Arbitrarily Shaped Rectilinear Module Placement Using the Transitive Closure Graph Representation

Jai-Ming Lin, Hsin-Lung Chen, and Yao-Wen Chang, *Member, IEEE*

***Abstract*—In this paper, we deal with arbitrarily shaped rectilinear module placement using the transitive closure graph (TCG) representation. The geometric meanings of modules are transparent to TCG as well as its induced operations, which makes TCG an ideal representation for floorplanning/placement with arbitrary rectilinear modules. We first partition a rectilinear module into a set of submodules and then derive necessary and sufficient conditions of feasible TCG for the submodules. Unlike most previous works that process each submodule individually and thus need to perform post processing to fix deformed rectilinear modules, our algorithm treats a set of submodules as a whole and thus not only can guarantee the feasibility of each perturbed solution but also can eliminate the need for the postprocessing on deformed modules, implying better solution quality and running time. Experimental results show that our TCG-based algorithm is capable of handling very complex instances; further, it is very efficient and results in better area utilization than previous work.**

***Index Terms*—Floorplanning, placement, transitive closure graph.**

I. Introduction

As technology advances, design complexity is increasing at a dramatic pace. To handle the design complexity, hierarchical designs and IP modules are widely used for design convergence. These trends make floorplanning/placement more important than ever. Traditional placement is to determine the locations for a set of rectangular modules such that no module overlaps and some predefined cost metric (e.g., area, wirelength, and/or routability) is optimized. However, to fully optimize silicon area and/or wirelength, nonrectangular modules are also used. Therefore, how to deal with the placement of arbitrarily shaped rectilinear modules becomes an important issue in the floorplan design.

A. Previous Work

Placement/floorplanning with rectilinear modules has been extensively studied in the literature [2], [5]–[7], [10], [16], [20], [21]. In [10], Lee represented an arbitrarily shaped rectilinear module with a set of four linear profiles which describe the contours of a module viewed from four sides. He minimized chip size by performing a bounded two-dimensional (2-D) contour searching algorithm on the profile of a design. Due to the high complexity for computing the profiles, the approach is limited to the placement problem with a small number of modules.

Unlike the work in [10], most previous work partitioned a rectilinear module into a set of rectangular submodules and operated on the submodules under some constraints induced from the original rectilinear module. There are a few existing partition based approaches using well-known representations: for example, bounded-sliceline grid (BSG) [5], [7], [16], sequence pair [2], [6], [21], B^*-tree [20], O-tree [18], and corner block list (CBL) [13].

Murata *et al.* in [14] used two sequences of module names, namely sequence pair, to represent the geometric relations of modules for nonslicing floorplan design. Xu *et al.* in [21] explored the conditions of feasible sequence pair for L-shaped modules. After all rectangular modules and submodules were packed, a post processing was performed to adjust misplaced submodules to fix the shapes of rectilinear modules. However, they can only deal with "mound-shaped" rectilinear modules. Kang and Dai in [6] derived three necessary and sufficient conditions for recovering the shapes of convex rectilinear modules. Similarly, they also needed a post processing to retrieve the original shapes of rectilinear modules after packing. Recently, Fujiyoshi and Murata in [2] presented an approach to represent rectilinear modules using sequence pair. They also derived a necessary and sufficient condition for feasible sequence pair for rectilinear modules. In particular, they augmented a constraint graph by adding constraint edges to examine the feasibility of a sequence pair and packed modules, without resorting to a post processing for fixing misplaced submodules. However, the constraint graphs are no longer acyclic after the augmentation, resulting in a longer running time for packing ($O(m^3)$ time, where m is the number of modules).

Nakatake *et al.* in [15] proposed the BSG representation, which is composed of a set of horizontal and vertical line segments. The rectangular region enclosed by four line segments is called a room; floorplanning can be done by assigning modules into rooms. Kang and Dai in [5] proposed a BSG-based method to pack L-shaped, T-shaped, and soft modules by using a stochastic approach that combines simulated annealing and a genetic algorithm. Nakatake *et al.* in [16] handled pre-placed and rectilinear modules using BSG. To handle a rectilinear module, they placed its submodules one by one until all submodules were packed at the right relative positions. Then, the placed submodules were treated as pre-placed modules. Kang and Dai in [7] used BSG and sequence pair to solve the topology constrained module packing for a specific class of

Manuscript received September 25, 2001; revised January 24, 2002. This work was supported in part by the National Science Council of Taiwan, R.O.C., under Grant NSC-89-2215-E-009-117.

