4th Int. Conf. on Parallel Problem Solving from Nature (PPSN IV), Berlin, Germany, September 22-27, 1996

An Adaptive Parallel Genetic Algorithm for VLSI-Layout Optimization

Volker Schnecke and Oliver Vornberger

University of Osnabrück, Dept. of Math./Computer Science
D-49069 Osnabrück, Germany

Abstract. The generation of a high quality layout during the design of a VLSI microchip is a very complex combinatorial optimization problem. Components of a circuit have to be placed, and signal nets have to be routed on an overall minimal area. In this paper a parallel Genetic Algorithm for the combined optimization of placement and routing is presented. The main focus is on the self-adaptation of the search process: Several islands execute a sequential GA with different strategies. At fixed intervals these strategies are ranked and each strategy is adjusted to the next better one by assimilating its characteristical parameters.

1 Introduction

One strength of Genetic Algorithms (GAs) is their robustness, which mainly is caused by the fact that they deal with a sample of candidate solutions to an optimization problem at a time, and that these solutions are encoded in a problem independent representation. In most cases this *genotype* coding is a string *(chromosome)* of elementary data-types, preferably bits or floats. During the optimization process, new candidate solutions are composed using more or less standard mutation and crossover operators. When solving discrete real-world optimization problems by a GA, it is often necessary to use a complex (non string) genotype representation and operators which only generate admissible solutions. Problem-specific knowledge is often used for hillclimbing during the search.

After finding a feasible coding and implementing the operators, the 'only' work to do is to find a set of control parameters, that enable the GA to reach a global optimum and to minimize the time needed for convergence. Because these parameters depend on the representation and on the specific application, and might be changed during the evolution process, it is useful to add a self-adaptation mechanism for the values of these parameters.

In this paper, a parallel GA for a combinatorial optimization problem arising during the computer aided design of microchips is presented. The main feature of this algorithm is a tree-structured genotype representation with problem-specific operators. The paper is organized as follows: It starts with a description of the application and the chosen representation. After the outline of the genetic operators, the strategy adaptation is explained and results for real-world circuits are presented.

2 The Design of Macro-Cell Layouts

The layout generation is one of the most time consuming tasks in the design cycle for VLSI- (very large scale integrated) microchips [13, 17]. The input to this problem is a partitioned circuit, i. e. the elementary components of the circuit are grouped to build up to 50 *macro-cells* (modules). On the borders of these cells, *terminals* are located which have to be connected by *signal nets*.

The output of this design process is a layout for the circuit (fig. 1). The layout describes the *placement* of the cells and the routes for the interconnection wires between them. The main objective in layout optimization is to find an arrangement with a minimal overall area. Cells are not allowed to overlap each other, and the routing has to meet the technical constraints: space between parallel wires has to be added to prevent short circuits and for some critical nets the delay has to stay below a given threshold, which results in maximal admissible wirelengths for these nets.

Due to the complexity of the single tasks, *placement* of the cells is usually optimized separately from the computation of the *routing* for the interconnection nets. During placement an estimated amount of routing space is added, which restricts the maximal number of nets to be routed inside a *channel* between two cells. The following routing is two-phase: in the *global routing* the 'loose' (global) routes for all interconnection nets are determined, whereas during the *detailed routing* the exact ways for the wires in the channels between the modules are fixed. Due to an overestimation of the needed routing space during placement, the layout has to be *compacted* after completion. In case of fault estimation, the routing cannot be completed inside the reserved routing area. This is either realized during the computation of the global routes, or during channel routing. In the latter case, the process has to backtrack and different global routes for some nets have to be chosen. If even global routing is impossible, the modules have to be rearranged, i. e. the process has to backtrack to the placement task.

The splitting of the layout design process results from the fact that the combined optimization of both tasks is too complex to be managed by most optimization techniques. Nevertheless, because of the interdependencies between placement and routing, it is wise to combine both tasks.

