Slicing Tree Is a Complete Floorplan Representation *

Minghorng Lai and D. F. Wong
Department of Computer Sciences, The University of Texas at Austin, Austin, TX 78712, USA

Abstract

Slicing tree has been an effective tool for VLSI floorplan design. Floorplanners using slicing tree representation take full advantage of shape and orientation flexibility of circuit modules to find highly compact slicing floorplans. However, slicing floorplans are commonly believed to suffer from poor utilization of space when all modules are hard. For this reason, a large body of literature has recently been devoted to various new representations of non-slicing floorplans to improve space utilization. In this paper, we prove that by using slicing tree representation and compaction, all maximally compact placements of modules can be generated. In conclusion, slicing tree is a complete floorplan representation for all non-slicing floorplans as well.

1 Introduction

Floorplanning has become one of the most challenging tasks facing VLSI circuit designers as deep sub-micron fabrication technology is close to making billion-transistor chips a reality. As a result of the enormous sizes of the new circuits, it is practically impossible for even the most experienced circuit designers to optimally arrange all the components on a VLSI chip manually without resorting to design automation tools. Furthermore, with rapidly narrowing interconnects and escalating interconnect-delay/gate-delay ratio in circuits, floorplanning is also becoming a crucial phase in timing considerations. Therefore, a good floorplanning strategy is highly desirable in the early stage of the physical design process.

Floorplans can be classified into two categories: slicing and non-slicing. Slicing floorplans, generally represented by slicing trees or Polish expressions, are obtained by recursively bisecting the chip areas horizontally or vertically with a slicing line. There have been several efficient algorithms [4, 10, 13, 17, 18] for finding optimal slicing floorplans. These algorithms are very efficient because the slicing structure limits the size of the solution space and because they exploit shape and orientation flexibility of the modules. Slicing tree representation has been shown to be a good choice for handling various placement constraints as well [17]. Furthermore, it has been mathematically proven that slicing floorplans are capable of producing compact placement for modules with shape flexibility [16].

However, it is still commonly believed that even an optimal slicing floorplan suffers from poor utilization of space. For this reason, many efforts have been devoted to creating representations of non-slicing floorplans [1, 2, 3, 5, 6, 7, 8, 9, 11, 12, 14, 15]. Although they are efficient for handling hard modules, many of these representations are unable to take advantage of shape and orientation flexibility of soft modules. As a result, they either use very complicated specialized procedures or require computation-intensive techniques for sizing soft modules.

In this paper we prove that slicing tree is a complete representation of general floorplans. We show that augmented with a simple compaction procedure, slicing tree representation can generate all maximally compact placements of modules. In conclusion, slicing tree is a complete representation of both slicing and non-slicing floorplans.

The remainder of the paper is organized as follows: In Section 2, we discuss the general floorplan design problem. In Section 3, we review the slicing tree representation of slicing floorplans. In Section 4, we prove that slicing tree, when augmented with a simple compaction procedure, is a complete representation of floorplans. In Section 5, we provide some concluding remarks.

2 Floorplan Design Problem

A module M_i is a rectangle with fixed area A_i, height h_i, and width w_i. The aspect ratio r_i of module M_i is defined to be h_i/w_i. A hard module is a circuit component with fixed height and width. A soft module has flexibility in its shape: the aspect ratio of a soft module is required to be in a range $[r_{min}, r_{max}]$. A module has free orientation if rotation of the module is allowed. A module with fixed orientation cannot be rotated freely.

A placement of the modules fixes the locations of the modules in the floorplan. A feasible placement is one in which no modules overlap each other and all soft modules are consistent with their aspect ratio constraints. The area A of a placement is defined to be the area of the smallest enclosing rectangle. The total wiring cost is W. The main objective of floorplan design is to find a feasible placement of a set of modules that minimizes the cost function $A+\lambda W$.

