A Genetic Algorithm for Macro Cell Placement

Henrik Esbensen*

Computer Science Department
Aarhus University
DK-8000 Aarhus C, Denmark
e-mail: hesbensen@daimi.aau.dk

Abstract

A new genetic algorithm for the macro cell placement problem is presented. The algorithm is based on a generalization of the two-dimensional bin packing problem. The genetic encoding of a macro cell placement and the corresponding genetic operators are described. The algorithm has been tested on MCNC benchmarks, and the quality of the produced placements are comparable to the best published results.

1 Introduction

Many types of algorithms for the placement of cells in VLSI layouts have been developed [11]. At the current state of the art, simulated annealing (SA) is one of the most popular. SA algorithms produces high quality placements at the cost of extensive runtimes.

A less prevalent type of placement algorithm is the genetic algorithm (GA). In [2, 10] GA's for standard cell placement are developed. The performance of these algorithms is comparable to SA algorithms. High quality placements are obtained at the cost of extensive runtimes.

To our knowledge, only one paper has been published, in which a GA for macro cell placement is presented [1]. This algorithm is based on a two-dimensional bitmap representation of the macro cell placement problem.

A GA for the two-dimensional bin packing problem has been developed by Kröger et al [6]. The two-dimensional bin packing problem can be seen as the unrealistic special case of the macro cell placement problem in which no nets exists, i.e. no routing will be performed.

In this paper a GA for the macro cell placement problem is developed based on comprehensive extensions of the genetic encoding and corresponding operators found in [6]. This approach is fundamentally different from the one found in [1]. The resulting algorithm is capable of producing placements having a quality comparable to the best published results.

*Currently at Department of Electrical Engineering and Computer Science, University of Michigan, Ann Arbor, MI 48109, USA.

2 Problem definition

In the literature the definition of the macro cell placement problem varies slightly. The following definition is used in this paper. As input we have

i) a set of rectangular *cells*, each with a number of *terminals* at fixed positions along the edges of the cell

ii) a netlist specifying the interconnections of all terminals

iii) an approximate horizontal length W of the cell to be constructed.

The macro cell placement problem is to find

i) absolute coordinates for the lower left corner of each cell

ii) the orientation and reflection in the horizontal and vertical axis of each cell

iii) a rectangle R having approximate horizontal length W enclosing all cells

The above quantities are to be determined so that the area of R is minimized subject to the following constraints:

i) no pair of cells overlap each other

ii) the space inside R not occupied by cells is sufficiently large to contain all routing necessary to realize all specified interconnections

To meet the second constraint, the area needed for routing is estimated during the placement. The estimate assumes that two layers of metal are used for routing, the area occupied by cells and the area used for routing are disjoint and all nets will be treated as signal nets.

An example macro cell placement corresponding to the problem definition given can be seen in fig. 5.

The macro cell placement problem is NP-hard [7]. Furthermore, the size of the solution space is enormous. Simply placing n cells in a single row and determining the orientation and reflection of each cell gives $O(n!8^n)$ as a weak lower bound on the size of the solution space.

3 Description of the algorithm

GA's are based on the idea of optimizing by simulating biological evolution [4]. In nature, the individuals of

0-8186-2780-8/92 $3.00 © 1992 IEEE

a population adapts to the environment in which they live. The fittest individuals have the highest probability of survival and tend to increase in numbers, since their reproduction rate is high and their characteristics are inherited by their descendants. On the other hand, the less fit individuals tends to die out. This principle is known as "survival of the fittest", and can be used in optimization. Given some optimization problem, e.g. the macro cell placement problem, define an *individual* to be a solution and define a measure of *fitness* of an individual. Then generate a *population*, and simulate the process of evolution. The most important components of this process are *reproduction* and *mutation*, for which application specific operators has to be designed. If the simulation works, highly fit individuals will emerge after a number of generations. These correspond to good solutions of the given optimization problem.

```
generate(P_C);
evaluate(P_C);
repeat noOfGenerations times:
  P_N := ∅;
  repeat noOfOffspring times:
    select p_1 ∈ P_C, p_2 ∈ P_C;
    P_N := P_N ∪ crossover(p_1, p_2);
  end;
  evaluate(P_C ∪ P_N);
  P_C := select(P_C ∪ P_N);
  ∀ p ∈ P_C : possibly mutate(p);
  evaluate(P_C);
end;
q := selectBest;
optimize(q);
```