J.-M. Lin is with Realtek Semiconductor Corporation, Hsinchu 300, Taiwan, R.O.C. (e-mail: gis87808@cis.nctu.edu.tw).

H.-L. Chen is with Etron Technology, Inc., Hsinchu 300, Taiwan, R.O.C. (e-mail: is85019@cis.nctu.edu.tw).

Y.-W. Chang is with the Department of Electrical Engineering and the Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 106, Taiwan, R.O.C. (e-mail: ywchang@cc.ee.ntu.edu.tw).

Digital Object Identifier 10.1109/TVLSI.2002.808431

rectilinear modules, named *ordered convex rectilinear modules*, and extended the method to handle arbitrary rectilinear modules.

Guo *et al.* in [3] first proposed the O-tree representation for a left and bottom compacted placement. In a horizontal O-tree, a node denotes a module and an edge denotes the horizontal adjacency relation of two modules. Each O-tree is encoded by a pair of strings, denoting the DFS traversal order of the tree. Pang *et al.* recently in [18] used the O-tree representation to handle rectilinear modules. The packing scheme has relatively lower time complexity, but the overall approach is less flexible and more restricted. For each rectilinear module, they need to partition it into a set of L-shaped submodules and all modules must be compacted.

Chang *et al.* in [1] presented a binary tree-based representation for a left and bottom compacted placement, called B^*-tree, and showed its superior properties for operations. In a B^*-tree, a node denotes a module, the left child of a node represents the lowest adjacent module on the right, and the right child represents the first module above and with the same x coordinate. Wu *et al.* in [20] handled rectilinear modules using the B^*-tree representation. A rectilinear module can easily be represented using B^*-tree by vertically partitioning the module into a set of rectangular submodules. However, they need to repartition a rectilinear module whenever the rectilinear module is rotated. Besides, they need a postprocessing to adjust submodules to maintain the shapes of rectilinear modules.

Recently, Hong *et al.* in [4] proposed the CBL representation for nonslicing floorplans. CBL is composed of three tuples that denote the packing order of modules, the orientations of modules, and the number of modules that each module covers, respectively. Ma *et al.* in [13] used CBL to deal with the placement abutment constraint and extended the method to deal with L- and T-shaped modules.

B. Our Contribution

In this paper, we deal with arbitrarily shaped rectilinear module placement using the transitive closure graph (TCG) representation. We first partition a rectilinear module into a set of submodules and then derive necessary and sufficient conditions of feasible TCG for the submodules. The geometric relationship of modules/submodules is transparent to TCG and its induced operations, implying that any violation of the topology of a rectilinear module during perturbation can easily be detected. Unlike most previous methods that process each submodule individually and thus need post processing to fix deformed rectilinear modules, our algorithm treats a set of submodules as a whole and thus not only can guarantee the feasibility of each perturbed solution, but also can eliminate the need of the post processing on deformed modules, implying better solution quality and running time. In particular, our packing scheme takes only $O(m^2)$ time, compared to $O(m^3)$ time for the sequence pair based method for arbitrarily shaped modules presented in [2]. All these properties make TCG an ideal representation for dealing with the floorplan/placement design with rectilinear modules. Experimental results show that our TCG-based algorithm is capable of handling very complex instances; further, it is very efficient and results in better area

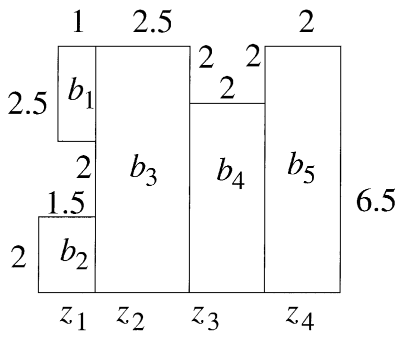

Fig. 1. A concave rectilinear module consisting of four zones z_1, z_2, z_3, z_4, where $z_1 = \{b_1, b_2\}$, $z_2 = \{b_3\}$, $z_3 = \{b_4\}$, and $z_4 = \{b_5\}$.

utilization (average dead space $=$ 5.00%) than the previous work [21] (average dead space = 7.65%).

The remainder of this paper is organized as follows. Section II formulates the floorplan/placement design problem with rectilinear modules. Section III reviews the TCG representation. Section IV presents the feasible TCG and packing algorithm for convex and some concave rectilinear modules. Section V introduces the perturbation algorithm for rectilinear modules. Section VI extends TCG to deal with general rectilinear modules. Experimental results are reported in Section VII, and concluding remarks are given in Section VIII.