3 Genetic Layout Optimization

The classical work on layout generation with GAs is done by Cohoon *et al.*[3, 4]. They encode a placement by a polish notation of a binary slicing tree, thus having a chromosome represented by a string. They use different recombination operators, which work either directly on this string, or take the tree structure into consideration by decoding the chromosome. Chan *et al.* [2] represent a placement in a bit-matrix. The layout area is divided into discrete regions and each row in the matrix describes the orientation and the position for a single cell. Recombination is done by mixing two matrices. During the optimization incorrect placements with overlapping cells are allowed and are handled by adding a

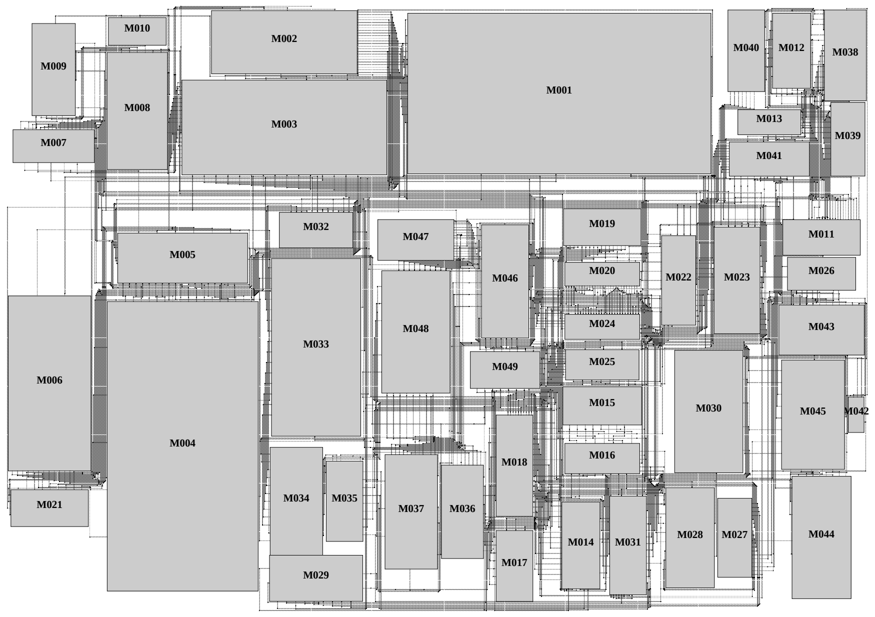

Fig. 1. A layout for the circuit *ami*49 with 49 macro-cells and 408 nets

penalty term when computing the fitness. Esbensen [6] describes a GA for macro cell placement where the genotype representation is a binary tree. In contrast to the approach of Cohoon *et al.*, this tree does not directly characterize a placement, but it can be generated by decoding the tree in a traversal due to a given order. His operators directly work on the tree structure.

The major drawback of the previously mentioned approaches is the fact, that they only optimize the placement without considering the routes to be chosen for the interconnection wirings. The approach described in this paper also uses a genotype representation as a binary tree and non-standard genetic operators, but here the global routes for the signal nets are computed before fixing the positions of the cells. Due to this, routing is not restricted by any limited capacities arising from wrong estimates for the need of routing space.

3.1 Genotype Representation

The genotype is encoded as a binary slicing tree, which defines the relative placement of the cells (fig. 2, left). It is composed in a bottom-up fashion. In each inner node two *blocks* (in the lowest level these are single cells) are joined to a *meta-block* (partial placement). In each meta-block the orientations of the combined blocks are fixed (fig. 2, right). If the blocks represent flexible macro-cells, which are cells with variable aspect ratios, different shapes are stored for the meta-block. Therefore every tree describes several possible shapes for the corre-

