General Floorplanning with L-shaped, T-shaped and Soft Blocks Based on Bounded Slicing Grid Structure

Maggie Kang
maggiek@cse.ucsc.edu

Wayne W.-M. Dai
dai@cse.ucsc.edu

Computer Engineering, University Of California,
Santa Cruz, CA, 95064, U.S.A.

Abstract— A new method of non-slicing floorplanning is proposed, which is based on the new representation for non-slicing floorplans proposed by [1], called bounded slicing grid(BSG) structure. We developed a new greedy algorithm based on the BSG structure, running in linear time, to select the alternative shape for each soft block so as to minimize the overall area for general floorplan, including non-slicing structures. We propose a new stochastic optimization method, named genetic simulated annealing (GSA) [3] for general floorplanning. Based on BSG structure, we extend SA-based local search and GA-based global crossover to L-shaped, T-shaped blocks and obtain high density packing of rectilinear blocks.

I. Introduction

A floorplan is a dissection of a rectangle into a set of non-intersecting rectangles, called rooms. A floorplan specifies the topological relationship among rooms. Each room has a set of alternative shapes or realizations of the room. A realization of a floorplan is obtained by selecting a realization for each block. There are two kinds of floorplans: slicing and non-slicing. A *slicing floorplan* is a floorplan which can be obtained by recursively cutting a rectangle into two parts by either a vertical or a horizontal line [8], [5], otherwise is *non-slicing floorplan*.

For building block layout with multi-layer technology, most of channel routing will be replaced by area routing and blocks are packed together to minimize the area. This technology shift makes non-slicing floorplan becomes more and more important.

Ref. [5] proposed a normalized Polish expression to represent the slicing floorplan solution space, which enables efficient neighborhood search. Ref. [1] introduced the bounded slicing grid(BSG) structure and [4] proposed the Sequenced Pair (SP) to represent non-slicing floorplans. Both BSG and SP provide a finite solution space at least one of which is optimal if the solution space is large enough, and the non-slicing floorplan solution can be evaluated efficiently during the stochastic optimization process.

To solve the non-slicing floorplanning problem, we propose a new stochastic optimization method, named genetic simulated annealing (GSA) which combines the local stochastic hill climbing features from simulated annealing (SA) and the global crossover operation from genetic algorithm (GA). We extend both SA-based local search and GA-based crossover to L-shaped and T-shaped blocks. The experiment results demonstrate our algorithm can obtain efficient packing for rectilinear blocks very quickly.

Due to the physical layout purpose, most blocks in floorplanning have a set of alternative realizations. We develop an efficient greedy algorithm, running in linear time, to minimize the total area by selecting a realization for each block.

Following Section 2 introduces the basic BSG structure and non-slicing floorplan realization by BSG sizing. Section 3 presents the new greedy algorithm for general floorplanning with soft blocks. The genetic simulated annealing algorithm and the placement of L-shaped and T-shaped blocks will be described in Section 4. Section 5 gives some concluding remarks.

II. Bounded Slicing Grid Structure(BSG)

The bounded slicing grid structure (BSG) can be obtained as follows [2]: make a row of non-overlapping horizontal line segments of two unit length and repeat them row by row, shifting by one unit length between the adjacent rows. A set of columns of vertical line segments with two unit length can be constructed in a similar way. Those line segments are called *Bounded Slice Lines*, or BS-lines. None of the BS-lines are intersecting each other (See Fig.1). The rectangle region enclosed by four BS-lines is called a *room*. With BSG model, a floorplanning is represented by an assignment of blocks to rooms and this assignment is referred to as a *BSG-seed*. An *empty room* contains no block. Otherwise, the room is called *occupied room*. Given a BSG-seed, a floorplanning can be realized by stretching or shrinking, collectively called *sizing*, the BS-lines. The sizing operation shrinks the empty rooms to zero area and determines the shapes, thus the coordinates of the blocks.