II. Preliminaries

Rectilinear modules can be classified into two types: *convex rectilinear modules* and *concave rectilinear modules*. A rectilinear module is said to be *convex* if, for any two points in the module, they have a shortest Manhattan path inside the module; the module is said to be *concave*, otherwise. Besides, a module is said to be *sliceable* if there exists a horizontal or a vertical slicing on the module and the slicing does not result in two separate submodules; otherwise, it is *nonsliceable*.

Theorem 1: All convex modules must be sliceable.

Proof: If there exists a nonsliceable convex module, we can slice the module along vertical or horizontal boundaries and get two separate submodules. The shortest Manhattan path between two arbitrary points, one in each submodule, must be outside the module, contradicting to the definition for the convex rectilinear module. ■

Corollary 1: All nonsliceable modules are concave modules.

Proof: Given a nonsliceable module, there exist two separate submodules after slicing the module along horizontal or vertical boundaries. The shortest Manhattan path for two arbitrary points in the separate submodules must be outside the module. Therefore, the nonsliceable module must be a concave one. ■

In the nonsliceable module shown in Fig. 1, there exist two separate submodules b_1 and b_2 resulting from slicing the module along the vertical boundaries. The shortest Manhattan path for two arbitrary points in b_1 and b_2 is outside the module, and thus it is a concave module.

A rectilinear module b can further be partitioned into a set of zones $\{z_1, z_2, \ldots, z_p\}$ by serially slicing the module vertically (horizontally), and each zone z_i consists of a set of rect-

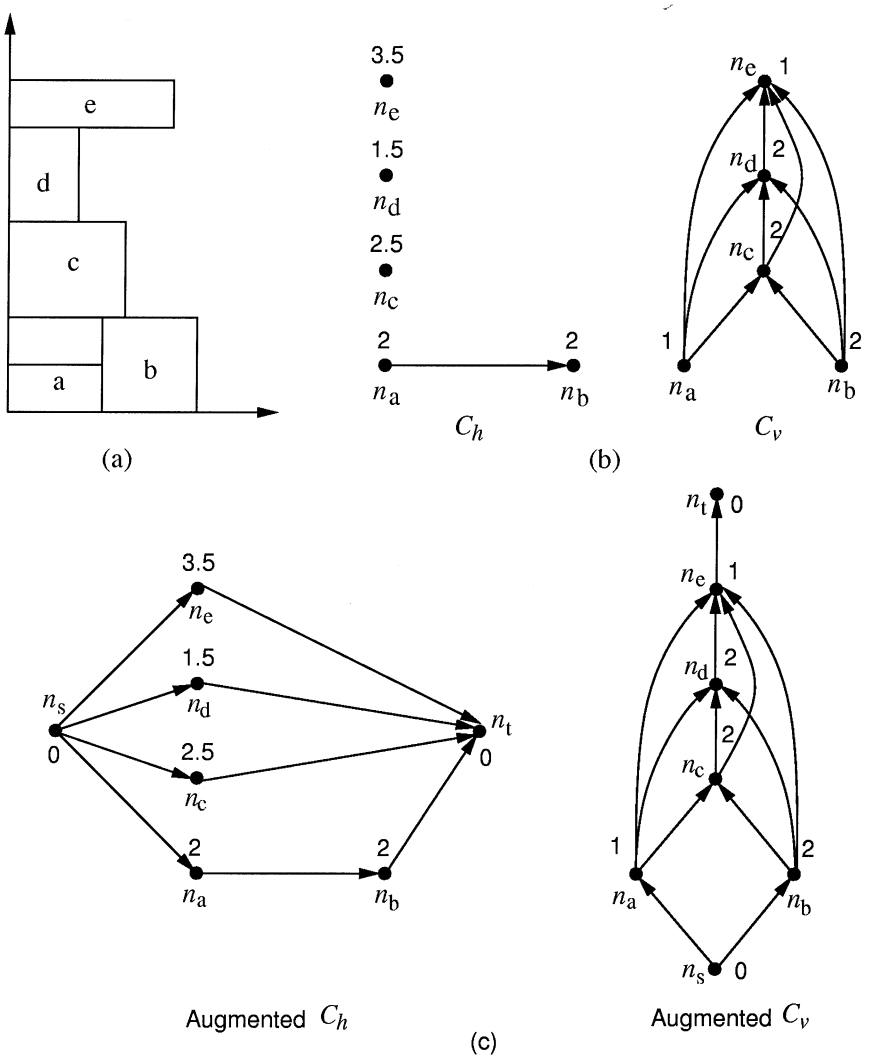

Fig. 2. (a) A placement in a chip. (b) TCG. (c) Augmented TCG (Augmented C_h and Augmented C_v).