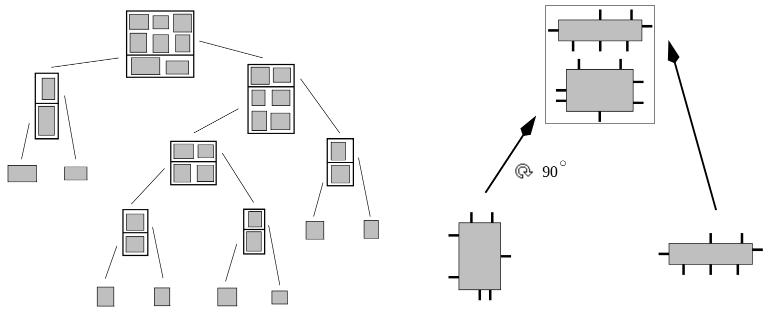

Fig. 2. The genotype (left), and the composition of a meta-block (right)

sponding layout, which enormously improves the performance of the GA. While here only results for circuits with fixed cells are presented, more information and empirical data for the flexible case can be found in a former paper [15].

After the construction of the tree, the relative positions of the blocks are given and the locations of all terminals are known. Out of the tree a routing graph (fig. 3, (II)) is constructed. The edges in this graph represent routing regions, which are parts of a channel, the vertices describe connections between two channels. For each net the shortest paths between its terminals in the routing graph are computed. Thus afterwards, the number of all nets routed through each region is known. For each channel inside a meta-block, routing space is added depending on the widest region within this channel (fig. 3, (III)).

During the construction of the initial individuals a special heuristic, the *iterated matching* introduced by Fritsch and Vornberger [10], is used to take care that highly connected cells are placed next to each other. For that purpose, out of all possible combinations those pairings of blocks are chosen which yield a maximal global quality with respect to the number of common nets. For further details on this point see [16].

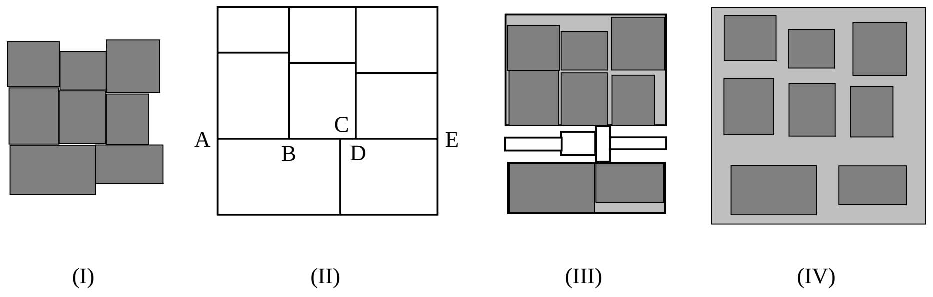

Fig. 3. A placement (I), the routing graph (II), the width for channel [A,E] is established by its widest region [C,D] (III), and the layout with added routing space (IV)

3.2 Genetic Operators

During the optimization process the placement of the blocks has to be changed and the routing has to be updated. The genetic operators directly work on the tree-structure by combining subtrees of different individuals (crossover) and modifying the tree of an individual (mutation).

The crossover operator takes two individuals out of which one offspring is composed by combining two disjunct subtrees of both parents. Unfortunately, these parts usually do not add up to a complete layout. After the combination of the two subtrees, the missing blocks have to be added to the tree to ensure that the offspring finally represents a correct layout. Here the iterated matching is used again to ensure that inside the added part of the layout highly connected cells are located near together.

A set of different mutation operators is used. The simplest operator changes the orientation of a block or a meta-block. Another operator exchanges subtrees inside the tree of the mutated individual. A distinction is made between exchanging simple blocks, a block (leaf) with a meta-block (subtree), and exchanging two meta-blocks. These cases represent the exchange of two cells, a cell with a partial layout, and the exchange of two partial layouts on the layout surface. Another mutation operator modifies the structure of the slicing tree by removing a subtree or a single block from the tree and inserting it at a different position. This corresponds to moving partial layouts or cells on the layout surface.

