks A and B is estimated as $\lambda = \sqrt{t_A + t_B} + \sqrt{nW(l_A, b)}$, where $\lambda$ is the spacing of the routing grid, $t$ is the number of terminals along the relevant side of block $X$, and $l$ is the length of that side. The first term estimates congestion in the channel, 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
knowledge constitute the current state-of-the-art in microcell placement
and global routing. To facilitate meaningful performance compar-
isons, a further selection criterion has been that the used problem defi-
tions 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 methods have been an issue,
although for the problems a previous G-based approach exist, it has been decided to facilitate detailed comparisons to the work presented
in this thesis.

Section 4.1 presents approaches for microcell 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 prob-
lems such as partitioning and chip routing are provided in Section 4.3.
The performance of the approaches will be compared in Chapter 5. Fa-
nility with the optimization techniques of simulated annealing and
branch-and-bound is assumed throughout the chapter.

4.1 Micro-Cell Placement

Five approaches to microcell placement are presented including algo-
rithms based on simulated annealing, branch-and-bound, and the GA.
For a survey on (microcell) placement techniques in general, the reader
is referred to [Sundar 91, Sarwai 92].
be caused partly by antler fracture, which profound as "god's inpt individuals. Therefore, a scheme needs in which it is pragmatically "backward" through a sequence of operators, which produce (the ancestors of) a specific individual.
of times for each problem using the parameter values repeatedly by the individual of the nataGA. The search space explored by the nataGA is very small compared to used GA applications, e.g., in [Gefenstette 86] the space merely consists of 22144 points. This is one reason why it is reasonable to assume that the parameters used for the nataGA are not too critical. Otherwise, of course nothing valid has been gained. Gefenstette used the parameter values recommended [De Jong 75] for the nataGA. Other consequences of the extremely small search space of the nataGA is that a very small population is sufficient and that a good result can probably be obtained in very few generations. Still, the explicit interaction between the various parameters of the GA to be tuned is captured much better by the nataGA than by simple strategies applied when searching for parameter settings manually. The obvious drawback of the nataGA approach is the runtime requirements. In [Gefenstette 86] it is said that "...the trained experts represent a sizable number of GAs", 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, this devises a scheme for dynamically updating some parameters during the execution of the GA [L Dais 89]. The parameters in question are the probabilities which define 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 most. Initial values for all parameters are still needed but are less critical. Apart from being able to find 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 algorithm itself introduces some new parameters, e.g., how often to perform minor updates. In [L Dais 89] the 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 my
However, the main limitation of both [D Jorg 73] and [Gemanette 88] is that the proposed values only apply to GA based on fixed length binary representation.

In [Gemanette 88] Geman presents a theoretical approach to the determination of an "optimal" population size, where "optimal" is defined in terms of smallest processing per individual. This approach is also limited to fixed length binary representations. The results presented suggest a population size which grows exponentially with the length of the encoding. As pointed out in [Rees 95], such population sizes will make the GA inferior to other optimization methods as real-world problems. Instead, Rees determines a lower bound on the usable population size, the objective being that of using the smallest possible population [Rees 95]. 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. Assuming that the initial population is generated uniformly at random as lower bound on population size can then be determined. Rees show that the minimum population size computed this way grows exponentially with the cardinality of the alphabet used for the encoding and presents this as an argument in favor of binary representation [Rees 95]. However, based on similar considerations, a different conclusion can also be reached. In [The 95] 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 constraining convergence, the runtime penalty of a larger population is avoided.

The third approach listed is to consider the search for good parameter values as a metaoptimization problem which is solved by using e.g., a GA referred to here as the metaGA. This approach was introduced by Gemanette, who as mention earlier mentioned a new set of generally accepted default values this way [Gemanette 88]. In [Shanker 92] a metaGA is used to find suitable values for a GA for standard benchmark. To individ metaGAs are 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. Using a metaGA fitness computation consists of executing the GA on a representative set of problems, preferably another.
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 nontrivial task, since the parameters interact in a complicated way. It, from a partial 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-consuming task. Furthermore, it does not provide a fair basis for copying the performance of the GA to that of other approaches.

For main approaches for selection of parameter values can be identified in the literature, which will be outlined in the following:

1. Ignore the problem and do problem-specific tuning. Fortunately, it is not hard to implement 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 (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 at in a 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 consuming. 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 1986, De Jong suggested a set of general applicable parameter values based on extensive work with test sets of functions, which is still widely used [De Jong 75]. Later, Grefenstette suggested a set of parameter values generated by a meta-GA [Grefenstette 88], which were shown to outperform De Jong's values. Grefenstette's settings have been widely used by other researchers and are generally accepted as reasonable defaults.
rains the easiest and best method when the feasible region is large
relative to the total domain or when the problem is "smooth". Other-
wise, when possible, constraint enforcement is "probably the best way to
tackle constraints". This viewpoint is similar to the experience of the au-
thor of this thesis. In a Masters project the highly constrained problem
of VSI flight-planning we approached using a GA based on the parity
method. The algorithm never worked well, since it turned out to be not
to impossible to find suitable values of the weights of the parity terms.
In other words, the modified cost function 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.

Tableaux approaches for constraint handling but quite fit into the
discussion, since they rely on constraint enforcement while still being quite general. The GEnCO system presented in [Myklevich 91,
Myklevich 92] is a general GA system for numerical optimization
problems with a set of linear constraints. Constraint satisfaction is
enforced by a scheme which relies on the totality of the feasible region.
Consequently it can not easily be generalized to nonlinear constraints.
In [Myklevich 91] GEnCO is compared to a GA using the parity
method, a specialized GA using constraint enforcement (GEnCO2)
and a package for numerical programming (GEnCO), on a test site of
six functions. The best results are obtained by GEnCO2, which is
slightly better than GEnCO. Moreover, GEnCO is dead superior
to GEnCO, while the parity-based GA fails to find any feasible solutions
at all.

The other approach presented in [Shumer 91]. The basic idea
is to execute the GA several times, each time satisfying a new yet satisfied
constraint. In the first execution the cost function is simply \( p \)
the final population is (ideally) solutions satisfying the first constraint.
Further starting point, the GA is executed again this time with \( p \)
constraint. Solutions violating the first constraint are discarded 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. 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 com-
putationally expensive, and the success of the approach relies on diversity
in the population being carefully maintained in each execution.
should state 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-dimension problem, but it is noted that the technique cannot be easily generalized. As such, for dimensionality pricing of penalty terms is proposed in [Sith9], and pricing results are reported. However, only one penalty instance is considered with the only one type of constraints, i.e. $k = 1$.

<table>
<thead>
<tr>
<th>Property</th>
<th>Penalty method</th>
<th>Enforcement method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generality of approach</td>
<td>High</td>
<td>Low, none</td>
</tr>
<tr>
<td>GA theory applicable</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Agri time related</td>
<td>Short</td>
<td>Long</td>
</tr>
<tr>
<td>Characteristics of function optimized</td>
<td>As cost function</td>
<td>Worse than cost function</td>
</tr>
<tr>
<td>Size of search space</td>
<td>Large</td>
<td>Small</td>
</tr>
<tr>
<td>Some feasible solution guaranteed</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Weight adjustment problems</td>
<td>Likely</td>
<td>No</td>
</tr>
<tr>
<td>Solution quality obtained</td>
<td>Problem dependent</td>
<td>Problem dependent</td>
</tr>
<tr>
<td>CPU-time requirement</td>
<td>Problem dependent</td>
<td>Problem dependent</td>
</tr>
</tbody>
</table>

The 31: 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 [Sith9] the quality solution(s) of highly constrained problems tend to lie on the boundary of the feasible region, and therefore, many neighbors 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 [Richardson9, Sith9]. In terms of search, the agent is that the feasible solutions are at least a relatively low portion of the total solutions, which should be put together to form the feasible global optimum [Sith9]. Consequently, it may be difficult for a GA-based constraint enforcement to find a path to a good solution, let alone a global optimum. Sith and The [Sith9] further points out that due to the complexity of constraint enforcing decoding and operators, these questions can be the bottleneck of the search, which is another argument in favor 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 inevitably depends on the specific problem. According to [Shemer9] the penalty method...
Clearly the two methods can be combined so that some of the constraints of a given problem are handled by parity 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 researchers might agree with the view to be presented. The main points of the discussion are summarized in Table 31.

The parity method is the most general, since it only modifies the cost function while all other parts of the algorithm remain unaltered. For the same reason, existing Genetic algorithms are applicable, while this is not the case when a specialized representation and/or specialized operators are used to enforce constraint satisfaction. Such representation and operators are further more trivial to design and consequently constraints 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 parity 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 any genotype to the scanpoint in the phenotype space, perhaps in a very "invariant" way. Here, 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 decreasing fraction of the domain.