If most modules in the set are soft modules, slicing floorplanners using slicing tree representation (Section 3) exploit the shape and orientation flexibility to find highly compact slicing floorplans. In Section 4, we show that augmented with compaction, slicing tree representation is also able to represent all non-slicing placements of modules.

3 Slicing Tree Representation of Slicing Floorplans

A *slicing floorplan* is a rectangular area that is sliced recursively by a horizontal or vertical slicing line into a set of rectangular regions, called *rooms*, to accommodate a set of

*This work was partially supported by the National Science Foundation under grant CCR-9912390, by the Texas Advanced Research Program under Grant No. 003658288, and by grants from Avant!, Intel and IBM.

circuit modules M. Slicing floorplans (Fig. 1) are generally represented by slicing trees (or equivalently, by Polish expressions.)

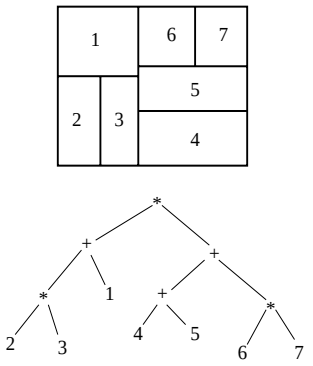

Figure 1: Slicing floorplan, slicing tree, and Polish expression.

The leaf nodes in a slicing tree represent the modules. Each internal node of a slicing tree is labeled by operator $*$ or operator +, corresponding to a vertical or horizontal slicing line, respectively. Every subtree rooted at an internal node represents a supermodule consisting of one or more component modules and/or supermodules. In a slicing floorplan, each supermodule has a rectangular shape. The Polish expression for a slicing floorplan is obtained by traversing the slicing tree in postorder.

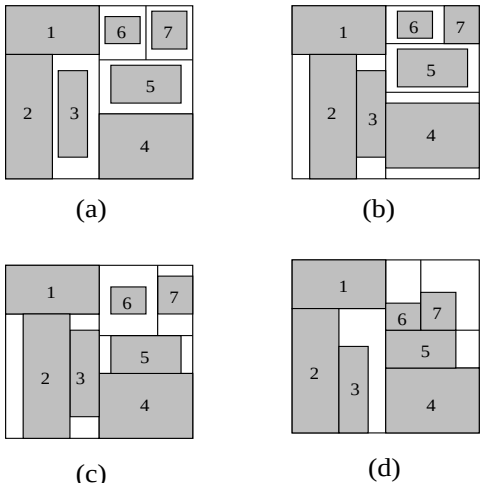

Figure 2: Slicing floorplans and placements corresponding to the same slicing tree in Fig. 1. Floorplans in (a), (b), (c), and (d) are equivalent floorplans represented by the same slicing tree. The slicing lines in floorplan (d) cannot be moved horizontally to the left or vertically downward. Additionally, the modules in (d) are all placed in the lower left corner in their individual rooms. The placement in (d) is defined to be the slicing placement for the slicing tree in Fig. 1.

We note that a slicing tree is merely a hierarchical description of the bisections of rectangular areas in the floorplan - no dimensional or positional information of the modules is specified. One of the objective of floorplan design is to minimize the area of the bounding rectangle. As a result, for a slicing tree T, slicing floorplanners produce a slicing floorplan with the smallest bounding rectangle. Sometimes there exists a small amount of dead space even in the smallest bounding rectangle. In this case, the slicing lines are free to move in the floorplan, without increasing the area of the bounding rectangle, as long as the rooms stay large enough to accommodate the modules. Therefore, a slicing tree represents a set of equivalent slicing floorplans with the same dimensions (Fig. 2). For example, floorplan 2(a), 2(b), and 2(c) have exactly the same dimensions; however, the locations of slicing lines are different. For our discussion, we assume that in the corresponding slicing floorplan for a slicing tree T, no horizontal slicing lines can be moved to the left and no vertical slicing lines can be moved downward. For each slicing tree, there is exactly one such corresponding slicing floorplan (Fig. 2(d)). The modules can be placed anywhere in their rooms. We define the *slicing placement* P_s of a slicing tree T to be the placement where each module is placed in the lower left corner of its room in the corresponding slicing floorplan for T. In Fig. 2, placement 2(d) is the slicing placement for the slicing tree in Fig. 1.