The sizing operation is based on two directed acyclic graphs: the *horizontal adjacency graph* G_h and the *vertical adjacency graph* G_v (See Fig.2). Given a BSG structure, a vertex $v \in G_h$ represents a vertical BS-line. There

ASP-DAC '97
0-89791-851-7$5.00

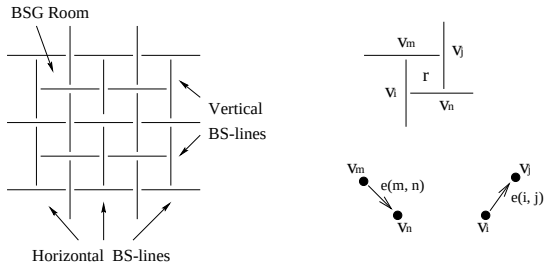

Fig. 1. Bounded Slicing Grid Structure.

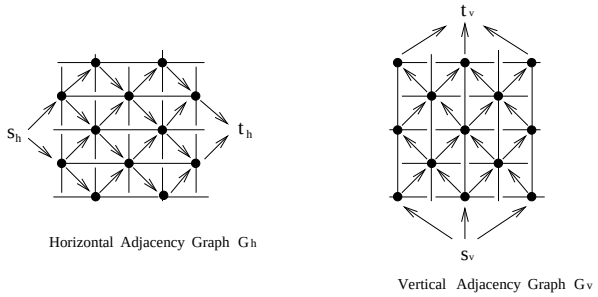

Fig. 2. The adjacency graphs of the BSG structure of Fig. 2

is a directed edge or an arc from vertex v_i to vertex v_j if the BS-line corresponding to vertex v_i is on the left of the BS-line corresponding to vertex v_j and they share the same room (See Fig.1). In particular, there is an arc from the source vertex $s_h \in G_h$ to each of the vertices representing the leftmost BS-lines. Similarly, there is an arc from each of the vertices representing the rightmost BS-lines to the target t_h (See Fig. 2). G_v can be defined similarly. Corresponding to each room r in a BSG structure, there is an unique arc e_r^h in G_h and an unique arc e_r^v in G_v. The weights of e_r^h and e_r^v are given by the width and height of the block assigned to the room r respectively, if the room is occupied. Otherwise, the weights are zero. For each vertex $v \in G_h$, $l(s_h, v)$ denotes the length of the longest path from the source s_h to v. $l(s_h, v)$ gives the x-coordinate of the vertical BS-line represented by v, or the x-coordinate of one side of the rooms bounded by the BS-line. In particular, the length of the longest path from the source s_h to the target t_h, $l(s_h, t_h)$, gives the width of the overall layout. The y-coordinates of the blocks and the height of the overall layout can be determined in a similar way by the lengthes of longest pathes in G_v. Since the adjacency graphs G_h and G_v are directed acyclic graphs, the longest paths can be found in linear time.

III. Area Minimization for General Floorplans

The area minimization problem of a floorplan can be defined as follows: given a BSG structure and a BSG-seed, given the realizations of each block, determines the realization of the floorplan which has the minimal area.

We develop a new greedy algorithm which alternatively reduces the overall width and height monotonically until no such reduction is possible. Since the height reduction and width reduction process are similar, in the following, we describe only the height reduction.

Recall that each room r in BSG structure corresponds to one arc e_r^h in G_h and one arc e_r^v in G_v. The weights $w(e_r^h)$ and $w(e_r^v)$ represent the width and the height of the room r respectively.

For any arc $e(i,j) \in G_h$, $w(i,j)$ denotes weight of $e(i,j)$, that is the width of the room corresponding to $e(i,j)$. We define the slack of the arc $e(i,j)$ as the maximum amount of increase in the width of the corresponding room such that no increase in the overall width. Let $l(s_h, v_i)$ and $l(v_j, t_h)$ denote the length of the longest path from the source s_h to v_i and that of the longest path from v_j to the target t_h respectively. The slack of $e(i,j)$ is given by:

$$sl(i,j) = l(s_h, t_h) - l(s_h, v_i) - l(v_j, t_h) - w(i,j) \qquad (1)$$

We define the slack of a path p as the minimum value of $sl(i,j)$ for $e(i,j) \in p$, that is:

$$sl(p) = \min_{e(i,j) \in p} sl(i,j) \qquad (2)$$

At each iteration of the height reduction, we choose one path p_h from s_h to t_h in G_h and change the heights of the corresponding blocks, such that no increase in the overall width. As the result of the height reduction, the widths of the blocks may increase. The slack of the path p_h amounts to the maximal accumulated increase in widths of those blocks without increase in the overall width.