Figure 1: *Outline of the algorithm.*

An overview of the GA presented in this paper is shown in figure 1. Initially, a population P_C is constructed from randomly generated individuals. The fitness of each individual is computed by evaluate(P_C), described in section 3.2. The quality of any individual is relative to the rest of the population. Hence, computation of fitness requires the complete population as input. One cycle of the repeat loop corresponds to the simulation of one generation. Throughout all generations, the number of individuals $|P_C|$ is kept constant.

Reproduction initiates each generation. Mating is simulated by the crossover operator described in section 3.4. Given a pair of individuals, one offspring is produced. The overall purpose of crossover is to assure convergence of the process. Hence the offspring produced have to resemble their parents. By repeated selection and mating of individuals from P_C, a set of offsprings P_N of size noOfOffspring is generated. The strategy for selection is described in section 3.3. By replacing some individuals in P_C by individuals from P_N a new current generation P_C appears. This selection also described in section 3.3 depends on the fitness of all existing individuals. Hence, the whole population $P_C \cup P_N$ is taken as argument in the evaluation. The parameter noOfOffspring determines the potential rate of replacement in the population, that is the potential speed of the evolution. If the value is too high, the amount of information accumulated in the population are not exploited properly, and the process becomes unstable.

With a small probability, any individual in P_C is subject to mutation, i.e. one or more random changes. The purpose of mutation is too avoid getting stuck in local minima and to assure the exploration of new regions of the search space. In section 3.5 five 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. Too low mutation probabilities may cause convergence into local minima only.

Each generation is completed by evaluation of all individuals. At the end of the simulation, the best individual q, which has ever existed, is selected. Finally, optimize(q) tries to improve q a little more by performing a sequence of mutations, each of which improves q. An exhaustive strategy is used, so that when optimize(q) has been applied, no mutation exists, which can improve q further. q then constitutes the resulting placement.

At any point in time, each individual satisfies all constraints. This is a fundamental design decision which is in contrast to the strategy used in [1]. Our choice makes it easier to define a suitable fitness measure. On the other hand, it also complicates the tasks of the genetic operators considerably and tends to increase their time consumption. To counterbalance this effect, a genetic encoding has been developed, in which some of the constraints are implicitly represented. Such constraints need not be considered by the genetic operators.

3.1 Genetic encoding

In GA's a distinction is made between the *genotype* and the *phenotype* of an individual [4]. A genotype is a coding 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 has to be expressed in terms of a phenotype. A *decoder* is used to compute the phenotype corresponding to a given genotype. Estimation of the routing area needed is performed during decoding.

3.1.1 Genotype and decoder

The genotype of an individual is a modified and extended version of the genotype developed in [6]. Suppose an individual has n cells $c_1, ..., c_n$. Let (c_i^{xl}, c_i^{yl}) denote the lower left and (c_i^{xu}, c_i^{yu}) the upper right corner of cell c_i in the phenotype. At genotype level, the placement of cells is represented as follows. A binary tree (V, E), $V = \{c_1, ..., c_n\}$, in which the i'th node corresponds to cell i, represents (c_i^{xl}, c_i^{yl}) for all cells. Two kinds of edges exists: top-edges and right-edges, so that

$E = E_t \cup E_r$, $E_t \cap E_r = \emptyset$. All edges 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. Let e_{ij} denote an edge from c_i to c_j. $e_{ij} \in E_t$ (E_r) means that cell c_j is placed above (to the right of) c_i in the phenotype. That is,