When most modules are soft, slicing floorplanners have been mathematically proven to be very efficient for generating highly compact placements [16]. Experiments show that slicing tree representation is able to produce close-to-zero dead space for soft modules. However, when all modules are hard, it is believed that even an optimal slicing floorplan suffers from poor utilization of space because slicing trees cannot represent non-slicing floorplans. We will show in next section that by performing simple compaction, slicing tree representation can generate all maximally compact placements of modules.

4 Slicing Tree Is a Complete Representation of Non-Slicing Floorplans

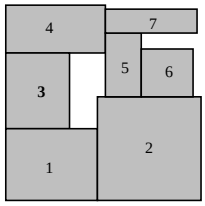

Figure 3: A maximally compact placement of $M = \{1, 2, 3, 4, 5, 6, 7\}$.

We define a *maximally compact placement* (Fig. 3) of a set of modules to be a placement in which no module can move horizontally to its left or vertically downward without moving any other modules (This is equivalently defined as an admissible packing in [2].) Notice that the slicing placement P_s of a slicing tree T is not necessarily a maximally compact placement since if slicing lines are removed, modules in slicing placement P_s are then free to move left or downward (Fig. 4). We prove in this section that slicing tree

representations, augmented with a simple compaction procedure, generate all maximally compact placements of modules.

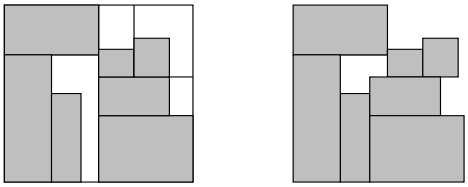

Figure 4: Slicing placement is not necessarily a maximally compact placement.

The x-compaction is a procedure that slides modules in a placement horizontally toward the left boundary of the floorplan. The y-compaction slides modules in a placement vertically toward the lower boundary of the floorplan. The iterative xy-compaction is a sequence of successive x- and y-compactions (Fig. 5). The iterative yx-compaction is a sequence of successive y- and x-compactions. Compaction is especially helpful for placements in slicing floorplans. In Fig. 5, the dead space was reduced from over 20% to a highly compact placement by simple compaction procedure. Note that the placement produced by the compaction is a non-slicing placement.

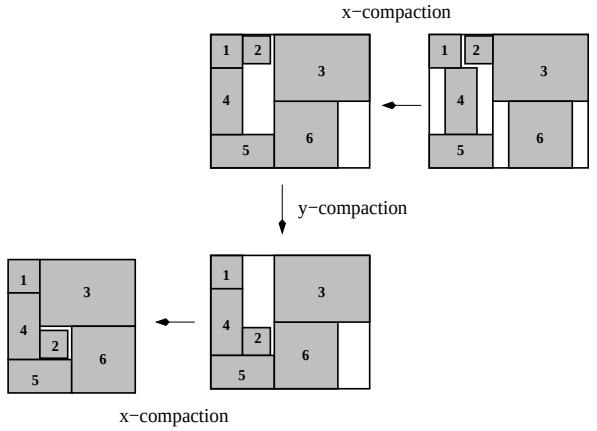

Figure 5: A maximally compact placement is obtained by performing an iterative xy-compaction on a placement in slicing floorplan.

Given any placement of modules, we can construct a *horizontal adjacency graph* $G = (V, E)$ as follows. The set of vertices V corresponds to the set of modules. There is an edge (u, v) in E if and only if the left boundary of v is immediately adjacent to (i.e. touching) the right boundary of u. The *vertical adjacency graph* can be similarly defined in terms of the abuttment of the top/bottom boundaries of the modules. It is easy to see that G is a directed acyclic graph. Fig. 6 shows the horizontal adjacency graphs of two different placements where the placement in (a) is not maximally compact but the one in (b) is.