Notice that for any pair of paths p_h from s_h to t_h in G_h and p_v from s_v to t_v in G_v, there exist exact one arc e_h on p_h and exact one arc e_v on p_v such that e_h and e_v correspond to the same room. We call a block *h-critical* if the corresponding block is on a longest path in G_h. Similarly, *v-critical* if the arc is on a longest path in G_v. Corresponding to a path p_h, if we reduce the heights of all v-critical blocks by Δh and reduce the heights of other blocks accordingly, the overall height will be reduced by exactly Δh.

Next we derive a sufficient condition for the height reduction not to increase the overall width. Let $\Delta w(i,j)$ denote the increase in weight of $e(i,j)$. Obviously the overall width will not be increased if the following condition holds:

$$\sum_{e(i,j) \in p_h} \Delta w(i,j) \le sl(p_h) \qquad (3)$$

The *maximal slack* $sl'(i,j)$ for arc $e(i,j) \in G_h$ can be obtained by increasing the height of the corresponding

block until it becomes critical, that is the slack of the corresponding arc in G_v is zero. Due to the property of BSG structure, for any two arcs on path p_h in G_h, the corresponding arcs in G_v are not on the same path. So corresponding to the arcs on a path p_h in G_h, setting the slacks of those arcs in G_v to zero will not increase the length of the longest path in G_v.

Let $num(p)$ denote the number of arcs in path p. We define a *bottleneck path* p from s_h to t_h in G_h as a path that maximize $\frac{sl^{'}(p)}{num(p)}$, where $sl^{'}(p)$ is computed using *maximal slack* of arcs on p. We define the slack of vertex v_j as $sl^{'}(p)$, where p is a bottleneck path from s_h to v_j. Similarly, $num(v_j)$ is the number of arcs on path p. To find a bottleneck path, we initialize $sl(s_h)$ and $num(s_h)$ as follows:

$$sl(s_h) = \infty \quad (4)$$

$$num(s_h) = 0 \quad (5)$$

At each step, to update the slack of vertex v_j, we first find a particular vertex v_i^* among all predecessors v_i in G_h, such that $\frac{\min(sl(v_i^*), sl^{'}(i,j))}{num(v_i^*)+1}$ is maximized. Then we update $sl(v_j)$ and $num(v_j)$ as follows:

$$sl(v_j) = \min(sl(v_i^*), sl^{'}(i,j)) \quad (6)$$

$$num(v_j) = num(v_i^*) + 1 \quad (7)$$

Since G_h is an acyclic directed graph, an appropriate visiting order is topological: we order the vertices reachable from s_h so that if $e(i,j)$ is an arc, vertex v_i appears before vertex v_j in the order. Such topological order can be found in O($\mid V \mid$) time. Thus we can find the bottleneck path p in G_h by linear time.

In the following we discuss the detail how to adjust the heights of the blocks corresponding to the bottleneck path $p \in G_h$. Let $\Delta w(e_r^h)$ denote the width increment of arc e_r^h, $\Delta w(e_r^v)$ denote the height reduction of arc e_r^v, where e_r^h and e_r^v correspond to the same block. Assume the block has been sized in such way that $sl(e_r^v)$ is zero and $sl(e_r^h)$ is maximized.