The parity method requires the design of suitable parity functions and corresponding weights, which is not a trivial task. If penalties are too low no feasible solution ever be found while this is guaranteed by constraint enforcement. To high penalties may turn the optimization into a search for a feasible solution only while not being able to distinguish the quality of different feasible solutions. Therefore, if the parity functions differ 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 parity terms changes during the optimization process. To overcome this problem the weights need dynamically adjust. Some general guidelines for the design of parity functions are given in [Richardson 8]. Try conclude that a good parity function should not just count the number of constraint violations. Instead, it should direct the distance from a feasible solution. That is, the parity
4. Rather than initializing the population randomly, it can be seeded with individuals generated heuristically, as discussed in [Shultz 91, Grefenstette 87], and others.

GA exploiting problem-specific knowledge is sometimes referred to in the literature as hybrid GAs, knowledge-augmented GAs or Evolution Programs. In [Mihaleviciu 92] a case study is presented in which new ad hoc problem-specific knowledge is incorporated into a specific GA. At one level, 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 sacrificed. Furthermore, any knowledge of algorithmic properties obtained through theoretical analysis as discussed in Section 3.3 will be sacrificed. Alas, the time it takes to develop the algorithm will increase significantly [Mihaleviciu 92].

3.4.3 Constraint Handling

Since almost all real-world problems have trivial constraints, techniques for constraint handling in GAs is a very important, but almost unexplored topic. Either researchers interested in these techniques are twinned approaches to constraint handling in GAs:

1. Ternary CRs. 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 \( c^- \) of the form

\[
    c^- (s) = c (s) + \varepsilon \sum_{i=1}^{k} \lambda_i p_i (s)
\]

where \( s \) denotes a solution, the function \( p \) assigns to each constraint \( i \) a weight \( \lambda_i \) \( i \geq 0 \) that measures the degree of violation of the \( i \)th constraint, and \( \varepsilon \) is a weight determining the relative importance of violation of the \( i \)th constraint and equals 1 for minimization problems and \(-1\) for maximization problems.

2. Constraint satisfaction. Infeasible solutions are added. Any feasible solution is ever considered. Constraint satisfaction can be enforced at all times by using a representation in which a solution satisfying (some of) the constraints can be expressed and/or by using a decoder with圮 impairing constraint violation, as this improves the quality of the genetic operators, and helps genetic operators to generate 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 Laurence Davis. In [L Davis 8] he wrote:

"...it has seemed true for some time that we cannot solve real-world problems with binary representation and an operator set consisting only of binary crossover and binary mutation. One reason this is that many real-world problems have 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 certainly has borne out in my experiences applying genetic algorithms to industrial problems. Binary crossover and binary mutation are knowledge-blind operators. Hence, if we wish to add knowledge to our genetic algorithms, they are likely to underperform analytically. Nevertheless, the optimization algorithms that do take account of such domain knowledge.

[...] I believe that genetic algorithms are the appropriate algorithms to use in a genetic real-world application. I also believe that one should incorporate real-world knowledge in one's algorithm by adding it to one's decoder or by encoding 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 omits the best individuals using a heuristic to improve the fitness of the produced offspring. This is discussed in e.g. [Goldfeder 87, Goldberg 89, Davis 91].

3. New operators can be added which perform local optimization of a given individual using any problem-specific methods available. This is investigated in e.g. [Shh 87].
presented in [Bai et al. 8], but as noted by the author the approach has some serious limitations. One of these is that to add visiting all points of the search space qistais has to be estimated by a sampling method for which no alternative measure is provided.

A technique for reducing qistais by increasing the size of the representation is presented in [D. Easley 8]. However, this method is limited to continuous optimization problems and the price paid for losing qistais is that of a much larger search space. There is no general applicable technique available to facilitate the design of a low qistais 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 QAS.

As for the second property mentioned above, the desirable load-distance between mutually dependent genes, the situation is somewhat better. In general, it is not known what the genes are related and consequently it is generally impossible to determine a good ordering statistically. Instead, the best approach is to add a so-called reordering operator to the set of genetic operators. As the QAS evolves, such an operator reaches given genotypes, thereby attempting to group together the mutually dependent genes dominantly [Goldberg 89]. 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 of independent genotype effectively represents a significant expansion of the search space. An alternative to reordering operators is presented in [Goldberg 89]. 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 a rather small problem. Another approach is presented in [Ha 98], which is applicable to a certain class of graph problems. It is assumed that the genotype is a bitstring in which each bit selects or deselects 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 empirical results are reported, which show that the preprocessing significantly improves performance for certain graph types, while it has no effect on other graph types.
processes as expressed in the Halding Block Hypothesis and the Sura Theorem.

If the epistasis of a representation is very low, i.e., there is little or no nonlinear interaction between the variables, the problem is easy in the sense that it can (almost) be solved by optimizing a single variable at a time. Conversely, the GA will be at its best in solving the problem. On the other hand, if epistasis is extremely high, the GA will fail to progress substantially in a useful manner, and the search becomes random. Here, relative to capturing 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 known only to the designer. This coincides with the results of Section 3.1.2 regarding the application area of GAs, since any encoding of a difficult optimization problem may have some degree of nonlinearity. Yet, the designer's task is to optimize performance by devising an encoding giving the lowest possible epistasis level.

<table>
<thead>
<tr>
<th>Hillclimbing</th>
<th>GA</th>
<th>Random Search</th>
<th>Search Method</th>
<th>Epistasis Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>low</td>
<td>high</td>
<td>very high</td>
<td></td>
</tr>
</tbody>
</table>

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, We and Beinh have shown in [WBE 01] 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. This theoretical result cannot be directly applied in practice since the construction of the representation effectively requires the given problem to be solved. However, a key 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.
Many ways of representing a solution. Admittedly, it is often very hard if not impossible to obtain a completely accurate performance or a real-world problem from a binary encoding. A lot of work has been done on coding based on integers, e.g., [Bahl et al. 92] and floating-point values, e.g., [Jalava 92, Mihaljevic 92]. But when the properties of a problem are not linear, as for example in certain real optimization problems, different representations based on e.g., graphs may be more suitable. In [Iba et al. 92] a binary tree is part of a genotype.

Tody the GA community is divided in these two beliefs that the binary representation should always be used, and these who are in favor 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 (un-binary) representation, performance is often gained at the cost of sacrificing the general applicability as well as the theoretical foundation of the algorithm. Therefore, right age that there may be theory to sacrifice anyway.

However, the Heidelburg Bok hypothesis and the Shm theory still provide useful guidance on how to design a good (un-binary) representation for a real-world problem. The Shm theory suggests that a good encoding is one which has two properties, which will be explained below.

1. The dependence between the contents of the representation is small.
2. The distance between mutually dependent contents is small.

The contents 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 interdependence between contents is called *pistasis* [Daley 92]. There is no generally accepted definition of the term *pistasis*. But the idea is that low *pistasis* refers to a low degree of mutual gene interdependence, that is, the fitness of the individual is close to being a linear function of the gene values. Similarly, high *pistasis* means that fitness is a highly nonlinear function of the gene values. To empirically define “mutually dependent contents” refers to contents, which by their interaction affect fitness significantly. In other words, the necessary criteria of a good encoding is that mutually dependent genes are close together and that *pistasis* is as low as possible. If a genotype satisfies these criteria, it allows substitution usefully
is assumed to be infinite. Also in [Goldberg 8] and [Holland 8] only two possible individuals, '0' and '1', exist. Generally, the insight gained so far from the initial 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 need to be addressed. For example, to avoid quickly getting trapped in a (poor) local minimum, it is also important to avoid very slow convergence. The variance of the fitness values needs to be controlled through a GA creation. For this purpose, the fitness measure is not often defined as a straightforward 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 8].

This section focuses on four specific problems that are especially important for the algorithm presented in this thesis, but which may not be addressed or generally accepted solutions exist. On the contrary, these problems are open research questions.

The key point in designing a GA is the design of a suitable genotype, which is the topic of Section 3.4.1. Exploration of problem-specific knowledge is discussed in Section 3.4.2. The topic of Section 3.4.3 is techniques for constraint handling since in real-world constraints are usually 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 focused on algorithms based on binary representations for many reasons [Deb 91]. The binary representation is due to the view of the GA as a deterministic, general-purpose approach to optimization. Miller's original work [Miller 7] focused primarily on binary representations and the implementation of the GA as a binary representation. A natural question is: What is a good representation? It is generally believed to be preferable from a practical point of view since it minimizes the number of solutions, 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. It is shown that the number of schema
fully 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 GA.