A vertex u in G is said to be a left-boundary vertex if u is a module placed on the left boundary of the placement

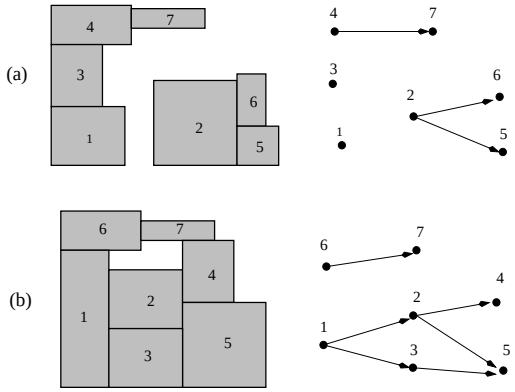

Figure 6: Placements and their corresponding horizontal adjacency graphs.

area. For example, vertices 1, 3, and 4 in Fig. 6(a) and vertices 1 and 6 in Fig. 6(b) are left-boundary vertices. It is easy to see that all left-boundary vertices have in-degree 0. The converse may not be true in general (e.g., vertex 2 in Fig. 6(a)), but it is true for maximally compact placements (see Fig. 6(b)). Another important observation is that for a maximally compact placement, all vertices in G are connected to the set of left-boundary vertices. We summarize our observations in the following Lemma 1.

Lemma 1 *Let G be the horizontal adjacency graph of a maximally compact placement. Let B be the set of left-boundary vertices. We have (1) G is a directed acyclic graph, (2) B is exactly the set of vertices with in-degree 0, and (3) each vertex v in G is reachable from at least one vertex u in B (i.e., there exists a path from u to v).*

We prove next that we can obtain all maximally compact placements of modules by performing compaction on slicing floorplans.

Theorem 1 *Given any maximally compact placement P, there exists a slicing tree T such that performing compaction on the slicing placement P_s of T generates P.*

Proof:

Let G be the horizontal adjacency graph of P. Let $B = \{b_1, b_2, \cdots, b_m\}$ be the set of left-boundary vertices. According to Lemma 1, every vertex in G is reachable from at least one vertex in B. It follows that we can find a spanning forest $Q = \{T_1, T_2, \cdots, T_m\}$ of G where T_i is a tree rooted at left-boundary vertex b_i, $i = 1, 2, \cdots, m$. (Recall that a spanning forest of a graph is a collection of disjoint trees which are subgraphs of the given graph and that every vertex is in one of the trees.) For example, the graph in Fig. 7(b) is a spanning forest (consisting of three trees) of the horizontal adjacency graph in Fig. 7(a) with left-boundary vertices A, D, and G as the roots of the trees.

We now describe how to obtain a slicing tree from Q. We may assume that $b_1, b_2, \cdots, b_m$ are in the order of increasing y positions. We create $m - 1$ horizontal slicing lines dividing the floorplan into m panels for the trees rooted at $b_1, b_2, \cdots, b_m$. For each tree, the transformations shown