$$\frac{w(e_r^v)}{(w(e_r^v) - \Delta w(e_r^v))} = \frac{(w(e_r^h) + \Delta w(e_r^h))}{w(e_r^h)} \quad (8)$$

$$\Delta w(e_r^v) = \Delta h \quad (9)$$

$$\sum_{e_r^h \in p} \Delta w(e_r^h) \leq sl^{'}(p) \quad (10)$$

where $w(e_r^v)$ and $w(e_r^h)$ denote the weight of arc e_r^v and e_r^h respectively. From the equations, we derive the width increment of the block as follows:

$$\Delta w(e_r^h) = sl^{'}(p) \times \frac{w(e_r^h)/w(e_r^v)}{\sum_{e_r^h \in p} w(e_r^h)/w(e_r^v)} \quad (11)$$

Accordingly we can obtain the block' height reduction Δh, which is the amount of overall height reduction.

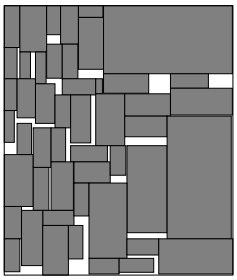

Fig. 3. Placement of MCNC benchmark: ami49 without soft blocks. Achieve total area of 38,813,880 and total wire length of 2,761,122 in 2.27 hours on Sun SPARC 20 workstation. The result is the average of 10 runs.

The process of height reduction terminates when the following condition is true: for any path p from s_h to t_h in G_h, there exists one block corresponding to an arc on p, which is both v-critical and h-critical. When this happens, we cannot further reduce the overall height by changing the heights of blocks corresponding to p without increasing of overall width. The width reduction can be performed similarly. We reduce height and width alternatively until both of them terminate. At this point, no reduction of overall area is possible by reducing the heights or widths of a set of blocks corresponding to a path in G_h or G_v.

Since the overall height and width are reduced monotonically, the algorithm will terminate eventually. Once a block becomes critical in one dimension, it will remain critical in this dimension. This implies that once the height or width of a block is reduced, it will never be increased later. we can show that the floorplan area minimization algorithm runs in linear time.

The experiment result shown in Fig. 4 is the non-slicing floorplanning given each block is soft. For comparison, the placement of same blocks without soft assumption, is shown in Fig. 3. Using the greedy algorithm presented above, we can obtain the area minimization for general floorplanning.

IV. GSA Optimization and Placement of L-shaped, T-shaped Blocks

A. Genetic Simulated Annealing Algorithm

To improve the performance of simulated annealing (SA) and genetic algorithm (GA), several hybrid algorithms were proposed, such as SAGA [11], AG [12] and PGSA [13]. We apply a new optimization method, named genetic simulated annealing (GSA) [3], which combines the local stochastic hill climbing features from SA and the global crossover operations from GA. The further discussion and comparison between GSA and previous hybrid algorithms can be seen in [3].

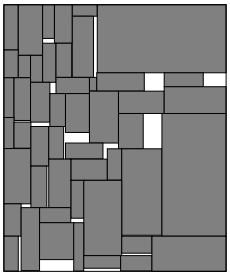

Fig. 4. Floorplanning of MCNC benchmark: ami49 given each block is soft. Achieve total area of 37,334,668 (%3.9 improvement) and total wire length of 2,893,677 in 2.28 hours on Sun SPARC 20 workstation. The result is the average of 10 runs.

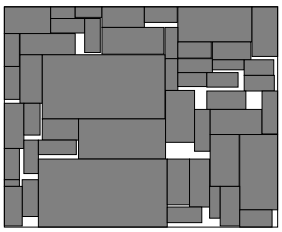

Fig. 5. Placement of MCNC benchmark: ami49. Achieve total area of 37,935,800, and total wire length of 2,761,122 in 1.52 hours on Sun SPARC 20 workstation. The result is the average of 15 runs. Notice that we didn't compare our result with [6] because the current implementation of our approach uses cost function. The final result has to be affected by the coefficient in the cost function.