To fully exploit the effect of implicit parallelism should be advanta-
ges to have as many schemas as possible. Given an alphabet of
cardinality $k$ and a string length of $l$, the size of the search space
is $k^l$. The number of possible schemas is $(k + 1)^l$. For a fixed search space size it can then be seen that the minimum
number of schemas 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, [Mairesse 89] interprets schemas differently and
concludes that the binary alphabet does not minimize the number of
schemas. On the contrary, an alphabet in correspondence with the initial alphabet of higher cardinality has more expressive power and represents more schemas. Goldberg has later presented arguments why high cardinality
alphabets may be useful, this attempt to account for the different viewpoints [Goldberg 91]. Further critical view of the traditional schema
theory is presented in [Grefenstette 89]. Here it is pointed out that Holl-
land's $O(n^{-3})$ estimate of implicit parallelism is impractical for the
individuals and have only hold in the first few generations. It is also
noted that the Schema Theorem is formulated in terms of the fitness
function rather than a general function. This is problematic since the
fitness function is part of the GA itself and as such is a priori determined;
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 behavior 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 GA. Such an analysis is presented in e.g. [Goldberg 87, T. E. Bäck 91, Nx 92, Sank 91, Him 91]. 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 exam-
ple, in [Nx 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 \( t \) with a representation \( H \). 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

\[
E(m(H, t + 1)) \geq m(H, t) \frac{\bar{f}(H)}{\bar{f}} \left[ 1 - p_c \frac{\delta(H)}{l} - o(H)p_m \right]
\]

This important result is called the Schema Theorem. An derivation of the theorem can be found in [Goldberg 89]. If \( o(H) \) and \( \delta(H) \) are small, and \( \bar{f}(H) > \bar{f} \), then \( m(H, t + 1) \) is expected to be multiplied by a factor greater than one. In other words, the Schema Theorem states that in the GA, the expected number of individuals representing a schema with short defining length, lower or above average fitness, will increase exponentially.

What does the Schema Theorem have to do with Categorization?

The answer depends on the validity of The Building Block Hypothesis [Goldberg 89], which states that by finding good partial solutions, called building blocks, good complete solutions emerge. By further assuming that good building blocks correspond to schemata with short defining length, lower or 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 here we have an explanation by the SCAwds.

The Schema Theorem has been generalized in various ways, notably by Vose. In [Vose 91], 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 91], it is also independent of the specific genetic operators. The two last terms of the theorem ensuring 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 Theorems and discuss another important phenomenon. Since a binary string of length \( l \) represents 2 \( 2^{l} \) distinct schemata, some are between 2 and 2 \( 2^{l} \) schemata will be represented in an application of size \( n \). However, crossover destroys schemata of relatively high defining length, hence not all schemata will be use-
**Figure 32** 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 ad "tuplete" theory which fully explains the properties of the GA. However, hypothesis have been formulated that at least partially explain the behavior of GAs and provide insight into the mechanism of the algorithm. This Section briefly presents the classical explanation of the simple GA from the previous Section and comments on recent theoretical developments.

In [11] and [12], the search performed by the SAs is investigated in terms of sampled schemata, or hyperplanes. Let \( v = v_1 v_2 \ldots v_n \), \( v, v \in \{ 0, 1 \} \), denote the genotype of an individual of the GA and let a scheme be a string of length \( l \) over the alphabet \{ 0, 1, \# \}. The symbol \# indicates 0 or 1, and a given string (genotype) is said to represent a given scheme if it matches the scheme at all its fixed positions. E.g., if \( l = 6 \), the scheme \(#0\#0\#0\#\) is represented by e.g. 000111 and 110110. A schema specifies a hyperplane in the search space corresponding to the set of strings which represents the scheme. The idea of 112 aids in that the strings present in a population estimates the fitness of the hyperplane they represent, that is, they estimate the average fitness of all possible strings representing the hyperplane. 112 aids show 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 occupy 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 \( d \ (H) \), is the distance between the first and last fixed position in \( H \). E.g., \( o (\#0\#0\#) = 3 \) and \( d (\#0\#0\#) = 4 \).

Therefore, let \( \bar{f} = \frac{1}{n} \sum_{i=1}^{n} f_i \) be the average fitness of all individuals in
in Fig 32. When crossover is not performed copies of the selected parents of individuals are added to $\Pi_n$. Following this step called reproduction, every individual is subjected to possible mutation. The mutation operator performs point wise mutation i.e., given a bitstring each bit is independently mutated with a given small probability $p_m$, as illustrated in Fig 32. Each generation is completed with an update of $\Pi_{cur}$ and evaluation of all individuals, as a basis for the selection to take place in the next generation.

```
generate($\Pi_{cur}$);
calculate($\Pi_{cur}$);
repeat $G$ times
  $\Pi_{new} := \Phi$ ;
  repeat $n/2$ times
    select $\pi_1 \in \Pi_{cur}$, $\pi_2 \in \Pi_{cur}$ ;
    if random then
      crossover($\pi_1$, $\pi_2$, $\phi$, $\phi$);
      $\Pi_{new} := \Pi_{new} \cup \{ \phi, \phi \}$ ;
    else
      $\Pi_{new} := \Pi_{new} \cup \{ \pi_1, \pi_2 \}$ ;
    end
  end
end
\forall \pi \in \Pi_{new} : \text{mutate}(\pi) ;
$\Pi_{cur} := \Pi_{new}$ ;
calculate($\Pi_{cur}$);
```

Fig. 31: The SGA. The population size $n$ is assumed to be even.

Crossover is the main operator of the GA. By combining good partial solutions, even better solutions will often emerge. Here, 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 ensure that this lost information can be recreated.
are either not applicable or have difficulties finding a good solution. A
will be clearer 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 prespecified distance from the global optimum. However,
from the practical viewpoint 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 called
SA. Although the practical value of SAs is very limited, as will be dis-
cussed in Section 4, it is a good idea to gain insight into the basic
principles of the GA. Therefore, the original theoretical arguments
on why GAs work, which will be presented in Section 3.3, are based on
the SA.

In SA, the genotype of an individual is simply a bitstring of fixed
length $l$ and the fitness of any individual is defined as a function with
given the phenotype represented by a bitstring returns a positive, real
value. The SAs outlined in Fig. 31. Firstly generate generates the
initial, empty population $P_c$. This consists of $n$ random bitstrings
of length $l$. The population size $n$ is left fixed throughout the process.
Lastly evaluate evaluates the fitness of every individual. At 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 $P_{new}$ is generated. A pair of individuals,
$\pi_1$ and $\pi_2$, is selected from $P_c$, a total of $n/2$ times. The selection
is proportional to the fitness of the individuals, that is, individual $\pi_k$
is selected with probability

$$f_k = \frac{f_k}{\sum_{i=1}^{n} f_i}$$

where $f_k$ denotes the fitness of individual $\pi_k$. Two individuals are
selected independently and any individual can be selected any number
times in the same generation. The parameter $p_c$ is the crossover prob-
ability. Finally rand(x) returns true with probability x. When crossover
is performed, the algorithm generates two offspring $\phi_1$ and $\phi_2$, which are then added
to $P_{new}$. Simple one-point crossover is performed by selecting a cross-
point at random and then replacing the two substrings as illustrated
considers only a single solution at a time. Clearly, both EA and SA are based on rules of nature simplified to the extreme where it can hardly be recognized. In this sense, the analogy to phenomena 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 pointed out in the literature is the robustness of the algorithm [Goldberg 89, Davis 89]. A contrast to the vast majority of other methods, the GAs do 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 GAs are robust in the sense that it can be used for optimization in highly complex and irregular search spaces. This claimed generality of the GA is supported by the literature, in which applications in diverse fields are reported. References to applications of GAs in e.g. biology engineering, computer science, business and social science can be found in [Goldberg 89]. In [Staats 98] one finds 40 references to papers on EA and applications are given, and [Nissen 98] lists about 20 references on EA in various science. The specific application areas of GAs include neural network optimization, combinatorial optimization, image processing, pattern recognition, design and machine learning [Goldberg 89, Davis 91]. While genetic algorithms still being used at universities, an increasing number of projects are reported from industry [Goldberg 99]. For example, General Electric is using a GAs-based system to design turbines and jet engines, Hughes Missile System Company, in California is using genetic programming for infrared image target discrimination and Allied Signal, General uses a GAs to solve problems arising from seismic surveys related to oil exploration [Goldberg 99].

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 not likely show superior performance, both in terms of solution quality and runtime. Other, the natural application area of the GA is for very hard problems for which other methods
current research topics. Sinceillard's original work [Illard 73] is very
general and quite fundamental it is hard to relate to GAs.

Te GAs belong to a well-established class of algorithms the Evolutionary Al-
gorithms (EAs), surveyed in [Illard 98]. Although the GAs by far the
most popular EA this class also consists of Evolution Strategies (ES)
and Evolutionary Programming (EP). Te main feature of EAs is that
they all maintain a population of individuals and apply various operators
to evolve individuals of increasing quality over time. It there are also
a number of significant differences between the algorithms. First of all,
the crossover is the most important operator of the GAs and mutation
is only considered a background operator, as will be discussed in Sec-
tion 3.2 in ES and EP algorithms mutation is the main operator and
crossover is less important. Te EP algorithm doesn't have a crossover
operator at all. Furthermore, while the parameters of the GAs are fixed,
yea are dynamically adjusted in ES and EP. Te ES algorithm dif-
fers from both GAs in two major ways. Firstly, the fitness function equals the objective function while in EP and GA, the (rela-
tive) fitness of each individual is a (non-trivial) computation based on
the objective value of all existing individuals. Secondly, selection is de-
terministic in ES algorithms while it is probabilistic in EP and GA.
Of course, the discussion in the EAs has given here just outlines
the general differences. Given a specific EA it might clearly belong to
any of the three categories as described here. Te current state of EAs
research is surveyed in [Illard 98].