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

The tree is decoded as follows: The cells are placed one at a time in a rectangular area having horizontal length W and infinite vertical length. Each cell is moved as far down and then as far left as possible without violating the routing area estimate described in subsection 3.1.2. This is called the *BL-strategy* (bottom left) and the resulting placement a *BL-placement*. The cells are placed in ascending order according to their *priority* defined by the one-to-one mapping $p: V \rightarrow \{1, ..., n\}$. Any node has higher priority than its predecessor in the tree. The orientation of any cell is defined by the function $o: V \rightarrow \{\text{turned, not turned}\}$. Reflection in the x-axis is defined by $r_x: V \rightarrow \{\text{reflected, not reflected}\}$. Similarly, reflection in the y-axis is defined by $r_y: V \rightarrow \{\text{reflected, not reflected}\}$.

When all cells are placed, the decoder computes the rectangle R. From R_{se}, the smallest rectangle enclosing all cells, R is constructed by extending R_{se} until the routing area estimate is satisfied along all sides of R.

An example tree with 8 cells is shown in figure 2 together with the corresponding phenotype. Note that for a given cell placement several possible trees may exist.

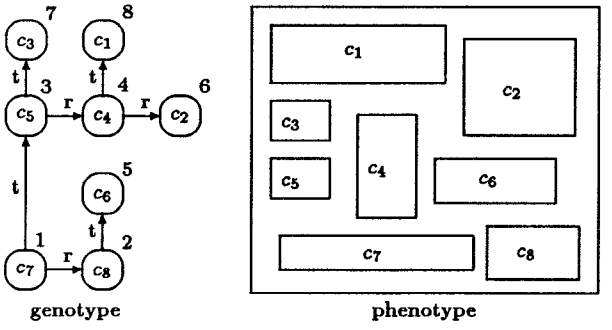

Figure 2: *Cell-part of phenotype and genotype. The number at each node denotes its priority.*

3.1.2 Estimation of routing area

When decoding the binary tree, estimation of the routing area needed is performed as each cell is placed. When placing the i'th cell, the distance needed in each direction $s \in S = \{\text{north,east,south,west}\}$ to previously placed cells is computed by a function D, which depends on all previously placed cells. Each cell is placed according to the BL-strategy and as close to the previously placed cells as allowed by D.

Figure 3 illustrates how D is computed. When testing if cell c_i can be placed at some given position (c_i^{xl}, c_i^{yl}), the four areas indicated by dashed squares are considered.

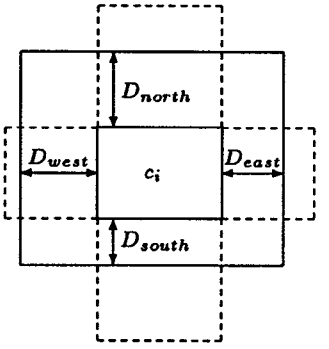

Figure 3: *Estimation of routing area.*

D_s depends on all terminals at side s of c_i and of all terminals in previously placed cells, which are inside the square at side s and which are placed at some side parallel to side s of c_i. Given this set of terminals, the channel density d_s is computed as if the square were the routing channel. In order to account for global routing, D_s is then computed as

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

where h_s is the length of side s of c_i, λ is the spacing in the routing grid, $round(x)$ is the rounded value of x and a and b are user defined parameters. The area inside the solid rectangle shown in fig. 3 is uniquely determined by D. Cell c_i can be placed at the given position if and only if this area contains no (parts of) cells apart from c_i itself. When $a = b = 0$, the estimated routing area is a lower limit of the area needed by any router regardless of the channel definition. If $a > 0$, the corresponding term of D_s encreases with h_s. The argument for this definition is that the longer the channel, the more likely nets will pass through it [12]. Note that moving c_i in any direction may affect the value of D for all four values of s.