4 Strategy Adaptation

The performance of each GA depends on a set of control parameters like population size, crossover and mutation rates, and on the selection strategy. While there was a lot of work done in determining a globally optimal set of control parameters for a wide range of applications, in the nearer past some work on self-adapting GAs has been published. Here no static parameter set is determined, but an adaptation of the parameters during the optimization process takes place. This can be done by externally changing the mutation-rate in a predefined manner like Fogarty describes [8], or controlling the parameters of the GA by a simulated annealing like 'cooling-schedule', which is done by Esbensen and Mazumder in their GA for macro-cell placement [7]. Adaptation can be controlled by the evolution process itself by using a (meta-) GA for the optimization of the parameters like Bäck [1] or Freisleben and Härtfeler [9] have done. Wang *et al.* [18] introduce a hierarchy of GAs and suggest even to adapt the representation. The concept of *competing subpopulations* by Schlierkamp-Voosen and Mühlenbein [14] is a self adapting approach, where the sizes of subpopulations, which pursue different strategies, are adapted. Successful strategies are used by larger populations, while the size of less productive populations decreases.

The adaptation in this application also takes place on the subpopulation level, but here the population sizes are fixed, while the strategies are flexible. A strategy is characterized by a parameter set for the GA. The parameter set consists of the mutation rate, the crossover rate and the threshold for the truncation selection. Further a strategy fixes the frequency of the different mutation operators.

Due to the fact that the power of the crossover operator to create new individuals is limited to the combination of two subtrees of both parents, within this application mutation is more important than in common GAs with simple encoding. The crossover operator only combines meta-blocks, but never creates a small meta-block, because in both parents always 'real' subtrees (with more than one leaf) are chosen. Hence there are several mutation operators, and every offspring is created either by crossover or by mutation. In all generations the total number of created offspring is constant.

After a fixed interval all strategies are ranked, and the parameters of each strategy are adapted to the values of the next better strategy. The best strategy is expanded, i.e. its characteristical parameters are strengthened. For example, if the best strategy is distinguished by a low crossover rate, then this value will be decreased.

If the optimization on an island gets stuck, i.e. if no progress has been realized since the last adaptation, the strategy of this island is reset. Therefore the parameters are set back to one of the initial settings. This ensures that a strategy which has got lost in the beginning, but might be helpful at this time, is reactivated to boost the optimization.

The most important point in the implementation of an adaptation is to fix the quality criterion, regarding to which the strategies are ranked. In [14], the progress of the islands is measured based on the fitness of the best individuals observed over a period of time. Because of the small subpopulation size in this application, a fitness based quality criterion turned out to be not useful. This value is very much influenced by migrating individuals. A better criterion to describe the progress during the optimization process is the *response to selection* used in Mühlenbeins BGA [11]. This response is defined by the difference between the average fitness of a (sub-) population in two succeeding generations.

5 Results

The GA has been tested with real-world circuits. These are taken from a suite of benchmarks made available for design workshops in the early 90's, and are mostly cited in the literature as the MCNC-benchmarks[1]. It is still the most referenced benchmark suite for macro-cell layout generation, and includes partitioned circuits with 10 to 49 cells, and up to 408 nets.

In fig. 4 the effect of the strategy adaptation for a sample optimization run is visualized. The circuit is *ami*33, which consists of 33 cells and 123 nets. The

[1] The name has been historically established. These benchmarks were originally maintained by 'North Carolina's Microelectronics, Computing and Networking Center' (MCNC), but are now located at the CAD Benchmarking Laboratory, North Carolina State University.