An interesting special case of the GAs is Genetic Programming (GP),
introduced by [Koza 92]. Given a specific problem the idea is to let
the GAs evolve a computer program which solves the problem
sufficiently well. An individual is a program and is evaluated
by executing the program and measuring how good the program
at solving the problem. By using the genetic operators as usual, the GA
evolves programs which perform reasonably well with respect to solving
the problem. Te 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 intro-
duced in 1983 by [Kirkpatrick 83], which is inspired by thermodynamics. O-
timization is performed based on an analogy to the process of cooling
down a solid in such a way that thermal equilibrium is attained. Te
two fundamental differences between the GAs and SAs is that the GAs
contains a number of solutions simultaneously while the SA algorithm
the capability of adapting to the environment in which they live. The fittest individuals have the highest probability of survival and reproduce the most. The process of sharing their traits so that highly fit individuals usually pass on highly fit offspring. Therefore, during the evolution process, the fittest individuals tend to increase in numbers while the less fit individuals tend to decrease. The 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*.

Artificial 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 initiates sexual reproduction. The crossover operator is designed in such a way that the generated offspring resembles the parents. 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 a mutation or random change, by the mutation operator. After being simulated a number of generations, highly fit individuals will emerge, corresponding to good solutions to the given optimization problem.

Adaptation 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 solution in terms of their genotypes, while fitness is measured in terms of phenotypes. A function called the decoder converts the phenotype corresponding to a given genotype.

Introduction to GA can be found in many texts, e.g. [Eiben 94, L Davis 87, L Davis 91, Michalewicz 94], and [Goldschmidt 80]. The book contains the reference textbook. Another two good papers [D Easley 93, D Easley 94] provides an introduction to GA 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 GA. Section 3.1 introduces the basic idea of GA. The simplest possible GAs presented in Section 3.2 and Section 3.3 presents the theoretical aspects as to why the simple GA works and also discusses the current status of GA theory. Those familiar with GA 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 GA, 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 Holland, whose Ph.D. thesis from 1975 [Holland 75] is considered the origin of the field. One of the main application areas of GA is that of optimization, which is the only application area considered in this thesis.

3.1.1 The Basic Idea of GA

The GAs is inspired by the process of natural evolution studied in population biology. In nature, the individuals constituting a population have
that the entire rating area is divided into sub-nets by this scheme, thus completely eliminating the need for sub-bins. The obvious drawback of restricting the search space to sliding structures is that if the optimal layout is not a sliding structure, it will never be found. Sliding structures are briefly considered again in Section 5.2.3.
rating in itself requires the solution of perhaps hundreds of mutually 
dependent, N-body problems, although many of these problems may 
have relatively small search spaces. Grad rating as well as switch 
box rating are N-body (Symsl 8), and so is for example via 
miniization (Ndai 0 8). In addition, the problems are mutually 
dependent, and hence relies heavily on estimate of what will happen 
succeeding steps. In other words, the cost functions involved are not 
accurate, but relies on the accuracy of the estimate. These inherent 
problems have two important consequences: Firstly, certain steps of the 
layout synthesis process, or perhaps the whole process, typically need to 
be iterated a number of times to obtain a satisfactory result, as 
illustrated in Fig 22. For example, as mentioned previously it may not 
be possible to complete the rating of a given placement without 
back and adjusting the placement. Secondly, the quality of solutions ob 
tained for some intermediate step of the process cannot be accurately 
caloated immediately. For example, two different placements cannot be 
objectively compared without actually completing the rating of the layouts.

In general, the same steps are performed to move it from the draft 
layout quality. If for example a very poor placement is generated, it can 
not be compensated for in succeeding steps, in some way 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 focuses on placement and global rating rather than 
detail rating. Especially the floorplanning and placement problem are very 
hard, and consequently they are often solved manually (Sawai 9). 
A reason previously, a key problem here is the estimation of the 
moderate rating area. Naturally, as the layout synthesis process proceeds, 
estimations become increasingly accurate.

Sliding structures is a class of building blocks layouts which have been 
very popular, since it simulates a set of the problems described 
above. Aslicing structure is a building block layout, which can be 
recursively partitioned and sliced by a sequence of horizontal and vertical lines 
each of which goes all the way through the layout, until no more than 
e blocks is present in each partition. For example, the layout of Fig 23 
is a sliding structure. Horizontal line can separate A and Bfrom 
blocks C Dand E Tin, vertical lines can separate Dfrom 
blocks C and E and finally Dfrom E. Aslicing structure simplifies the rating 
step, each line corresponds to a dump, and a feasible rating order is 
the inverse of the order in which the lines were made. Especially note
dram routers are capable of rating the lower bound given by the
dram density in almost all cases occurring in practice. And when
they fail, only one or two additional tracks are needed. For this reason,
the dram 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 dram and the dram density.
As illustrated in Fig. 2.4, if \( n \) nets are present, the density can be any
integer between 0 and \( n \).

After completion of the routing step, the layout is functionally com-
plete. However, a final step of postprocessing is often performed in which
various kinds of final optimizations are attempted. One common
kind of postprocessing is \textit{compaction}, where the layout is expressed in
one or both dimensions subject only to the design goals. Compaction
reduces the total area and may improve performance by reducing wire
lengths as well. Other kinds of postprocessing are via minimization and
reassignment of the layers of some wire segments. The total layers used
for routing have different electrical properties, and more or fewer may
be preferable at times. In Fig. 2.3(b), some wire segments of layer 2 have
been reassigned to other layers, allowing layer 4 to be placed on top of it.
The resulting unused track allows reduction of the total area by subsequent
horizontal compaction.

This step of the layout synthesis process described above is com-
putationally hard to solve. Formulations of the partitioning and floor-
planning problems are all \#P-hard [Shi 80; Barth 81]. One
important formulation of the global routing problem is equivalent to that
of finding a minimum Steiner tree in a graph, which is \#P-hard for a
single net having more than two terminals [Kip 72]. Hence, global

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure2.4}
\caption{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.}
\end{figure}
or a time using a dural rotor or a switchbox rotor as appropriate. 

As in a transformer, the current flows in the opposite direction 

at which the core is at its peak efficiency. The dural rotor 

assumes that it can direct itself the correct position of where 

net gates and/or leaves the dural. When the position of a net 

crossing the board between neighboring dural, this of course 

affects both dural and consequently it imposes a partial ordering 

of the rating of dural. For example, in Fig 23 the dural between 

boards 1 and 2 to be rated before the dural between boards 2 

and 3 since otherwise the terminal position along the boundary 

of the latter dural is not fixed before the dural is rated. Given 

arbitrary routing and a switchbox dural ordering must exist and 

then become necessary to introduce a switchbox. Therefore, the dural 

ordering is also determined by the earlier routing and switchbox step. 

In Fig 23(b), a possible routing order is to first rate all vertical dural, 

i.e., those with terminals along vertical sides, in any order, and then rate 

all horizontal dural, in any order.

The objective of the detailed rotors is first of all to complete the 

routing within the available area. Dural rotors also take into account 

to minimize the total wire length together with 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 

required to route a net between two routing layers, as illustrated 

in Fig 23(c) and (d). Due to the poor electrical properties of vias, 

their usage are often minimized. For a given dural, a set of design 

rules and a routing layer, a two-dimensional lattice can be defined in a 

dural which determines the minimum spacing needed between vias 

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 

dural rotor increases the possibility of improving the layout in the 

subsequent processing step, which will be described later.

When considering a specific column, a lower bound on the number 

of vias which has to cross that column is the number of nets having 

terminals on both sides of the column. When minimizing this quantity 

over all columns, the resulting value is known as the channel density. 

Given the dural density, the number of available routing layers and 

the design rules, the minimum width needed by any rotor to 

implement the routing can easily be computed. This state-of-the-art
The wiringnet contains 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. Here, 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 of the estimated path 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. 23 (b). In this example all routing regions are small.

\[ \text{Figure 23} \quad \text{A macro-cell layout with five macro-cells denoted A through E. The numbers 1 through 4 are net-identities, positioned at the terminal locations. (a) shows the layout after placement, and (b) after channel definition and global routing. The dashed lines indicate the borders of the channels. (c) is the result after detailed routing, and (d) is the final layout after a change of layer assignment of net 2 followed by compaction.} \]

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 circuits, all interconnections may be described 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 desired, that is, all flexible blocks are turned into fixed blocks. Secondly, an absolute position and orientation is determined for each block. The spring between blocks should be sufficient to allow for additional interconnections to be implemented in succeeding steps. In the literature, the routing optimization criteria used are minimization of total estimated area and/or total estimated interconnect length. If one or more of the blocks are flexible, then this step is referred to as floorplanning, while if all blocks are macrocells, this step is called placement. Here, placement is as 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 be possible to implement the interconnections later on without altering the placement. Fig. 23 (a) shows a placement of the macrocells.

Floorplanning the purpose of the routing step is to implement all interconnections between the macrocells in accordance with the netlist. As indicated in Fig. 22, routing consists of three parts: 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 are located at the perimeter 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 with which is not a channel, is called a switchbox. The objective of routing region definition is to divide the routing area into a few regions as possible. Especially, the number of switchboxes should be minimized since a switchbox is much harder to route in the later detailed routing step than a channel.
design or not, if the design is hierarchical, it will consist of building blocks from certain lead.