In summary, given V, the genotype of an individual consist of the relations E and the functions o, p, r_x and r_y. The genotype has the nice and important property of implicitly representing most constraints of the problem. These constraints are therefore automatically satisfied, and as a consequence, the genetic operators need not consider these constraints. However, the constraint of approximate horizontal length W is not implicitly represented. Hence this constraint has to be considered by the genetic operators.

3.2 Definition of fitness

Fitness values are always computed for a population of individuals at a time. Let Π be the set of all possible individuals, and let $F: \Pi \rightarrow \Re_+$ be the fitness measure.

Suppose the population $P_C \subset \Pi$ is evaluated. Since the objective is to minimize layout area, initially F is defined as

$$F(p) = \frac{1}{A(p) - C}$$

where $A(p)$ is the area of the rectangle R of $p \in P_C$ and C is the sum of the area of all cells. Hence, all individuals having equal area will have equal fitness. When fixing the total area of a placement, the probability of a 100% routing completion is likely to increase as the total wire length decreases. Therefore, the fitness of individuals having equal areas are adjusted, so that the fitness increases as the estimated wire length decreases.

The total wire length of an individual is estimated as in [5]: Let M denote the number of nets, and let m_k denote the total number of terminals of the k'th net. $t_{ki} = (x_{ki}, y_{ki})$ is the position of terminal i in net k. The *center of gravity* of the k'th net is then defined by

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

and the estimated total wire length L is defined as

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

where $p \in P_C$ and $\|x\|$ denotes the usual Euclidian vector norm.

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

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

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

$$F'(p_k) = F(p_i) + (k - i)\delta F_{ij} \ , \ k = i + 1, \ldots, j$$

Since the fitness values now defined can be very small, they are normalized. Finally, the values are scaled linearly as described in [4], in order to control the variance of the fitness in the population.

3.3 Selection strategies

Selection takes place in two different contexts: Selection of mates for crossover, and selection of individuals for survival into the next generation. The selection strategies should reflect the principle of survival of the fittest. Since they have a major effect on the standard deviation in the population, they are crucial to the performance of the algorithm.

Using the terminology of [4], the selection strategy for crossover is stochastic sampling with replacement. That is, the individual $p_i \in P_C$ is selected with probability

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

The two mates needed for one crossover are selected independently of each other. Any individual may be selected any number of times in the same generation.

While the selection for crossover is stochastic, the selection for survival is completely deterministic. The next generation is simply defined to be the $|P_C|$ fittest individuals in $P_C \cup P_N$ (see figure 1).

3.4 Crossover operator

Given two individuals α and β, the crossover operator generates a new feasible individual γ, the descendant of α and β. The crossover operator is an extended version of the operator presented in [6]. Throughout this section, a superscript specifies which individual the marked property is a part of.

E^γ is constructed as follows. From the cell tree of α a connected subset $T_s = (V_s, E_s)$, $V_s \subset V$, $E_s \subset E^\alpha$ is chosen. T_s is chosen at random but subject to two constraints: First of all it is required that decoding T_s in the order defined by p^α, i.e. using $\min\{c \in V_s \mid p^\alpha(c)\}$ as root, causes no constraint violations. Secondly, E_s has to satisfy $E_{min} \leq |E_s| \leq E_{max}$, where E_{min} and E_{max} are user defined parameters. Initially E^γ is defined to be E_s. Hence, γ has inherited all cells in V_s from α. The remaining cells $V - V_s$ are then inherited from β by extension of E^γ. The cell tree of β is traversed in ascending order according to p^β. At any node it is checked if the corresponding cell c belongs to V_s, that is, has been placed in γ already. If so, the cell is skipped. Otherwise, c is added to the cell tree of γ by extending E^γ. c is added at the free position, which corresponds to the lowest possible position at phenotype level. This position is found by exhaustive search using the decoder. The orientation and reflection of any cell is inherited unaltered together with the cell itself. That is,

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

r_x^γ and r_y^γ are defined exactly the same way. p^γ should correspond to the order in which the cells were placed when creating E^γ. Since p^γ is one-to-one, the following constraints uniquely determines p^γ:

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

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

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

The construction of γ is illustrated in figure 4.