During the optimization process, GSA maintains a set of solutions named *population*: $S = \{s_1, s_2, s_3, \cdots, s_n\}$ and repeatedly applies the SA-based local search, population update and GA-crossover operation. GSA preserves the local best-so-far solution s_L^* during the SA-based local search, which each time produces a candidate solution $s^{'}$ by changing a small fraction of the current solution s. The new candidate is accepted with probability $\min\{1, e^{-\Delta f/T}\}$, in which Δf is the cost reduction and T is the temperature. Population S is updated by replacing the worst solution with the local best-so-far solution s_L^* at the end of SA-based local search. When the local search reaches a flat surface or the system is frozen, GSA picks up two parent solutions randomly in the population and makes a large jump in the solution space by using a GA-based crossover operation. Crossover generates the new solutions by combining the partial features from both parents. The new SA-based local search starts with the new generation and optimization process continues until CPU time reaches given limit. Finally GSA reports the global best-so-far solution s_G^* in population S.

B. Placement of Rectangular Blocks

During the GSA optimization process, the SA-based operation aims to create small changes of the solution state. It takes a single solution and modifies it at random in a localized manner. Based on the BSG structure, we apply three kinds of SA-based operator : *move*, *exchange* and *rotate*. GSA selects one of these operators at random and applies it in a randomized manner at each optimization step of SA-based local search. On the other hand, GA-based crossover aims to make a big jump in the solution space by combining the partial features of parent solutions. It will be addressed separately. Fig. 5 gives an experiment result for the placement of rectangular blocks. Our algorithm can obtain the high density placement very quickly.

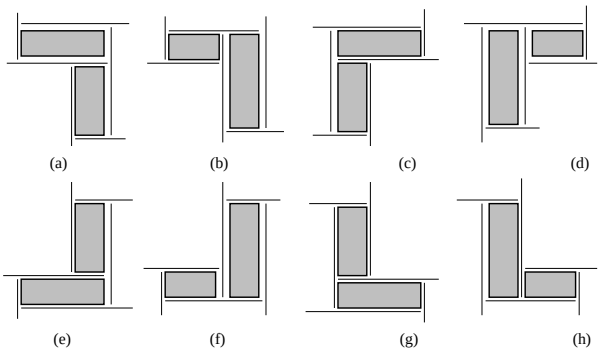

Fig. 6. Efficient L-partition.

C. Placement of L-shaped and T-shaped Blocks

The BSG structure provides a convenient way to handle rectilinear blocks which we may encounter due to the design reuse and special layout constraints. Here we discuss the representations of L-shaped and T-shaped blocks in BSG structure and the extension of optimization operations.

C.1 Representation of L-shaped block in BSG structure

Intuitively we partition an L-shaped block into two sub-rectangles. In BSG structure, there are eight ways in which the two sub-rectangles can form an L-shaped block. Through BSG sizing, other block can be packed into the area enclosed by the sub-rectangles in all of the eight conditions. During the optimization of GSA, the two sub-rectangles are adjacent each other in BSG structure. We extend the operations of SA-based local search to L-shaped blocks. The configuration of L-shaped block depends on that of the target rooms. For example, an L-shaped block shown in Fig. 6(a) is moved into the new

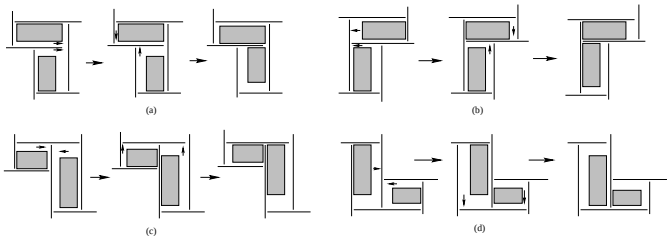

Fig. 7. Coordinate Alignment for L-shaped Block.

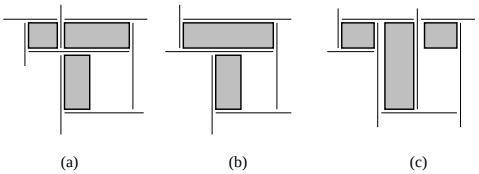

Fig. 8. Non-efficient partition schemes for T-shaped block.