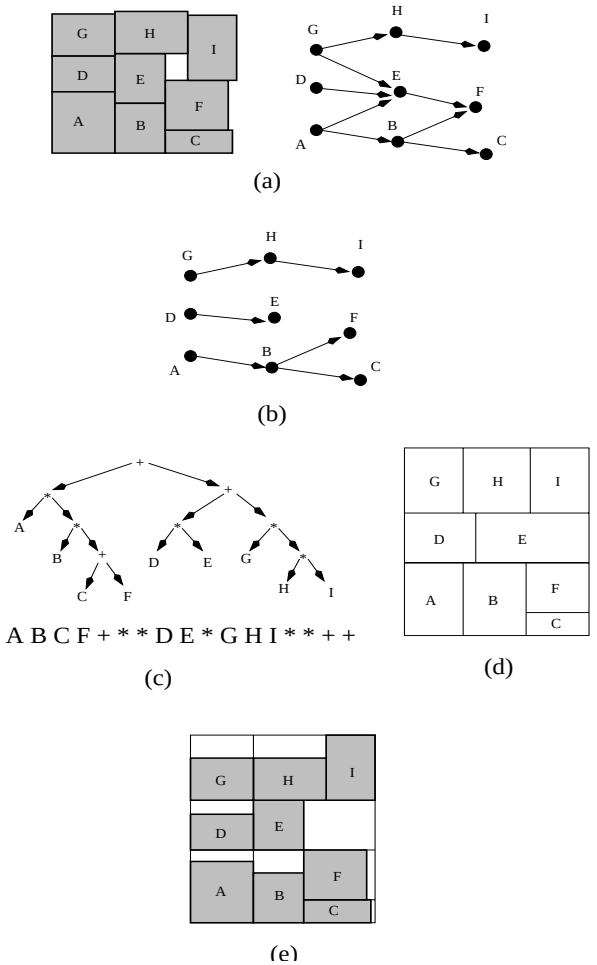

Figure 7: Generating a slicing placement from a spanning forest in the horizontal adjacency graph. A maximally compact placement and its horizontal adjacency graph is shown in (a). One spanning forest in the horizontal adjacency graph is shown in (b). For the spanning forest in (b) we can generate a slicing tree T (c) and its corresponding slicing placement P_s (e) such that performing compaction on P_s generates the original maximally compact placement.

in Fig. 8 are applied recursively to the sub-trees to further expand the slicing structure of the floorplan. For example, in Fig. 7(d) two slicing lines divide the floorplan into three horizontal panels for the three trees in Fig. 7(b). Then the transformation procedures in Fig. 8 are applied recursively to generate the slicing floorplan in Fig. 7(d). From the slicing floorplan, we can generate the slicing placement P_s (Fig. 7(e)) corresponding to the slicing tree. (Remember in a slicing placement, the slicing lines cannot move left or downward. Furthermore, the modules are placed in the lower left corner of their individual rooms.) It can be shown that the x-positions of the modules in the slicing placement are the same as their original x-positions in the maximally compact placement P. Therefore, a simple y-compaction transforms P_s into P.

Given a maximally compact placement, the formal procedure for generating a slicing tree T is shown in Fig. 9.

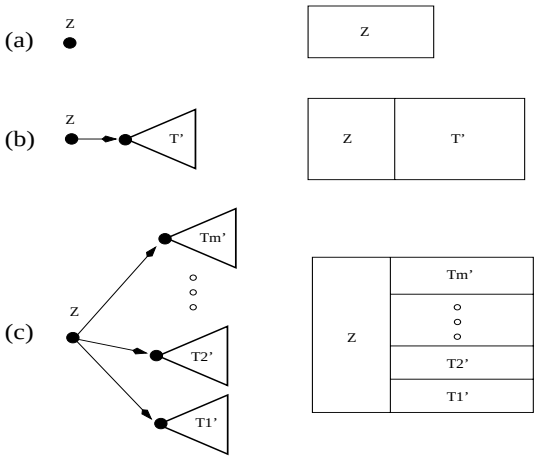

Figure 8: Generating slicing floorplans from trees in spanning forests.

`Input`: A maximally compact placement P
`Output`: A slicing tree T (in Polish expression)

Generate a horizontal adjacency graph G for P.
Find a spanning forest in G: $\{T_1, T_2, \cdots, T_n\}$. ($T_i's$ are in the order of increasing y-positions of root modules.)
return g(T_1) g(T_2) $\cdots$ g(T_n) $+_1 +_2 \cdots +_{(n-1)}$

g(tree T) {
  u = root of T
  `if` u has no children
    return u
  `if` u has one child v
    T_v = subtree rooted at v
    return u (g(T_v)) $*$
  `if` u has m children $v_1, v_2, \cdots v_m; m \geq 2$
    T_{v_i} = subtree rooted at v_i
    return u g(T_{v_1}) g(T_{v_2}) $\cdots$ g(T_{v_m}) $+_1 +_2 \cdots +_{(m-1)} *$
}

Figure 9: Generating a slicing tree T from a spanning forest in the horizontal adjacency graph of a maximally compact placement P.