angular submodules $\{b_{i_1}, b_{i_2}, \ldots, b_{i_k}\}$ ordered from bottom to top (from left to right). Fig. 1 shows a nonsliceable rectilinear module with four zones z_1, z_2, z_3, and z_4 by serially slicing along vertical boundaries, where $z_1 = \{b_1, b_2\}$, $z_2 = \{b_3\}$, $z_3 = \{b_4\}$, and $z_4 = \{b_5\}$. The number labeled beside each boundary of the module gives the length of the boundary.

Let $W_i \quad (W_{i_j})$, $H_i \quad (H_{i_j})$, $A_i \quad (A_{i_j})$, and (X_i, Y_i) $((X_{i_j}, Y_{i_j}))$ denote respective width, height, area, and the coordinate of the bottom-left corner of the module b_i (submodule b_{i_j}). A placement P is an assignment of $(X_i, Y_i), i = 1, \ldots, m$, for each b_i such that no two modules overlap and the shape of each rectilinear module is maintained. The goal of placement with rectilinear modules is to optimize a predefined cost metric such as the resulting area (i.e., the minimum bounding rectangle of P) and/or wirelength (i.e., the summation of half bounding box of interconnections) induced by a placement.

III. Review of TCG

We first review the TCG representation presented in [11]. TCG describes the geometric relations among modules based on two graphs, namely a *horizontal transitive closure graph* C_h and a *vertical transitive closure graph* C_v, in which a node n_i represents a module b_i and an edge (n_i, n_j) in $C_h \quad (C_v)$ denotes that module b_i is left to (below) module b_j. TCG has the following three *feasibility properties* [11].

1) C_h and C_v are acyclic.
2) Each pair of nodes must be connected by exactly one edge either in C_h or in C_v.
3) The transitive closure of $C_h \quad (C_v)$ is equal to $C_h \quad (C_v)$ itself. The transitive closure of a directed acyclic graph G is defined as the graph $G' = (V, E')$, where $E' = \{(n_i, n_j)$: there is a path from node n_i to node n_j in $G\}$.

Fig. 2(a) shows a placement with five modules a, b, c, d, and e whose widths and heights are (2, 1), (2, 2), (2.5, 2), (1.5, 2), and (3.5, 1.5), respectively. Fig. 2(b) shows the TCG $= (C_h, C_v)$ corresponding to the placement of Fig. 2(a). The value associated with a node in $C_h \quad (C_v)$ gives the width (height) of the corresponding module, and the edge (n_i, n_j) in $C_h \quad (C_v)$ denotes the horizontal (vertical) relation of b_i and b_j. Since there exists an edge (n_a, n_b) in C_h, module b_a is left to b_b. Similarly, b_a is below b_c since there exists an edge (n_a, n_c) in C_v.

Given a TCG, a placement can be obtained in $O(m^2)$ time by performing a well-known *longest path algorithm* [9] on TCG,