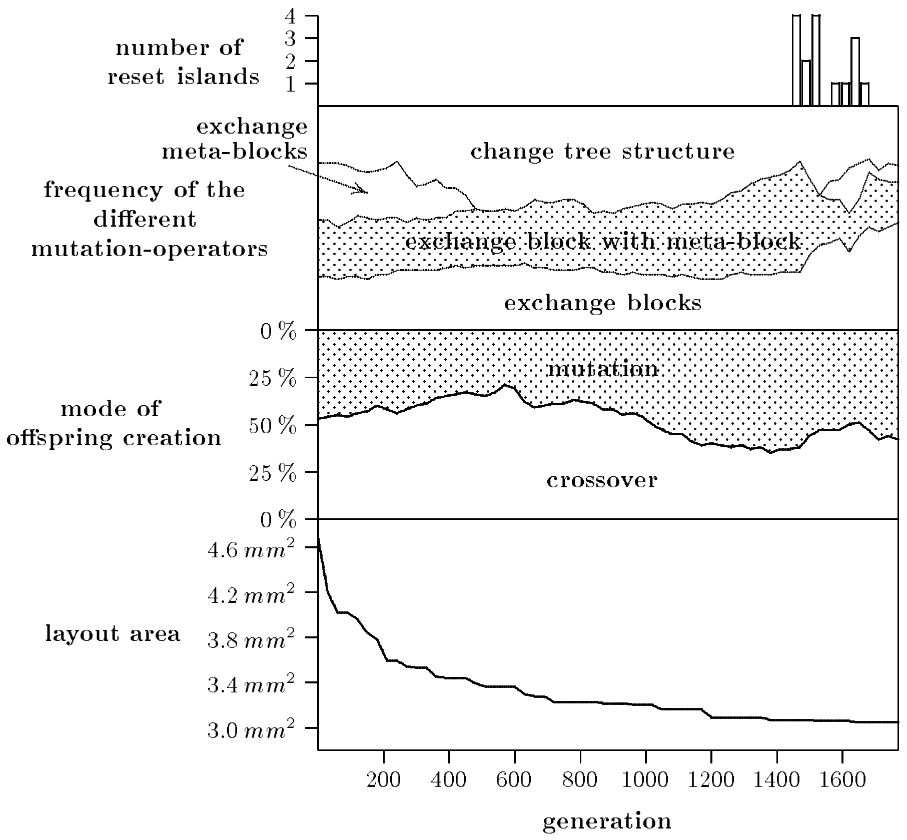

Fig. 4. The effect of the adaptation for a sample optimization run

results were achieved on a Parsytec GC/PP machine using a 12 processor network. Each island held a population of 20 individuals, the migration interval was 6 generations, and adaptation took place every 30 generations. An average of all values was taken over the 12 subpopulations except for the layout area (bottom), which describes the fitness of the best individual in the whole system. At the beginning of the optimization process, the crossover rate, i.e. the quota of offspring created using crossover rose up to 75 % around the 500th generation. After that point, with decreasing variation in the population, mutation replaced crossover as the favourite offspring creation method. At the end ($\approx$ after 1500 generations), some strategies were reset, which yielded in balancing out the crossover/mutation rate. The part of the figure which represents the frequency of the four tree-modifying mutation operators shows that the operator, which exchanges two meta-blocks might be useless, and that the changing of the tree structure seems to be more important, but this differs on single runs. Nevertheless, the shown progress of the mutation/crossover rate is typical for this circuit.

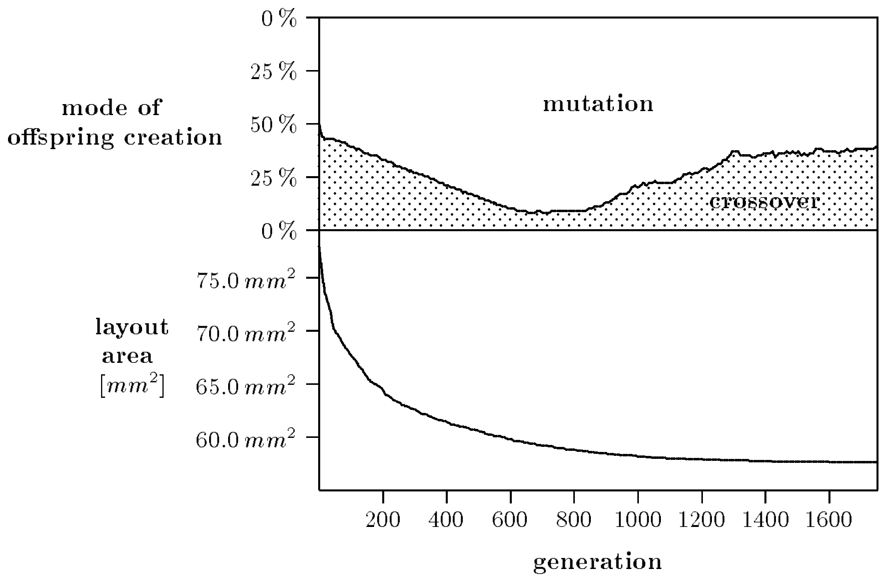

Fig. 5. The adaptation of crossover/mutation rate for circuit *ami*49 (avg. of 10 runs)