2.3 Building Block Layout Synthesis

Fig. 22 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 a time although they are mutually dependent [Shuai 9].

![Diagram of layout synthesis process]

Figure 22 Overview of the layout synthesis process for building-block layouts. The arrows indicate the order in which the steps are performed. After each step the designer may proceed 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.

Not all circuits can be handled by the CAD tools as a whole, due to the required computation time as well as the memory constraints. Therefore, the first step of layout synthesis is partitioning, in which the design is divided into a number of subcircuits of manageable size. For large circuits the partitioning may be hierarchical, cf. Section 22. 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
rare requirents can ben. Produing is anther application area of saiautonad sign.

Anin reason why full-autonad sign increases the delduat time sigfigtly is that many of the siublens to be selved during the last synthesis phase are sigfigtly hard to solve thm the corresponing problem for saiautonad sign. For eample, interconnections in a standard cell layout is typilaly implemeted insahaw thet oly are dimiinued to be consdrat at a tine. In a building blok layout this problem is truly twoamial. Consecutly, ny of the key siublens of layout synthesis are selvedly tooo bad that are design style spad: Inproed CAD tool preformance is fitt of all medd for the full-autonad sign style. Inferntly, these tooo often preformance poor that their saiautonadparts, and this is one of the reasons why the work preented in this thesis is conctratd on the full-autonad sign style.

The 21 sumarizes the caprip of the full-autonad saiautonad sign styles. Generally speaking, rare requirents are impoed on the layout, the differenrs listed will be inreaingly precatet. Inerter wek, the caprip of The 21 still hold if e.g. “full-auton” is replacedly “standardcell” and “saiauton” is replacedly “gte ar-ray”.

<table>
<thead>
<tr>
<th>Caprip</th>
<th>Fd-ndle design style</th>
</tr>
</thead>
<tbody>
<tr>
<td>flexiblity ut. requirents</td>
<td>full-auton</td>
</tr>
<tr>
<td>preformance of circuit</td>
<td>full-auton</td>
</tr>
<tr>
<td>layout area</td>
<td>full-auton</td>
</tr>
<tr>
<td>cost per circuit</td>
<td>full-auton</td>
</tr>
<tr>
<td>delduat time and cost</td>
<td>saiauton</td>
</tr>
<tr>
<td>tool support</td>
<td>saiauton</td>
</tr>
</tbody>
</table>

**The 21: Comparison of design styles.**

Rel-vael circuit design often consists of a mixture of layout styles. Some parts of the layout my be constructed from library standard cells, vile othe, rare critical parts are full-autonad sign. To hide the compleity of large circuits, the design my be hierarchically structured into two or rare leods. A rational preiioy bloks of a full-auton layout my be constructed from a number of smaller bloks. Standard cells my also be groupet toger to form a building blok at the nee higher level of the hierad. So vather standard cells are used in a
cells. The cells are placed in rows as illustrated in Fig 21 (right) and they are almost always designed so that some of the wires, typically power supplies and data signals, are positioned at the same height in all cells. The wires are then automotively created by substrat of the cells. While a full-custom layout rarely consist of more than 50 cells, a standard cell layout may consist hundreds of cells.

![Diagram of full-custom and standard cell layouts]

**Figure 21:** *Left: A full-custom layout. Right: A standard-cell layout. Only outlines of cells are shown.*

The other semi-custom layout styles are 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 [Shoai 8].

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. Circuit performance is also obtained by a full-custom design, and it is the most flexible with respect to typical 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 most products and for circuits having to meet strict performance requirements, such as CMOS. The main drawback of the full-custom design style is that it increases the development time and has cost significantly. Therefore, a semi-custom layout style is generally preferred for circuits to be produced in reduced quantities, provided perfor-

---

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 satisfactory results.

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. 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 them. 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.

[Mid8] and [Vle88] are classical textbooks on VLSI design in general. Intermediate topics are discussed in [Lase8]. Silicon compilation is the topic of [Gisi88], which presents a number of specific silicon compilers. Another well-written textbook on layout synthesis is [Shua89].

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, semi-custom, or full-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. All of a full-custom layout is also called a block or a building block. Blocks may be constructed from smaller blocks or it may be designed manually or automatically, e.g., a multiplier generator. The blocks are arranged and interconnected. The characteristic feature of the full-custom layout is irregularity. Each block can be any size and in general, any rectangular shape. Either, the blocks can be placed at any position subject only to the limitations imposed by the design rules, as illustrated in Fig. 21 (left).

In contrast to the full-custom layout, the semi-custom layout styles introduce various degrees of regularity. Standard cells, gate arrays, sea-of-gates, and gate matrix layouts are all semi-custom styles. A standard cell layout is made up of standard cells, which are rectangular and have identical height but varying widths. Each cell is designed manually or stem from a cell library of predesigned cells. Circuit silicon compilers typically use libraries of a few hundred standard
or using abstrat) assigned to an appropriate layer. Similarly a transistor is specified as a specific location of several 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. A typical tool is the first design tool as silicon compiler is also often referred to as a programing 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 specify minimums of shapes, minimum distances between shapes of different 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 minimized. For example, criteria are important depending on the specific application. If the circuit will constitute the CPU of a computer, speed will be crucial. For satellite applications, low power consumption will be of major importance. For radar equipment, reliability is likely the highest priority. To control the program in a wiring problem, production cost per unit will be a major concern. Similarly, it is likely to be a very important requirement. The optimization criteria must be specified in the literature and also adapted in this work. It is a minimization of layout area and total wire length. Minimizing area means minimizing yield and hence minimizing cost. It also means minimizing the functionality which can be implemented on a single chip. Minimizing wire length to some extent means minimizing delay and hence minimizing speed.

Several other similarities between compilers and silicon compilers exist. However, there are also major differences caused by the transcoding difference of the complexity of the problem considered. As will be described in Section 2.3, the silicon compiler has to deal with a sequence of mutually dependent, nonlinear 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. Many 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 a reference for further reading. Section 2.1 introduces layout synthesis in general, and Section 2.2 presents the concepts of full-automated and semi-automated layouts. An overview of layout synthesis of building block layouts, which is the specific problem 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 a circuit, 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.

High-level description languages, of which VHDL is the most popular [Lipsett 84], is commonly used for the input high-level description. Alternatively, general purpose programming languages such as C or C++ can be used [Grai 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 can only refer to a set of predefined geometric shapes, each shape is assigned to a specific layer of the circuit, which is identified by a name or a color. Typically rectangles are the only shape allowed. Some layers of the circuit are used for interconnection, or wires, while connections of other layers may correspond to, e.g., a transistor or an electrical connection between two specific layers. Here, in the layout, a wire is specified as a set of rectangles (overlapping...
ideas on how to handle the most important identified problem of the proposed algorithm. Fully the main conclusions of this thesis are given in Chapter 7.

1.4 Description of Papers

A previous section, the thesis is based on research papers, for of which constitutes Appendix A through D. The following presents the papers by listing information on the authors, 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:

   Proc. of The European Design Automation Conference, Heurosky, pp 357-367, 1992


Appendix A presents an introduction of the algorithm described in the first paper. An extended version of the second paper constitutes Appendix B, the third 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 an approach. The papers assume
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 macrocell layout synthesis is given in Chapter 2.
This includes a brief account of layout styles, with particular emphasis
on the characteristics of the macrocell layout style. A detailed
account of the individual steps of the macrocell layout synthesis process is given,
covering the complexity of each step and the interactions between
the steps. Readers familiar with macrocell 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. To
this end, an overview of genetic algorithms, practical considerations of applying
GA and macrocell research topics. Readers familiar with GA can skip Sec-
tion 3.1, 3.2, and 3.3, while Section 3.4 discusses various design options
and considers the basis for later discussion in Chapter 5.

Related work is reviewed in Chapter 4. The presentation given is
not meant to be exhaustive, but describes the state of the art approaches
to placement and routing of macrocell layouts. Furthermore, Section 4.3 gives a brief
overview of previous applications of GA within
the layout synthesis area in general. The GA has been applied to e.g.
standard cell placement and routing.

Assumptions of the untested 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 op-
tions in Section 3.4, the important characteristics of the algorithms
are summarized. The algorithms have some common properties, which
are believed to be the main reason 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. And from the fact that these kind of analysis are very hard if not impossible to apply for CA, such performance measures are not the immediate interest of the CA 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 support each other. It 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 problems as considered in practice can always be solved in say a couple of seconds, the design engineer will not vary whether the complexity of the algorithm is $O(n \log n)$ or $O(n)$. Later on (s)he might be spending (s)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 elucidated implementing each algorithm into a tool to a set of existing CA tools. The algorithms are then tested using benchmark problem instances whenever possible, and the obtained quality for the benchmarks as well as absolute runtime are compared to those of current state-of-the-art tools. Comparison is done in a state-of-the-art approach, no matter if it is based on a completely different strategy such as hardware and/or simulated annealing. Again from the CA engineer's point of view the approach is interesting if and only if it is competitive to other approaches, whereas typical CA performance measure such as online or offline performance is of immediate interest on its own.