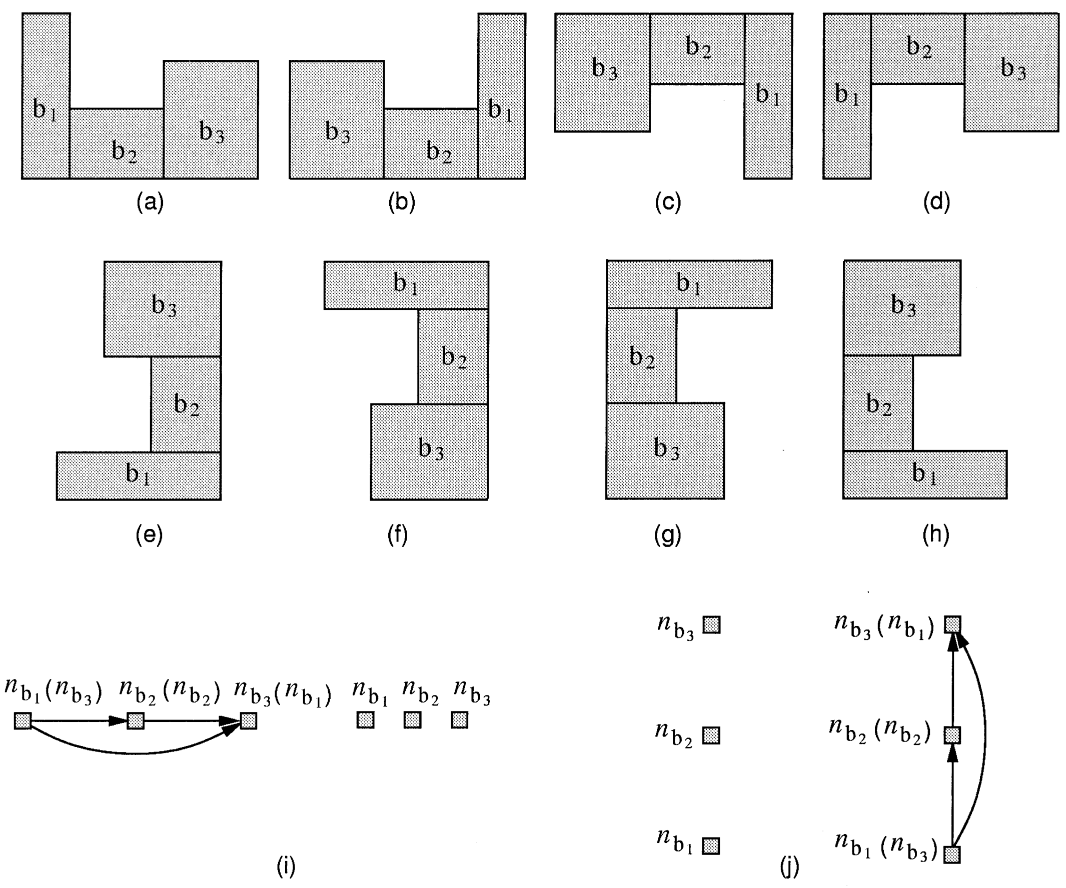

Fig. 3. (a)–(h) Eight situations of the submodules for the rectilinear module b_b. (i) The components in C_h and C_v corresponding to the rectilinear modules shown in (a)–(d). (j) The components in C_h and C_v corresponding to the rectilinear modules shown in (e)–(h).

where m is the number of modules. To facilitate the implementation of the longest path algorithm, the two closure graphs can be augmented as follows. For each closure graph, we introduce two special nodes, the source n_s and the sink n_t, both with zero weights, and construct edges from n_s to each node with in-degree equal to zero as well as from each node with out-degree equal to zero to n_t. (Note that the TCG augmentation is performed only for packing.) Fig. 2(c) shows the augmented TCG for the TCG shown in Fig. 2(b).

Let $L_h(n_i)$ $(L_v(n_i))$ denote the weight of the longest path from n_s to n_i in the augmented C_h (C_v). $L_h(n_i)$ $(L_v(n_i))$ can be determined by performing the single source longest path algorithm on the augmented C_h (C_v) in $O(m^2)$ time, where m is number of modules. The coordinate (X_i, Y_i) of a module b_i is given by $(L_h(n_i), L_v(n_i))$. Further, the coordinates of all modules are determined in the topological order in C_h (C_v). Since the respective width and height of the placement for the given TCG are $L_h(n_t)$ and $L_v(n_t)$, the area of the placement is given by $L_h(n_t)L_v(n_t)$. Since each module has a unique coordinate after packing, there exists a unique placement corresponding to any TCG.

IV. TCG FOR SLICEABLE RECTILINEAR MODULES

In this section, we first introduce necessary and sufficient conditions of feasible TCG for sliceable rectilinear modules. We then present the TCG packing algorithm for sliceable rectilinear modules. (We will present the TCG properties for nonsliceable rectilinear modules in Section VI.)

A. Feasible TCG

We have shown in Section III that there always exists a unique feasible placement corresponding to a TCG for rectangular modules. For rectilinear modules, we must also maintain their original shapes during placement. To identify feasible TCG for rectilinear modules, we introduce the concept of *transitive reduction edges* of TCG. An edge (n_i, n_j) is said to be a *reduction edge* if there does not exist another path from n_i to n_j, except the edge (n_i, n_j) itself; otherwise, it is a *closure edge*. For example, the edges (n_a, n_c), (n_b, n_c), (n_c, n_d), and (n_d, n_e) in C_v of Fig. 2(b) are reduction edges while (n_a, n_d), (n_a, n_e), (n_b, n_d), and (n_b, n_e) are closure ones. (Note it is clear later that both reduction and closure edges are essential for maintaining a feasible TCG for perturbation. We shall also note that a key contribution of TCG lies in the first *general* graph representation with the feasibility guarantee during perturbations.)