Figure 5 shows the average development of this rate for circuit *ami*49 with 49 cells and 408 nets. The runs were started with the same initial settings like those used in the other run shown in fig. 4. It is seen that for this case – in contrast to circuit *ami*33 – in the early stages of the optimization, mutation is much more efficient than crossover for enhancing the progress of the optimization. While both circuits differ in the number of cells and nets, the main difference is in the connectivity: each net in circuit *ami*33 connects an average of 3.7 terminals, whereas in circuit most nets only connect two terminals (avg. 2.3).

Figure 6 presents the effect of the adaptation on the performance, again for circuit *ami*33. The values are taken from runs on 16 processors with a subpopulation size of 10 individuals, an average of 20 runs is shown. The upper curve describes the performance resulting from experiments with the parameters set to the average values of the initial strategies of the general case. Thus all islands are pursuing the same strategy. The middle curve shows that the performance can be enhanced considerably, if the islands use different strategies. Note that in both cases no adaptation takes place. As it can be gathered from the lowest curve, the best performance is reached if the islands start with different strategies which are adapted during the search.

6 Conclusions and Future Work

The presented parallel GA computes densely packed placements for the modules of a circuit with short wirings for the signal nets. The main reasons for enabling

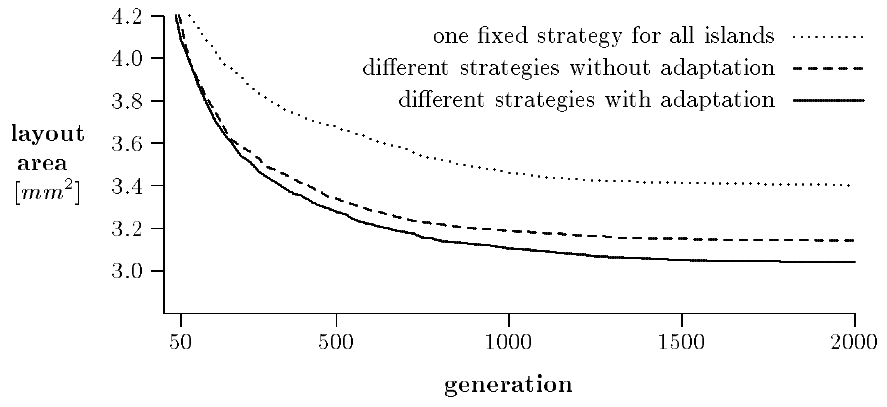

Fig. 6. The performance of the parallel GA depending on the use of different strategies and adaptation (avg. of 20 runs each)

the optimization of this real-world combinatorial problem by a GA are

- the use of a tree structured genotype encoding
- the integration of the routing in the placement process
- the application of problem-specific operators
- the strategy adaptation

Strategy adaptation returns to the GA some of its robustness, which may have been lost due to the inclusion of problem specific knowledge in the operators. It has been shown that especially the development of the crossover/mutation rate differs for the presented problems. Without adapting this parameter during the optimization, search would be less efficient.

After computing the global routes in the current implementation, the width chosen for a channel is too high in most cases, because one track is added for each net inside a routing region. This leads to layouts with an about 10 % larger area than the best results published in literature. Including a sophisticated detailed routing procedure before fixing the widths of the routing regions would avoid this drawback. Nevertheless, the main strength of GA described in this paper is the integration of the routing into the placement, which are usually separated tasks in other (genetic and non-genetic) approaches. Here the exact positions for the modules are fixed not until the computation of all global routes.