### 1.3 Organization of the Thesis

The thesis is based on papers written during my PhD 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 assuming that the reader has
does not capture the temperature dependency of delay. There is also
generic 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 discuss
the topic.

A described in Section 1.1 the overall purpose of this work is to in-
vestigate if the application of GA can improve the performance of CA
tools available to design engineers. The application oriented purpose of
of course affects the approach taken, which also becomes application or-
nated. The remaining of this Section discusses important consequences
of the application oriented approach.

When considering performance of the developed algorithm, the ob-
tained layout quality is assumed to be more important than runtime. Of
course runtime should be within reasonable bounds to minimize the de-
velopment time of a circuit. If a circuit is to be reproduced even
a slight improvement of a quality factor such as area will mean a signif-
ificant economic advantage. Hence, designers will not likely be willing
to spend the extra development time caused by slow CA tools. Fur-
thermore, when producing large, high-performance circuits, the designers
may not even have a choice. Just to obtain a layout satisfying the re-
quirements with respect to, say timing and area, the designers may have
to use the tools producing the best possible layout quality, no matter
what their runtime requirements are. It should be emphasized that the
priority of quality is being more important than runtime does not mean
that runtime is unimportant. It is merely a matter of priorities.

The application oriented approach under the application of GA
will. The GAs have been chosen as the subject of this study because
of an interest for the GAs itself, although I did find the basic idea of the
GAs intuitively appealing. However, the main reason to inves-
tigate the GAs 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 GAs widely accepted as being able to generate high-quality so-
lutions, while they often have problems coping with respect to runtime.
These characteristics meets or priorities.

Performance evaluation is strongly impacted by the application or-
nated approach. From a theoretical point of view the performance of
an algorithm can be evaluated in terms of convergence proofs, hard
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 down problem. Firstly, distributed as well as exist in GAs for macrocell layout problems. Chapter 4. Study the GA has been successfully applied for several other highly complex optimization problems. Section 3.12. 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 macrocell 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 macrocell layouts, when the main objective is high-quality results?
- Which performance can be obtained?
- Are there any algorithm design principles, which ensure to yield 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 focuses on the collocated optimization aspects of layout generation rather than the physical aspects. A layout is usually viewed as a set of twodimensional geometrical objects which should be organized in the plane such that some measure of quality is optimized. The quality measure is defined in terms of sizes and shapes of geometrical objects, distances between different objects, etc. This is in contrast to a physical point of view in which a layout is a set of interconnected transistors and the conductors discussed are rise and fall time, capacitance, distance, and so on. In two viewpoints, or “views,” one can of course very tightly related and may concepts are (completely or partly) transferable from one to the other. For example, the signal propagation through a wire depends on the geometrical 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 consists of a number of interconnected blocks, or cells, each of which implements a 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. General EDA tools packages include libraries of so-called standard cells, which implement a wide range of relatively simple and common functions. Standard cells are uniform and therefore relatively easy to put together to form larger blocks. More standard cells simplify and speed 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. Uniform blocks are called macrocells, and a level of the hierarchy consisting of macrocells is called a macrounit layout. To create a standard cell, a macrocell, and customized cell, 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 macrocell type from a lower level in the hierarchy than if standard cells were used.

Most EDA tools are aimed at rapid prototyping for microcontroller layout styles, so that the same subproblem of the layout generation process is addressed by different tools, depending on the current layout style. As a result, the best tools have been developed for standard cells, since a standard cell layout is easier to generate than a macrocell layout implementing the same functionality. Due to the increased complexity of the subproblem to be solved when using the macrocell design style, manual intervention is required more often to obtain a satisfactory result. In other words, the macrocell layout tools are the ones that need improvement the most, and are also the ones with the largest potential for improvement. This is the reason why this thesis focuses on tools for the macrocell layout style rather than standard cell layouts. Furthermore, the specific subproblems shown in Figure 1.1 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 algorithms for solving certain subproblems in the 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 1950s, a typical microprocessor such as the Intel 8008 consisted of about 5,000 transistors while in 1988 Intel's state-of-the-art processor Pentium contained 31 million transistors on a 17.2 by 17.2 mm area. This extremely rapid development will continue at least for some time to come. Because of the complexity of today's circuits, designers are totally dependent on powerful CAD tools to facilitate a (semi)automated transformation of an abstract 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
Acknowledgments

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 name 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 Evi for her all-out support, which included allowing me to go to Milga for a one-year period while she had to stay in London.

At Aarhus University I would like to thank my supervisor Peter Miller-Nielsen for being a very inspiring adviser and for always supporting and encouraging me. Also thanks to Ole Gøtze, Mogens Gjøp and Tanja Mielke for my constructive discussion suggestions.

At University of Milga I owe thanks to Frank Mainke for welcoming me in his group and for his useful suggestions and encouragement. Also thanks to my fellow students Mitha Sith, Ilkka Rotola, Siimar Mee and Villy Rauhut for the invaluable interaction.

This to Peter Knoop at Odense University for helping me at several occasions, and thanks to Jens Caeen, Copenhagen University for advice and suggestions regarding a specific part of my work. Last but not least I would like to thank Jan Laeg, Gudrun University Mimeo, Aarhus, 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 Danish Science Industrat at Aarhus University the ADARES research project ¹, and the Danish Research Agency.

Henrik Esbensen

Copenhagen Institute
Aarhus University

July 1994

¹Alternative Data Representations, Algorithms and their Implementation. Danish Natural Science Research Council, grant no. 5.21.08.02.
Here was a ‘real-vald’ area offering a large number of very hard and
densely interrelated optimization problems for which solutions we
easily needed. Although I did not have any background in physics,
I perceived that this area was possible to abstract away the
physics and consider the problem from a mathematical point of view.
So in 1990 I chose rating of microcell layouts as the topic of my
thesis [Esbensen 91], 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 Confer-
ence in Glasgow Scotland and heard the presentation of K Shanks
and P Muraki’s paper ‘Fast 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 PhD 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 Muraki and
his group at Depart of Electrical Engineering and Computer Scien-
ce, University of Mligen, An Abor, NJ USA. So in the winter
of 1991/1992 an arrangement was made allowing me to visit the group
at University of Mligen for one year, starting in June 1992. Here, I
ended up working with the people who I first heard about in Scotia
two years earlier. The year in Mligen 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
and 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 PhD degree.

Henrik Esbensen

Computer Science Department
Aarhus University
July 1994
Preface

This thesis is the result of a 3-year Ph.D. study carried out in the period from 1991 to 1994. The research was done at the Computer Science Department, Aarhus University, Denmark, and supervised by Associate Professor Peter Møller-Nielsen. An integral part of the study was at the Department of Electrical Engineering and Computer Science, University of Minnesota, USA for a one-year period from June 1992 to June 1993. During this time Associate Professor Rolf Mikkelsen was acting supervisor.

The purpose of this research is to investigate the possibilities of applying evolutionary algorithms to solve various subproblems arising in layout synthesis of VLSI integrated circuits. Especially, the research has focused on the design of genetic algorithms for placement and global routing of mixed-signal layouts. The goal is to contribute to answers for questions such as: Can the concepts of genetic algorithms successfully applied within this application domain, with the objective is to generate high-quality layouts? Which performance can be obtained? Which algorithms design principles yield 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 follows, i.e. 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 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 Master's program 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 one or less artificial problems and then study how to solve them
løbplante i en vis fastadliminde og dette var inspiration til udviklingen af en placingsalgoritme kaldet SGA, der lades en
Gard SAs BÅds algoritme i den initiale fase er en ren CA, men
 undersøgelse af algoritmen indvendigt skifter der mellem et
SA- og en SaSA-algoritme, der er mere
grænse og hyggeligere. En SaSA er publiceret i [Ehren 94],
og afhænger af appendix B er en udgivet version heraf.

Samtidig nævnte er SGA's tilpasning afbilledes når global
rating høres. Gitter SaSA udviklet som en første skridt mellem
Gard SAs global rating. Den første version af algoritmen blev publiceret
som [Ehren 94]. Resultaterne var meget lover, blev algoritmen
videruddelt, selvom karakterisikken førte til højere
resultater. Der blev
afhængig af ikke længere er direkte relevante for den globale rating.
Appendix C giver attilden [Ehren 94].

Endelig beskrives i appendix D den globale rating, der
som en algoritme anses Gitter fra appendix C, ligesom den
gitter i [Ehren 94], som giving i appendix D.
er det alligevel klart, at alle de udviklede algoritmer er lukknævnte samtidig med state-of-the-art algoritmer mht. den ogal løsningskvalitet. Spøjtet gør meget lovet resultater med SGA-

algoritmen og de globale ruter. Af den anden side er SGAalgoritmen den eneste, der og kan lukknævnte plæne. De toAndre algoritmer er alle betydelig langsomme ad denne samtlige mel-

dte unødvendig. Der er imidlertid atal lønsomme til at dem, også implementering af algoritmer er astandard, her-

for det foretages at lukknævnte kan reducere væsentlig, som beskrevet i kapitel 6. Resultater er CA met meget velgodt til parallel implementa-

tion.