For sliceable rectilinear modules, each zone contains exactly one submodule. Therefore, given a sliceable rectilinear module b_b with p rectangular submodules b_{b_i}, $i = 1, \ldots, p$, by slicing b_b from left to right (or from bottom to top) along vertical (hor-

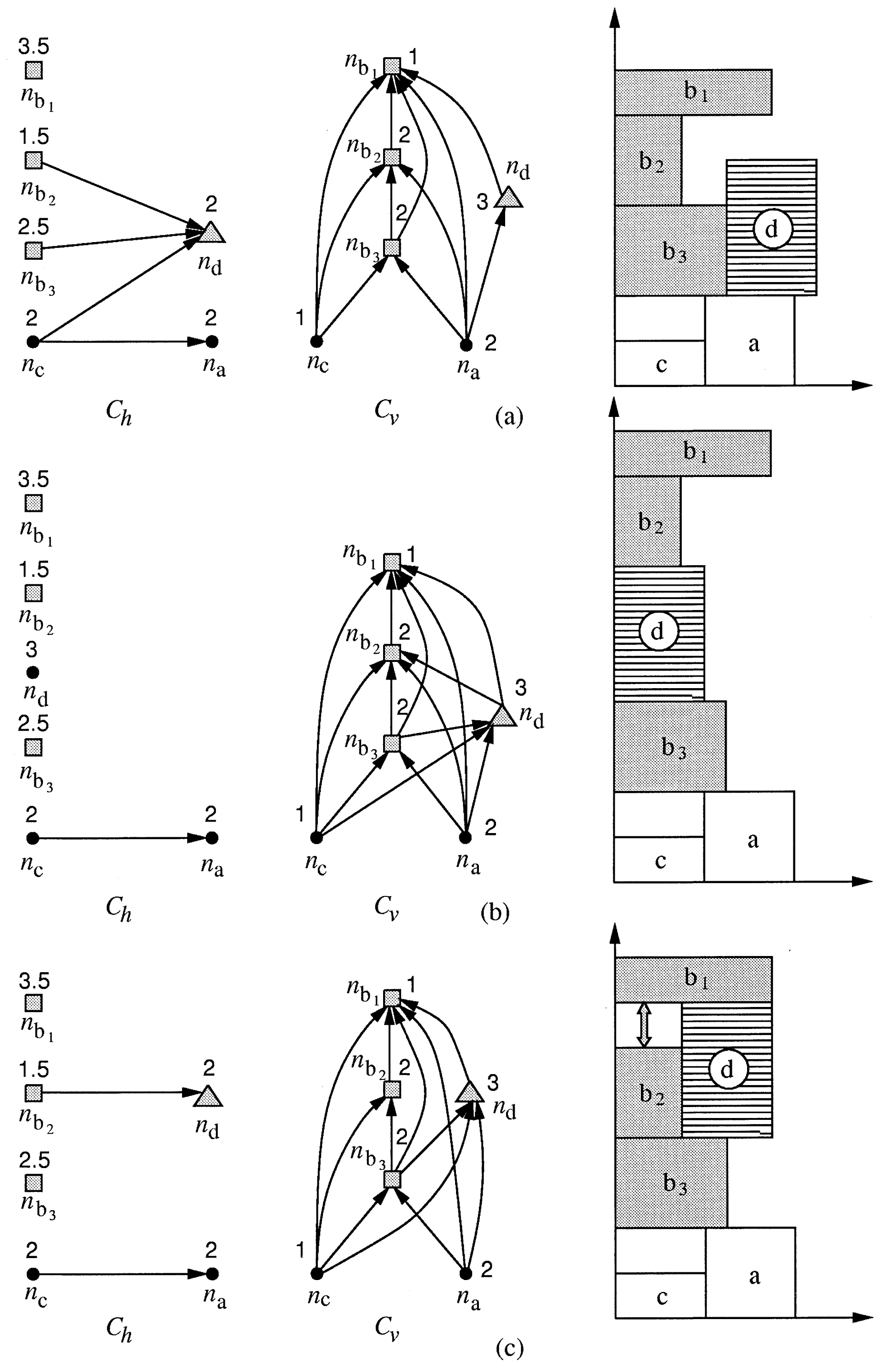

Fig. 4. (a) A feasible TCG for a rectilinear module b_b and its corresponding placement. (b) A TCG that violates the inseparability constraint and its corresponding placement. (c) Expanding submodule b_{b_2} to abut with b_{b_1}.