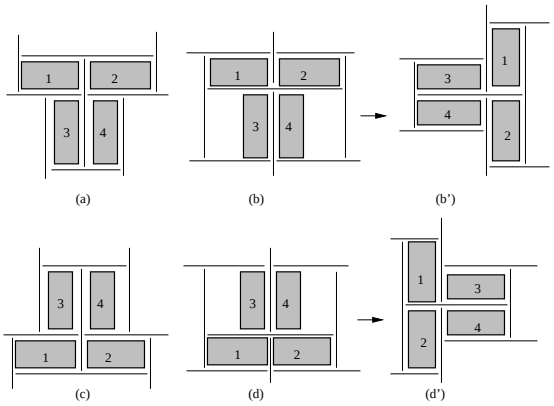

Fig. 9. Efficient partition scheme for T-shaped block.

rooms shwon in Fig. 6(b) and an L-shaped block in Fig. 6 (a) is rotated to (e).

Fig. 7 shows the coordinate alignment for L-shaped blocks. Given L-shaped blocks can be flip, this alignment obtains the feasible configuration of L-shaped block without changing BSG sizing. Therefore, even with non-rectangular blocks, BSG sizing still can evaluate solution accurately during the optimization.

C.2 Representation of T-shaped block in BSG structure

Fig. 8 shows the different ways to partition a T-shaped block to a set of rectangles. They are not efficient because no block can be packed into the area enclosed by the sub-rectangles for some particular BSG configuration. On the other hand, the partition scheme shown in Fig.9 is efficient given that we can replace the four sub-rectangles with each other and rotate them as from Fig.9 (b) to Fig. 9 (b').

Similarly GSA optimization can be extended easily to T-shaped blocks based on the BSG structure. Due to the regular structure of BSG, this kind of operations can be done very quickly. Also we can process the coordinate alignment for T-shaped blocks without changing the BSG sizing.

The experiment result shown in Fig. 10 demonstrates that our algorithm can handle L-shaped and T-shaped blocks efficiently and obtain the packing of rectilinear blocks with high density.

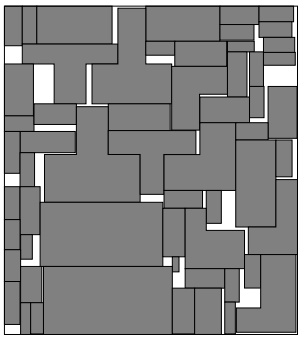

Fig. 10. Placement of modified ami49 with L-shaped and T-shaped blocks. Achieve total area of 60,628,152 and total wire length of 5,359,748 in 3.54 hours on Sun SPARC 20 Workstation. The result is the average of 20 runs.

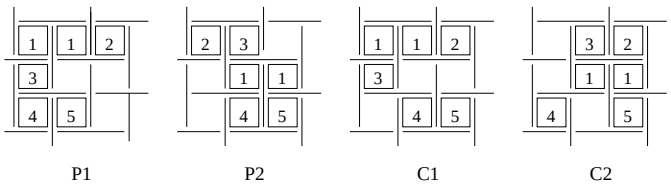

Fig. 11. BSG crossover creates C_1 and C_2 from P_1 and P_2.

D. Crossover Operation

There are three requirements to make the crossover operation desirable [3]. First, crossover should only produce the partial solution which belongs to either parent. All features in child solutions should be inherited from the parent solutions. Second, it should create a child in such a way that the more the parents have in common, the more the child has similarity to the parents. In the extreme case where both parents are identical, the child should be identical to the parents. Finally crossover should produce a feasible child solution.

We extend the crossover operation for rectangular blocks [3] to L-shaped and T-shaped blocks. Each block in child solution is placed in the same room or rooms as in one of the parent solutions. Notice that an L-shaped or T-shaped block occupies more than one rooms. Crossover copies blocks from both parents alternatively in order to inherit features fairly from both parents. Block M_1 is called *conflicting* with block M_2 if the room(s) in P_1 which contains M_1 has corresponding room(s) in P_2 which contains M_2. If a pair of blocks are conflicting with each other, crossover copies both blocks from the same parent. The crossover procedure always creates feasible solutions in which all partial features are consistent with either of the parents. Fig. 11 shows an example of crossover based on BSG structure, $C1$ and $C2$ are generated by starting with copying blocks from $P1$ and $P2$ respectively.