The slicing tree is represented by its equivalent Polish expression. The procedure described in Fig. 9 is based on a breadth-first marking procedure. We start this procedure on each of the trees in the spanning forest, in the order of increasing y-positions of the modules at the roots of the trees. We traverse each tree in a breadth-first fashion, generating the Polish expression for one level of the tree at a time. This is a very efficient procedure with linear complexity. For the spanning forest in Fig. 7(b), the procedure generates the slicing tree T in Fig. 7(c). Fig. 7(e) is the slicing placement of T in Fig. 7(c). We then perform a simple y-compaction on the slicing placement in Fig. 7(e) to generate the original maximally compact placement P in Fig. 7(a).

Another example of slicing tree generation is shown in Fig. 10. In this example, the vertical adjacency graph and its spanning forest are used to generate the slicing tree. In this case, the x-compaction is used to transform the slicing

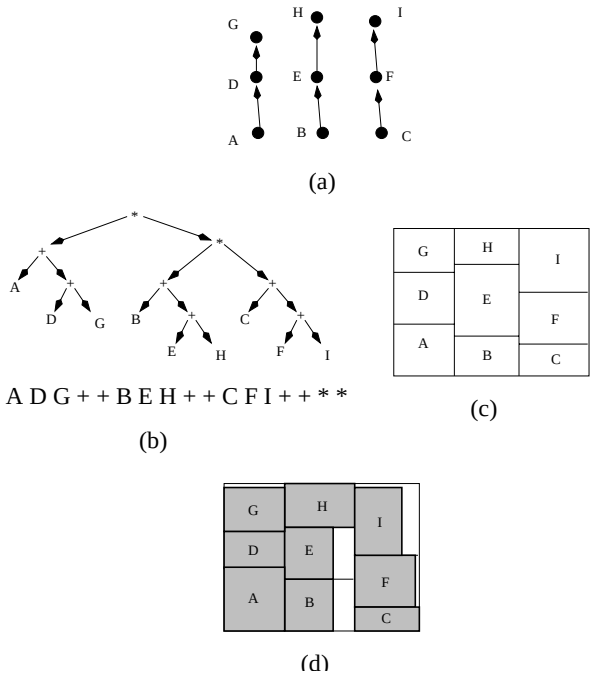

Figure 10: Using vertical adjacency graph (a) to generate a slicing tree T (b), the slicing floorplan (c), and its corresponding slicing placement P_s (d).

floorplan into the original non-slicing placement. □

It follows from Theorem 1 that slicing tree is a complete representation of floorplans in the sense that by using slicing tree representation and compaction, all maximally compact placements can be generated.

5 Concluding Remarks

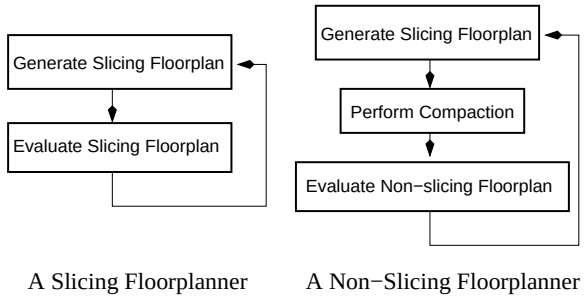

Figure 11: Slicing and non-slicing floorplanners.

We have proven that slicing tree is a complete representation of general floorplans. A simple compaction procedure extends the ability of slicing floorplanners to include generating non-slicing placements. As a result, a new set of non-slicing floorplanners can be based on existing slicing floorplanners, which are highly efficient for sizing soft modules and processing floorplanning constraints (Fig. 11). Although theoretically it is sufficient to perform only a single x- or y-compaction, in practical implementations it may be preferable to perform iterative xy- or yx-compactions to ensure that a maximally compact placement is evaluated during each iteration. We have shown that slicing tree representations can be used to generate all maximally compact placements of modules. Therefore, floorplanning based on slicing tree representation should remain an active area of research.