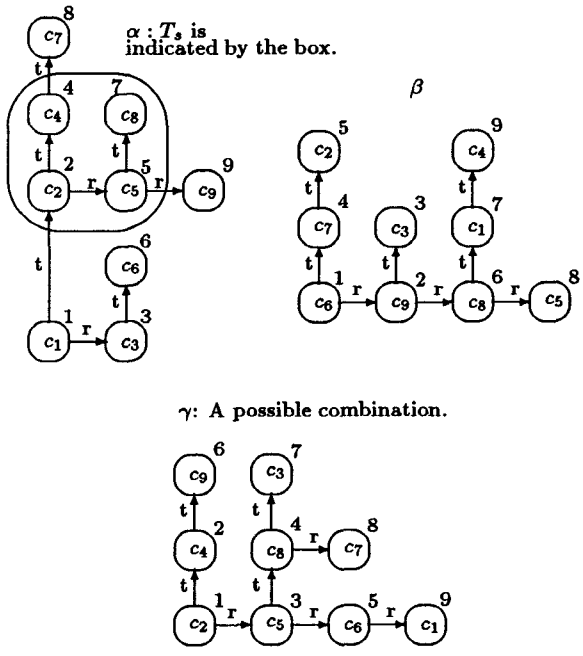

Figure 4: *Combining α and β.*

3.5 Mutation operators

Five different types of mutation exists for the random modification of a genotype. Each of these mutations involves the random selection of one or two cells, in the following denoted by c_i and c_j, $i \neq j$. The mutations are:

i) Alter the set of edges E by moving a leaf c_i to another free position chosen at random. The type of the edge going into the leaf may be changed as part of the move.

ii) Alter the set of edges E by exchanging c_i and c_j. The priorities of the cells are exchanged simultaneously, so that no pair of cells are prevented from being exchanged by the priority function p.

iii) Alter p by exchanging $p(c_i)$ and $p(c_j)$.

iv) Turn a cell. I.e. alter the value of $o(c_i)$.

v) Change the reflection of a cell. I.e. choose at random one of the functions r_x or r_y. Then alter the value of $r_x(c_i)$ or $r_y(c_i)$ corresponding to the function chosen.

When performing any of these mutations, a part of the genotype has to be decoded to check if the mutated individual satisfies all constraints. Mutations i,iv and v requires that all cells having priority $p(c_i)$ or higher are decoded, while mutations ii and iii requires decoding from priority $\min(p(c_i), p(c_j))$. If the mutation is not feasible, the mutation operator repeatedly tries out new random changes of the same type, until either a successful mutation is performed, or it has been detected that no feasible mutation of that type exists.

4 Experimental results

The algorithm has been implemented in the C programming language on a Sun Sparc ELC workstation. Approximate size of the source code is 10.000 lines. The performance of the algorithm has been tested on three benchmarks from the 1988 MCNC International Workshop on Placement and Routing. Table 1 lists the main characteristics of these examples.

Benchmark	Cells	Nets	Terminals
apte	9	97	287
xerox	10	203	698
hp	11	83	309

Table 1: *Benchmark characteristics*

The router used is part of the Magic layout system [9]. As opposed to others, this router is not capable of adjusting the placement during routing in order to obtain 100% routing completion and/or to reduce the total area of the layout. Therefore, the layouts are produced as follows. First, the router is applied to a placement produced by the algorithm presented. It may happen that the wiring is not fully completed even though sufficient space is present. In that case the wiring is completed manually. Afterwards a one-dimensional compaction facility, which is also part of the Magic system, is applied. Note that during this postprocessing phase, the placement is only altered by the compaction facility. The placement is never altered manually.