V. Summary and Conclusions

In this paper, we have demonstrated that the BSG structure is very effective for the general floorplanning, including non-slicing structures, L-shaped and T-shaped blocks, and soft blocks. We have presented a new stochastic optimization method GSA, which combines the local stochastic hill climbing features from SA and the global crossover operations from GA. We have also proposed a new efficient algorithm for area minimization of general floorplanning. The experimental results are very promising.

Based on BSG structure, the future work includes generalizing arbitrarily shaped, rectilinear blocks' packing, handling various configuration requirements, incorporating multiple objectives: area, wire length and timing constraints into the cost vector [6] during GSA optimization.

Acknowledgements

The authors like to thank H. Murata, S. Nakatake, and Y. Kajitani for helpful comments on an earlier version of the paper.

References

[1] S. Nakatake, H. Murata, K. Fujiyoshi and Y. Kajitani, "Bounded-Slicing Structure for module placement," *Technical Report of the Institute of Electronics, Information and Communication Engineers of Japan*, Vol. VLD94, pp. 19-24, 1994.

[2] S. Nakatake, K. Fujiyoshi, H. Murata and Y. Kajitani, "Module placement on BSG-structure and IC layout applications," *IEEE/ACM International Conf. on Computer Aided Design*, in press.

[3] S. Koakutsu, M. Kang and W. W.-M. Dai, "Genetic simulated annealing and application to non-slicing floorplan design," *Fifth ACM/SIGDA Physical Design Workshop*, pp. 134-141, 1996.

[4] H. Murata, K. Fujiyoshi, S. Nakatake and Y. Kajitani, "Rectangle-packing-based module placement," *IEEE/ACM International Conf. on Computer Aided Design*, pp. 472-479, 1995.

[5] D. F. Wong and C.L.Liu, "A new algorithm for floorplan design," *Proc. of 23rd ACM/IEEE Design Automation Conf.*, pp. 101-107, 1986.

[6] H. Esbensen and E. S. Kuh, "Exploring the design space for Building-Block placements considering area, aspect ratio, path delay and routing congestion," *Fifth ACM/SIGDA Physical Design Workshop*, pp. 126-133, 1996.

[7] W. W.-M. Dai, B. Eschermann, E. S. Kuh and M. Pedram, "Hierarchical placement and floorplanning in BEAR," *IEEE Trans. Computer-Aided Design*, Vol. CAD-8, pp. 1335-1349, 1989.

[8] R.H.J.M.Otten, "Automatic floorplan design," *Proc. of 19th ACM IEEE Design Automat. Conf.*, pp. 261-267, 1982.

[9] P. Pan and C. L. Liu, "Area minimization for floorplans," *IEEE Trans. on CAD*, Vol. 14, pp. 123-132, 1995.

[10] W. Shi, "An optimal algorithm for area minimization of slicing floorplans," *IEEE/ACM International Conf. on Computer Aided Design*, pp. 480-484, 1995.

[11] H. Esbensen and P. Mazumder, "SAGA: A unification of the genetic algorithm with simulated annealing and its application to marco-cell placement," *Seventh International Conference on VLSI Design*, pp. 211-214, 1994.

[12] F.-T. Lin, C.-Y. Kao and C.-C. Hsu, "Applying the genetic approach to simulated annealing in solving some NP-hard problems," *IEEE Trans. System, Man, and Cybernetics.*, Vol. 23, no. 6, pp. 1752-1767, 1993.

[13] S. Koakutsu, Y. Sugai and H. Hirata, "Floorplanning by improved simulated annealing based on genetic algorithm," *Trans. of the Institute of Electrical Engineers of Japan*, Vol. 112-C, No. 7, pp. 411-416, 1992.