Ober forårsætter hovedkæde af kapitel 6, undtagelse af de vigtige givninger der sat inkludere-

form af disse. Resultater unødvendige for at samstede plac-

ering og global routing til en ganske mhp. at redusere de unødv.

lige pulser samfald af unødv.

igelte estimater.

Indtil plaserer kapitel 7 afund af hovedklausur, så de udviklede algoritmer er lukknævnte mht. løsningskvalitet inkluderes det, at GA er en lovet metode til de unødv.

ige pulser. Afund af SGAalgoritmen er lukknævnte ikke tilfredstil-

lende, men det foretages at bygdige forbedringer kan opnå disse

pulser. I løbet af metode til højere af begrænsinger mens at

være de væsentligst for at de lovet resultater. Derfor inklud-

eres at for pulser medlignende karakteristik skal man fremsige de

traditionelle linje pulseresat og de traditionelle gentile

oratorer, der disse forudser at alle løsninger at til, kan aflysde alle

begrænsinger. Istedet går de lovet resultater ved design af pulser

spidse repræsentationer og oratorer.

Artiklerne: Appendices At til D

Atillængede i appendix A, A otterer en Atil placering af mikro-

eller. Hoveden er at forårsage placeringen af samme gen-

eralisering af det teknologiske bipiding pulser til håb at en GA

beskrives i [figur 9]. Placering algoritmen blev først publiceret som

[Blasen 9] men er samme høst forbedret på radie pulser, og

atillæg i appendix beskriver denne algoritme.

I lø Bottesken forfulg af en GA og simulatet anding (SA)
laded algoritme [8, 9], en handholdt algoritme [10, 11], og to store løbsystemer, BE [Ekman 88, 12, 13] og ThrMWC [Schen 89, 14, 15], der begge indeholder global placering algoritme. Resultat er Optimalagt, som vi vides er den første tidligere C-tilet mikrocell placerings [Gan 93, 16, 17, 18]. Tilfældet er at de globale ruter indhold i ThrMWC E bliver vist i forbindelse med global ruting er Seine i et graf (SG), somme almindelige pladsalgoritme og repræsenteres en C-tilet. Dette præsenterer kapitel 4 og en state-of-the-art SG-agt. algoritme baseret på en grundlagde [J. E. Reay 89, 19, 20]. Tællevis de global ruter præsenteres i appendix C, hvori de ene global ruter som vi vides er den første tidligere C-tilet SG.

En naturlig del af almindeligens omliggende er kapitel 5, halv de udviklede algoritmer spændes og evalueres. Først præsenteres og diskriger mange af de vigtige evalueringskriterier og tekniske gør, som lagt til grund for de udviklede algoritmer. Derudover forudsættes udviklingsløsning. I tilfælde algoritmer er udviklet, for is, at nødvendige og behov for denne algoritme. Allermindst præsenterer to algoritmer til placering af mikrocell, en algoritme til SG og en global ruter. Agitmates' karakteristika er summe i tenker af de fire pladsalgoritmer introduseret i afsnit 3.4 er en vigtigste pointe at alle algoritmer udviklet betyder logiske løsninger, der repræsenterer en løsning til det ene eller anden (en algoritme samt eller representant) og eller de vigtigste præ- natter to eller tidskærer at hver løsning afsluttet og evaluere de logiske og begrundinger. Afsnit 4 kan tillade løsninger, der udeluk (må en) evaluere, og fremsæt tilfældet stræk til kostfunktionen, det er og karakteristisk til pladenaf, vil den start set ikke ud, og at det har været relativ let at finde passende vader for konsul-præsenter. I udviklede algoritmers etnitter (der direkte eller indirekte indgår i kostfunktionene) samtlige udviklede etnitter, der anvendes i algoritme beskrevet i kapitel 4. Samlingerne viser, at forudstilte etnitter generelt er de mest ujævne og dynamiske etnitter, der først tilfældet at varez.

Andere faktorer vedligeholdt retfærdige algoritmer af de udviklede algoritmers ydre vel derdiser algoritmer, fordi de fleste algoritmer er fortsatt retvisse forhold altid herf.
microele fæng af et hierarki af bokse (eller mikroceller),
af en indbyggs fæng af meldeing. En højgradigt evigt
indhold træsstør, der implikerer en givet funktionelitet, 
af boksepiformere niveau består af et atal inder, 
indbys fæng af bokse. Atomisk generering af et givet niveau hierarki det quoles tra
ditionalt i et atal diagram, der løses udelukkende af status
indbyggs alternativ. To af diagramme er placering af bokser, 
som erfølgende af rating, ds. implantation af forbindelse til en
bokse. Rating quoles igen i global og local rating har global
routing dejer sig om bestemmede løsningsproceser, actualige
router. Det er karakteristisk for alle diagramme at de er MP-feltte.
Det implikere ikke generation af inbyggs alternativ mel-
lemomgør, som følger til udbredte annullering af estimater for høne-
le kannes af løsning af erhverv udførte diagram. Målbere eller
forstørret annullering af funktionæren. Pæring og global routing er de
for diagram, der har størst betydning for en quol til at fæng
kvalitet, billet er en væsentlig led til atmetic dse to diagram er valg som
omt et emn for denne afhandling

Getiske algiter (CA) introduceres i kapitel 3. Da grund
givde er at udføre optilinger ved at sammensætte et dynamisk reelt
lidenskab til optræes. Hvis det anførerde odeladelse (sur-
vival af the fittest) kan betragtes som optilinger har kravthe er at
givde budlig tilpasning til omgivelserne. I en CA generer
en population af individer, somber seker til en løsning til et givet
optilingsproblem. Ved annullering af relevant, relative, etc.
generere ne individ fra eksisterende individer, og efter nogle genera-
tioner frahere givindivid væsent til givløsninger til optilin-
gsproblem. Kapitel 3 skildrer annulleringsænd for CA. I vis-
forsøg åbner det, at jo særlig et problemer, jo mere værd er det
for en CA løsning af for det teoretiske grundlag for CA afsæ
isk annullering af teorier skildres. I en væsentlig del af kapitel 3
er afsnit 3.4, som skildrer de ændre positioner ved design
af CA. De karakteristiske egenskaber for en god representation af en
løsning udtale af pakker af skifte af dekring af begræsninger
og værd af værd for algits evne til at kultrumatricre. Diskussion af
diske dse erer dene grundlag for disse delsioner i kapitel 5.

Vedlige vælte og algiter, der repræsenterer state-of-the-art
indfor pæring og global rating af mikrocell sats, præsenteres
i kapitel 4. Pæring-vælterne er MP, somer hævet sp'sim
VSI lagt sytse. For tægnes et billede af state-of-the-art inden for aret ved en præsentation af de bedste eksisterende algoritmer og værktøjer. Derfor gøres et respekt for udelignede algoritmer, og dets karakteristika og ydelser evaluere vilje af samtlige med state-of-the-art. Indtil gør af handlingens øges i mål at fordi at fremtidig arbejde. Med at tilfører forsketter lønke til lagt sytse samt et vist lønke til GA, har det været hensigten at afhandlingens øges i mål at det ikke sker krævede specielle forudsatninger uder en dødelig dødelig grund. Lagt sytse introduseres i kapitel 2 og GA i kapitel 3 og disse kapitler skal gøre tilstrækkelige forudsatninger for resten af afhandling. Dette kan læse en lønke til lagt sytse undlade at læse kapitel 2 og læse en lønke til GA kan undlade afstige 31, 32 og 33

Oversigtsdelen: Kapitlerne 1 til 7

Afhandlings første kapitel beskriver b.l.a. den vilje til gnsind til arealet. Forløberne betragtes som helhedsvis og omfattende problemer, mens fysiske aspekter smide udelignede lønke gøres en skiltet grænsefølger igennem. Til gnsindelser er arealet som helhedsomt, billede for vigtige forskelser, b.l.a. for evningen af de udelignede algoritmer. I stedet for at tilfører eksisteringes ydeevne samtidigere met ved at samtlige medeksisterende state-of-the-art algoritmer ydeevnighed i dta. løsningsskabelonen relativt til højere er også nødvendig for bedste eksisterende algoritmer, og ved læst der først absoulute qutil. Dette er en grundlæggende atagelse, at løsningsskabelonen prioritiserer højere end kører. I mange praktiske områder vil designeren være villig til at huse end destratid for at få løsninger bedre løsning

Kapitel 2 introduserer lagt sytse af VSI kredsløb. Der findes et antal forskellige lagt typer (eng design styles) repræsenterende forskellige grader af reglament. Til hjælp er kredsløb som f.d.s. en (U) kendte kredsløb, som den mest fuld til design type og giver mulighed for det største antal transcender pr. arealved og da bedste ydeevne af det producerede kredsløb. Kapitel 1 betjener samtidig at mikroele kredsløts er værre at gennere (mult eller atomisk) er der en lagt type, hvorfor det største be hov for forbedret værktøj rettop haves for en leg af lagt kategori. E
Danish Smør (Danke Pørre)

Af handlingens emne, formål og struktur

Afhandling er resultatet af et 3-årig PhD-studium i perioden fra 1991 til 1994 ved Teknologisk Institut Aarhus Universitet. I tilfælde af dette arbejde blev det udført under et 3-årig uddannelse ved University of Michigan, USA, og derved frigavet Associate Professor Paul Murdie som vejleder.