4.1 Layout quality

In table 2 the layout quality obtained by our algorithm is compared with the best published results known to us. The results referenced should be taken with some caution due to minor variations in the problem definitions used.

Benchmark	System	Area (mm^2)
apte	BB [8]	54.05
	This work	54.93
xerox	MOSAICO[1]	29.01
	BEAR [3]	28.47
	Seattle Silicon [12]	25.79
	BB [8]	26.17
	VITAL[1]	31.71
	This work	29.33
hp	BB [8]	12.15
	This work	12.89

Table 2: *Comparison of quality with other systems*

Nevertheless, it can be concluded that the layout quality obtained by our algorithm is comparable to the best

[1] Referenced here as found in [3, 12].

published results. Furthermore it is very likely that using another routing facility would improve our results further, since the postprocessing phase described above is far from ideal. Fig. 5 shows the result obtained for the xerox benchmark.

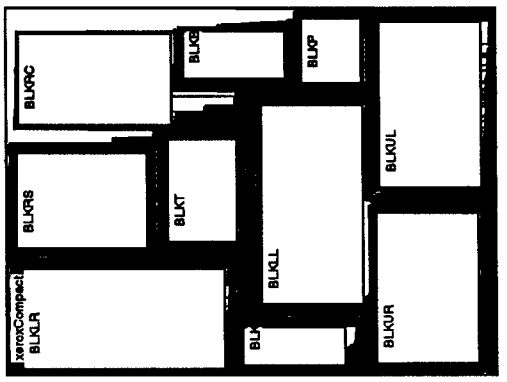

Figure 5: *Placement result for the xerox benchmark*

4.2 Parameter settings and runtime

For each benchmark, the result is produced by a single execution of the algorithm using an arbitrary initialization of the random number generator. The values of the most important parameters is as follows: $|P_C| = noOfOffSpring = 25$, noOfGenerations = 200. Probability for each of the five types of mutation: 0.006. The same parameter settings are used for all benchmarks.

The absolute runtime of the current implementation for the apte, hp and xerox benchmarks is 66, 126 and 1096 minutes, respectively. This is not acceptable nor competitive. Furthermore, this runtime problem currently prevents experiments with larger benchmarks like ami33 and ami49. Thus, time consumption needs to be reduced by an order of magnitude. There are two reasons why we believe that this reduction can be obtained: First of all, in the current implementation almost all the runtime is spent measuring channel densities in a very ineffective way. These measurements can be computed much faster by using another strategy. Secondly, the runtime can be reduced significantly by implementing a parallel version of the algorithm. Due to the inherent parallelism in any GA, a high speedup on any MIMD architecture can be expected [4, 6].

5 Conclusions

In this paper a genetic algorithm for the macro cell placement problem has been presented. The layout quality obtained by the algorithm is comparable to the best published results. Since this work is a novel approach to macro cell placement, further improvements are most likely. E.g. by trying different crossover operators, trying alternative estimation of routing area, etc. The current runtime is too extensive, but also here improvements can be made. In conclusion, genetic algorithms is a promising approach to the macro cell placement problem.

Acknowledgement

The author would like to thank Mr. Berthold Kröger, University of Osnabrück, Germany, for valuable discussions concerning this work.

References

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

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

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

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

[5] A. Herrigel, W. Fichtner, *An Analytic Optimization Technique For Placement Of Macro-Cells,* Proceedings of The 26th ACM/IEEE Design Automation Conference, pp. 376-381, 1989.

[6] Berthold Kröger, Peter Schwenderling, Oliver Vornberger, *Genetic Packing of Rectangles on Transputers,* Transputing '91, Vol. 2, IOS Press, 1991.

[7] F.T. Leighton, *Complexity issues in VLSI,* MIT Press, 1983.

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

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

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

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

[12] M. Upton, K. Samii, S. Sugiyama, *Integrated Placement for Mixed Macro Cell and Standard Cell Designs,* Proceedings of The 27th ACM/IEEE Design Automation Conference, pp. 32-35, 1990.