References

[1] Y.-C. Chang et al., "B*-Trees: A New Representation for Non-Slicing Floorplans," *Proc. ACM/IEEE Design Automation Conference*, 2000, To Appear.

[2] P. Guo, C.-K. Cheng, and T. Yoshimura, "An O-Tree Representation of Non-Slicing Floorplan and Its Applications," *Proc. ACM/IEEE Design Automation Conference*, 1999, pp. 268-273.

[3] M. Z. Kang and W. M. Dai, "Arbitrary Rectilinear Block Packing Based on Sequence Pair," *Proc. Int'l Conf. on Computer-Aided Design*, 1998, pp. 259-266.

[4] D. P. Lapotin and S. W. Director, "A New Algorithm for Floorplan Design," *Proc. Int'l Conf. on Computer-Aided Design*, 1985, pp. 143-145.

[5] H. Murata, K. Fujiyoshi, and M. Kaneko, "VLSI/PCB Placement with Obstacles Based Sequence pair," *Proc. Int'l Symp. on Physical Design*, 1997, pp. 26-31.

[6] H. Murata, and E. S. Kuh, "Sequence Pair Based Placement Method for Hard/Soft/Pre-placed Modules," *Proc. Int'l Symp. on Physical Design*, 1998, pp. 167-172.

[7] H. Murata et al., "Rectangle-Packing Based Module Placement," *Proc. Int'l Conf. on Computer-Aided Design*, 1995, pp. 472-479.

[8] S. Nakatake et al., "Module Placement on BSG-Structure and IC Layout Applications," *Proc. Int'l Conf. on Computer-Aided Design*, 1996, pp. 484-491.

[9] S. Nakatake et al., "Module Placement on BSG-Structure with Pre-Placed Modules and Rectilinear Modules," *Proc. Asia and South Pacific Physical Design Automation Conf.*, 1998, pp. 571-576.

[10] R. H. J. M. Otten, "Automatic Floorplan Design," *Proc. ACM/IEEE Design Automation Conference*, 1982, pp. 261-267.

[11] Y. Pang, C.-K. Cheng, and T. Yoshimura, "An Enhanced Perturbing Algorithm for Floorplan Design Using the O-tree Representation," *Proc. Int'l Symp. on Physical Design*, 2000.

[12] T. C. Wang and D. F. Wong, "An Optimal Algorithm for Floorplan and Area Optimization," *Proc. ACM/IEEE Design Automation Conference*, 1990, pp. 180-186.

[13] D. F. Wong and C. L. Liu, "A New Algorithm for Floorplan Design," *Proc. ACM/IEEE Design Automation Conference*, 1986, pp. 101-107.

[14] D. F. Wong and C. L. Liu, "Floorplan Design for Rectangular and L-shaped Modules," *Proc. Int'l Conf. on Computer-Aided Design*, 1987, pp. 520-523.

[15] J. Xu, P. Guo, and C.-K. Cheng, "Rectilinear Block Placement Using Sequence-Pair," *Proc. Int'l Symp. on Physical Design*, 1998, pp. 173-178.

[16] F. Y. Young, and D. F. Wong, "How Good Are Slicing Floorplans," *Proc. Int'l Symp. on Physical Design*, 1997.

[17] F. Y. Young, and D. F. Wong, "Slicing Floorplans with Pre-Placed Modules," *Proc. Int'l Conf. on Computer-Aided Design*, 1998, pp. 252-258.

[18] T. Yamanouchi, K. Tamakashi, and T. Kambe, "Hybrid Floorplanning Based on Partial Clustering and Module Restructuring," *Proc. Int'l Conf. on Computer-Aided Design*, 1996, pp. 478-483.