izontal) boundaries, we can construct a set of reduction edges $(n_{b_j}, n_{b_{j+1}}), j = 1, \dots, p-1$, and corresponding closure edges in C_h (C_v) since $b_{b_j} \vdash b_{b_{j+1}}$ $(b_{b_j} \perp b_{b_{j+1}}), j = 1, \dots, p-1$. (See Fig. 3(i) and (j) for an illustration.) To maintain the shape of a rectilinear module, we must treat the set of reduction and closure edges as a whole, and keep the edges $(n_{b_j}, n_{b_{j+1}}), j = 1, \dots, p-1$, as the reduction ones during processing (i.e., a reduction edge is not allowed to be changed into a closure one). Therefore, the TCG for rectilinear modules must satisfy the following constraint.

- *Inseparability Constraint*: For vertical (horizontal) slicing, the set of reduction and closure edges for a rectilinear module must be all in C_h (C_v) (i.e., there exists no edge between nodes n_{b_i}s in C_v (C_h)). Further, every edge $(n_{b_j}, n_{b_{j+1}}), j = 1, \dots, p-1$, remains as a reduction one.

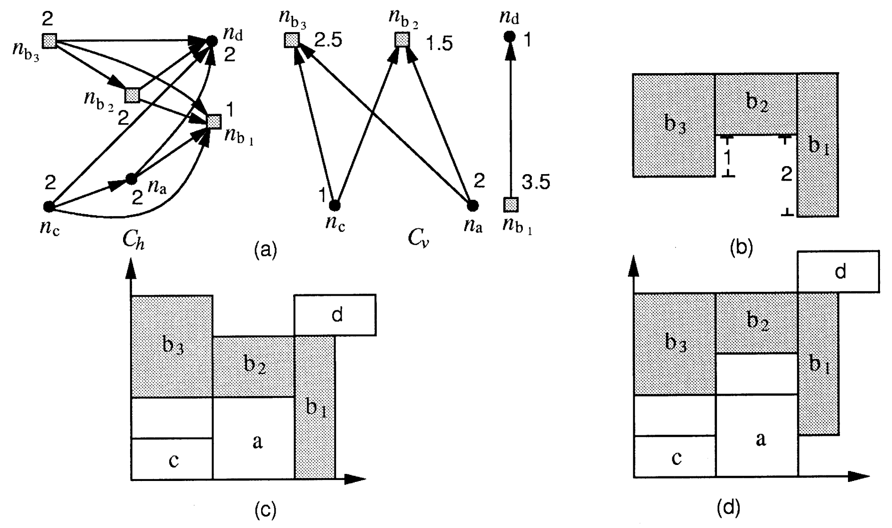

Fig. 5. (a) A TCG. (b) The rectilinear module b_b consists of three submodules b_{b_1}, b_{b_2}, and b_{b_3}. (c) An incorrect packing resulting from the original packing procedure. (d) A correct packing. (Note that this packing is suboptimal; we will show in Section V how to perturb the TCG to obtain an optimal packing.)

Fig. 3(a)–(h) show eight possible situations of rectilinear module b_b after rotation and flip. As shown in Fig. 3(a)–(d) [(e)–(h)], b_{b_1}, b_{b_2} and b_{b_3} are submodules of b_b, obtained by slicing along its vertical (horizontal) boundaries. For the situations shown in Fig. 3(a)–(d) [(e)–(h)], the corresponding TCG is illustrated in Fig. 3(i) [(j)].

The inseparability constraint will be violated if any reduction edge $(n_{b_j}, n_{b_{j+1}})$ becomes a closure edge (i.e., there exists another path $\langle n_{b_j}, n_x, \ldots, n_y, n_{b_{j+1}}\rangle$ from n_{b_j} to $n_{b_{j+1}}$). Fig. 4(a) shows a feasible TCG for the rectilinear module b_b with the submodules $b_{b_1}, b_{b_2}, b_{b_3}$ and its corresponding placement. In contrast, the TCG shown in Fig. 4(b) violates the inseparability constraint, in which the edge (n_{b_3}, n_{b_2}) in C_v becomes a closure one since there exists another path $\langle n_{b_3}, n_d, n_{b_2}\rangle$ from n_{b_3} to n_{b_2}. In this case, the rectilinear module b_b is divided into two since the submodules b_{b_2} and b_{b_3} are interleaved with another module b_d, resulting in an illegal placement.