Af handlingens emne, formål og struktur

Afhandling fokuserer på udvikling af algoritmer til løsning af problemer, der er vedrørende at optimalisere lageret af VLSI (Very Large Scale Integration) kredsløb. Enhedsstørrelsen af VLSI kredsløb er stort ekспlosivt, og derved udløber det at fortsætte lager uden for disse problemer. Dette er derfor nødvendigt at udvikle algoritmerne for at understøtte emnerne i løbet af det aktuelle projekt. Spærrig for projektet fokuserer på design af genetiske algoritmer (GA) til løsning og optimalisering af mikroelektriske lager, der udgør en grundlag for at designe mikroelektroniske komponenter. Af handling fokuserer på udvikling af en algoritmer til løsning af problemer samt tilføjelse af genetiske algoritmer, der giver en bedre løsning. Denne omsætning af væsentlige problemer ved derved tilgængeligt styrket.

Afhandling er baseret på skitser skrevet i løbet af studiet, og den væsentligste del af afhandling udføres af fire udelog arter, givet i opgaves A D kræver indhold af afhandlingens omsigt, når det udføres af projekt ledere 1 til 7. Her introduceres emnet, dvs. A som betragtede problemer indfør...
| D31  | Area Estimation .................................................. 20 |
| D32  | Area and Wavelength Optimization .............................. 21 |
| D4   | Experimental Results ................................................ 25 |
| D41  | Test Examples ................................................................ 25 |
| D42  | Method ........................................................................... 26 |
| D43  | Image Quality .............................................................. 26 |
| D44  | Results .......................................................................... |
| D5   | Conclusion and Future Work .......................................... 219 |
| D6   | Bibliography ............................................................... |
C Opting Near-Optimal Sutions to the Sexier Problem in a Graph Using a Genetic Algorithm  165

Astrat ................................................................. 165
C1 Introduction ...................................................... 166
C2 Problem Definition ............................................ 166
C3 Description of the Algorithm ................................. 167
  C31 Overview ..................................................... 167
  C32 Graph Editing ................................................ 168
  C33 Distance Network Heuristic (DNI) ......................... 168
  C34 Genetic and Decoder ......................................... 169
  C35 Fitness Measure ............................................. 169
  C36 Genetic Operator ........................................... 170
  C37 Mutation and Inversion Operators ......................... 170
  C38 Time Complexity ........................................... 171
C4 Experiments ..................................................... 171
  C41 Test Examples ................................................ 172
  C42 Iterated Smallest Path Heuristic (SPH) ................. 173
  C43 Experimental Model ......................................... 173
  C44 Results ........................................................ 174
  C45 Comparison with Topological Algorithm .................. 175
C46 Test Results .................................................... 175
C5 Future Work ..................................................... 176
C6 Discussion ....................................................... 176
C7 Computational Results ......................................... 177
C8 Bibliography ..................................................... 178

D An Experiments Based on The Genetic Algorithm 205

Astrat ................................................................. 205
D1 Introduction ...................................................... 205
D2 Phase 1 of the latter ........................................... 206
  D21 Tetramel rats .................................................. 209
  D22 Nis with at least three terminals ......................... 209
D3 Phase 2 of the latter ........................................... 210
# A Genetic Algorithm for MicroGil Parent

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>113</td>
</tr>
<tr>
<td>A1 Introduction</td>
<td></td>
</tr>
<tr>
<td>A2 Problem Definition</td>
<td>114</td>
</tr>
<tr>
<td>A3 Description of the Algorithm</td>
<td>116</td>
</tr>
<tr>
<td>A31 Genetic Encoding</td>
<td>119</td>
</tr>
<tr>
<td>A32 Fitness Measure</td>
<td>122</td>
</tr>
<tr>
<td>A33 Genetic Operator</td>
<td>124</td>
</tr>
<tr>
<td>A34 Mutation Operators</td>
<td>125</td>
</tr>
<tr>
<td>A35 Inversion Operator</td>
<td>126</td>
</tr>
<tr>
<td>A36 Stop criterion</td>
<td>127</td>
</tr>
<tr>
<td>A4 Experimental Results</td>
<td>128</td>
</tr>
<tr>
<td>A41 Experiments with the Genoer Operator</td>
<td>128</td>
</tr>
<tr>
<td>A42 Parameter Setting</td>
<td>130</td>
</tr>
<tr>
<td>A43 Input Quality</td>
<td>131</td>
</tr>
<tr>
<td>A44 Change Rate</td>
<td>132</td>
</tr>
<tr>
<td>A45 Optimization Time</td>
<td>133</td>
</tr>
<tr>
<td>A5 Conclusion</td>
<td></td>
</tr>
<tr>
<td>A6 Bibliography</td>
<td></td>
</tr>
</tbody>
</table>

# B SGA: MicroGil Parent by a Unification of the Genetic Algorithm with Simulated Annealing

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>137</td>
</tr>
<tr>
<td>B1 Introduction</td>
<td></td>
</tr>
<tr>
<td>B2 The Genetic Algorithm</td>
<td>139</td>
</tr>
<tr>
<td>B3 The Simulated Annealing Algorithm</td>
<td>142</td>
</tr>
<tr>
<td>B4 The Unified Algorithm</td>
<td>143</td>
</tr>
<tr>
<td>B41 The Single-Track SA</td>
<td>143</td>
</tr>
<tr>
<td>B42 SAGdrilled Matrices</td>
<td>145</td>
</tr>
<tr>
<td>B43 GA and SAs Spaid Gases</td>
<td>146</td>
</tr>
<tr>
<td>B5 Application to MicroGil Parent</td>
<td>146</td>
</tr>
<tr>
<td>B51 Problem Definition</td>
<td>147</td>
</tr>
<tr>
<td>B52 Genetic Encoding</td>
<td>147</td>
</tr>
<tr>
<td>B53 Fitness Measure</td>
<td>149</td>
</tr>
<tr>
<td>B54 Genetic Operator</td>
<td>149</td>
</tr>
<tr>
<td>B55 Mutation Operator and Elitistm</td>
<td>150</td>
</tr>
<tr>
<td>B56 Inversion Operator</td>
<td>152</td>
</tr>
<tr>
<td>B6 Experimental Results</td>
<td>153</td>
</tr>
<tr>
<td>B61 Behavior in Med Mat</td>
<td>153</td>
</tr>
</tbody>
</table>
5 Summary and Solution of Delquad Algirthma

51 Summary ................................................................. 61
   51.1 Basic Assumptions ............................................... 61
   51.2 Wt and Wy - the Delquad History ......................... 63
   51.3 GA Triple and Design Parameters ......................... 68
52 Solution of the Parent Algirthma .............................. 72
   52.1 Performance .................................................. 72
   52.2 Estimation of Rating Acceptance Intercast Length 77
   52.3 Search Space Reduction ..................................... 78
53 Solution of the Seinir Te Algirthma ........................... 79
54 Solution of the Global Better ................................. 82
   54.1 Performance .................................................. 82
   54.2 Estimation of Rating Acceptance Intercast Length 84
55 General Solution and Ojectives ................................. 85

6 Future Work ............................................................. 89
   61 Optimization Criteria ........................................... 89
   62 Subtareas Parent and Global Rating ......................... 91
   63 Future Roles .................................................. 92
   63.1 Impovets of the Seinir Algirthma ......................... 93
   63.2 Bottom Gotic Algirthma .................................... 94

7 Conclusion .............................................................. 97

Bibliography .............................................................. 99
# Contents

Irish Summary (Tábhacht ar Mheáin)............... vii

Preface ........................................... xiii

Acknowledgments .................................. xv

1 Introduction .................................... 1
   1.1 Title and Purpose of the Thesis ............ 1
   1.2 General Approach ............................ 3
   1.3 Organization of the Thesis ................. 5
   1.4 Description of Papers ...................... 7

2 Layout Synthesis ............................... 9
   2.1 Introduction ................................
   2.2 Layout Styles ................................
   2.3 Building Blocks Layout Synthesis ........... 14

3 Genetic Algorithm ................................ 21
   3.1 Introduction ................................
   3.1.1 The Basic Idea of GA .................... 21
   3.1.2 The Application Area of GA ............. 24
   3.2 The Style Genetic Algorithm ............... 25
   3.3 Theory of Genetic Algorithms .............. 27
   3.4 Practical Issues of Genetic Algorithms .... 30
       3.4.1 What is a good ending? ............... 30
       3.4.2 Exploiting Problem-Specific Knowledge 34
       3.4.3 Constraint Handling ................... 35
       3.4.4 Selection of Parent Vectors ......... 39

4 Related Work .................................... 43
   4.1 Modelling Parent .......................... 43
Placement and Global Routing of VLSI Macro-Cell Layouts Using Genetic Algorithms

Ph.D. Thesis
by
Henrik Ekløff

Computer Science Department
Aarhus University
DK-8000 Aarhus C, Denmark

July 1994