Further work in this project will be the implementation of a different recombination operator. Generating an offspring out of the trees of two parent individuals is not very efficient. On the one hand two **disjunct** subtrees have to be chosen, which decreases the number of potential pairings, and on the other hand a third part has to be added to produce a complete layout. Multi-parent crossover like it is used by Eiben *et al.* [5] or even implementing a gene-pool recombination operator like it is analysed in the work of Mühlenbein and Voigt [12] might improve the performance of the GA.

7 Acknowledgements

This work is being supported by the German government in the BMBF-project 'HYBRID–Application of Parallel Genetic Algorithms in Combinatorial Optimization'. The authors thank the *Paderborn Center for Parallel Computing (PC^2)* for the opportunity to use the parallel machines located there.

References

1. Th. Bäck, *Parallel Optimization of Evolutionary Algorithms*, PPSN III, Springer LNCS 866, 1994, 418–427
2. H. Chan, P. Mazumder, K. Shahookar, *Macro-cell and module placement by genetic adaptive search with bitmap-represented chromosome*, Integration, 12 (1991), 49–77
3. J. P. Cohoon, W. D. Paris, *Genetic Placement*, Proc. of IEEE Int. Conf. on CAD 1986, 422–425
4. J. P. Cohoon, S. U. Hegde, W. N. Martin, D. S. Richards, *Distributed Genetic Algorithms for the Floorplan Design Problem*, IEEE Trans. on CAD, Vol. 10 (4), April 1991, 483–492
5. A. E. Eiben, P.-E. Raué, Z. Ruttkay, *Genetic algorithms with multi-parent recombination*, PPSN III, Springer LNCS 866, 1994, 78–87
6. H. Esbensen, *A Genetic Algorithm for Macro Cell Placement*, Procs. of the European Design Automation Conference, 1992, 52–57
7. H. Esbensen, P. Mazumder, *SAGA: A Unification of the Genetic Algorithm with Simulated Annealing and its Application to Macro-Cell Placement*, Procs. of the 7th Int. Conf. on VLSI Design, 1994, 211–214
8. T. C. Fogarty, *Varying the Probability of Mutation in the Genetic Algorithm*, Procs. 3rd ICGA, J. D. Schaffer (ed), Morgan Kaufmann Publ., 1989, 104–109
9. B. Freisleben, M. Härtfelder, *Optimization of Genetic Algorithms by Genetic Algorithms*, Artificial Neural Nets and Genetic Algorithms, R. F. Albrecht, C. R. Reeves, N. C. Steele (eds.), Springer Verlag, 1993, 392–399
10. A. Fritsch, O. Vornberger, *Cutting Stock by Iterated Matching*, Operations Research Proceedings, U. Derigs, A. Bachem, A. Drexl (eds), Springer Verlag, 1995, 92–97
11. H. Mühlenbein, D. Schlierkamp-Voosen, *The science of breeding and its application to the breeder genetic algorithm BGA*, Evolutionary Comp., 1(4), 1994, 335–360
12. H. Mühlenbein, H.-M. Voigt, *Gene Pool Recombination in Genetic Algorithms*, Procs. of the Metaheuristics Int. Conf., I. H. Osman, J. P. Kelly (eds.), Kluwer Academic Publishers, Norwell, 1995
13. S. M. Sait, H. Youssef, *VLSI Physical Design Automation: Theory and Practice*, McGraw-Hill (1995)
14. D. Schlierkamp-Voosen, H. Mühlenbein, *Strategy Adaptation by Competing Subpopulations*, PPSN III, Springer LNCS 866, 1994, 199–208
15. V. Schnecke, O. Vornberger, *Genetic Design of VLSI-Layouts*, Procs. GALESIA '95, IEE Conference Publication No. 414, 1995, 430–435
16. V. Schnecke, O. Vornberger, *A Genetic Algorithm for VLSI Physical Design Automation*, Procs. ACEDC '96, University of Plymouth, UK, 1996, 53–58
17. N. Sherwani, *Algorithms for VLSI Physical Design Automation*, Kluwer Academic Publishers, 1993
18. G. Wang, E. D. Goodman, W. F. Punch, *Simultaneous Multi-Level Evolution*, Technical Report 96-03-01, MSU GARAGe, 1996