For the TCG shown in Fig. 4(c), there exist two paths $\langle n_{b_3}, n_d, n_{b_1}\rangle$ and $\langle n_{b_3}, n_{b_2}, n_{b_1}\rangle$ from n_{b_3} to n_{b_1} in C_v and $H_d > H_{b_2}$. b_b will be divided into two pieces because b_d and b_{b_2} are inserted between b_{b_1} and b_{b_3} simultaneously, and H_d is larger than H_{b_2}. As in [18] and [22], this can be resolved by expanding b_{b_2} to connect with b_{b_1} because it does not cause problems for some practical applications but waste some silicon areas.

B. Packing

To maintain the shape of a rectilinear module without resorting to post processing, we must also modify the packing algorithm for rectangular modules described in Section III. Fig. 5 illustrates the difference between the packings for a rectangular and a rectilinear modules. Fig. 5(a) shows a given TCG with four rectilinear modules, b_a, b_b, b_c, and b_d, where b_b is a nonrectangular module whose shape is illustrated in Fig. 5(b). Fig. 5(c) and (d) shows an incorrect packing resulting from the original packing algorithm and a correct packing, respectively.

To make a packing for a rectilinear module correct, the coordinate of its submodule must be determined not only by the longest path from the source of the induced TCG, but also by the relative positions to the other submodules of the same module. Let $r(b_i, b_j)$ denote the relative difference of the positions between the submodules b_{b_i} and b_{b_j}; $r(b_i, b_j) = X_{b_j} - X_{b_i}$ $(r(b_i, b_j) = Y_{b_j} - Y_{b_i})$ for horizontal (vertical) slicing. For the example shown in Fig. 5(b), $r(b_1, b_2) = 2$, $r(b_1, b_3) = 1$, $r(b_2, b_1) = -2$, $r(b_2, b_3) = -1$, $r(b_3, b_1) = -1$, and $r(b_3, b_2) = 1$. Suppose we have packed b_{b_3} at $Y_{b_3} = 2$. To keep the relative positions between submodules, Y_{b_2} (Y_{b_1}) must equal 3 (1) since $Y_{b_3} + r(b_3, b_2) = 2 + 1 = 3$ $(Y_{b_3} + r(b_3, b_1) = 1)$. Therefore, Fig. 5(d) gives a correct packing while Fig. 5(c) does not. As mentioned in Section III, all modules should be packed in the topological order in C_h (C_v). To process a nonrectangular module, further, the ancestors of the nodes associated with its submodules should have all been processed, and the descendants of those nodes should not be processed until the coordinates of all the submodules have been determined. For example, according to the C_h (C_v) of Fig. 5(a), we should determine the Y coordinates of modules b_a and b_c before processing the nonrectangular module b_b, and module b_d should not be processed until all Y_{b_i}s, $i = 1, \ldots, 3$, are determined. In contrast, if we determine the Y coordinates in the order Y_{b_1}, Y_d, Y_a, Y_c, Y_{b_3}, and Y_{b_2}, we may need to adjust Y_d if the submodule b_{b_1} cannot be packed at Y_{b_1}.

To obtain the packing order of submodules, we modify the augmented C_h and C_v before applying the topological sort algorithm to find the ordering. Let the *fan-in* (*fan-out*) of a node n_i, denoted by $F_{\text{in}}(n_i)$ $(F_{\text{out}}(n_i))$, be the nodes n_js with edges (n_j, n_i) $((n_i, n_j))$. Given a rectilinear module b_b with submodules $b_{b_i}, i = 1, \ldots, p$, by slicing b_b along its vertical (horizontal) boundaries, we introduce additional edges into the augmented C_h (C_v) to make $F_{\text{in}}(n_{b_j}) = F_{\text{in}}(n_{b_{j+1}}), j = 1, \ldots, p-1$, except the edges emanating from n_s. [See Fig. 6(b).] After determining a topological order for the nodes in the new augmented C_h (C_v), we remove the added edges and then compute the X (Y) coordinates of the modules in the topological order. We associate each node n_i a c-value, c_i, to book-keep the X (Y) coordinate of the module/submodule i during the computation on the

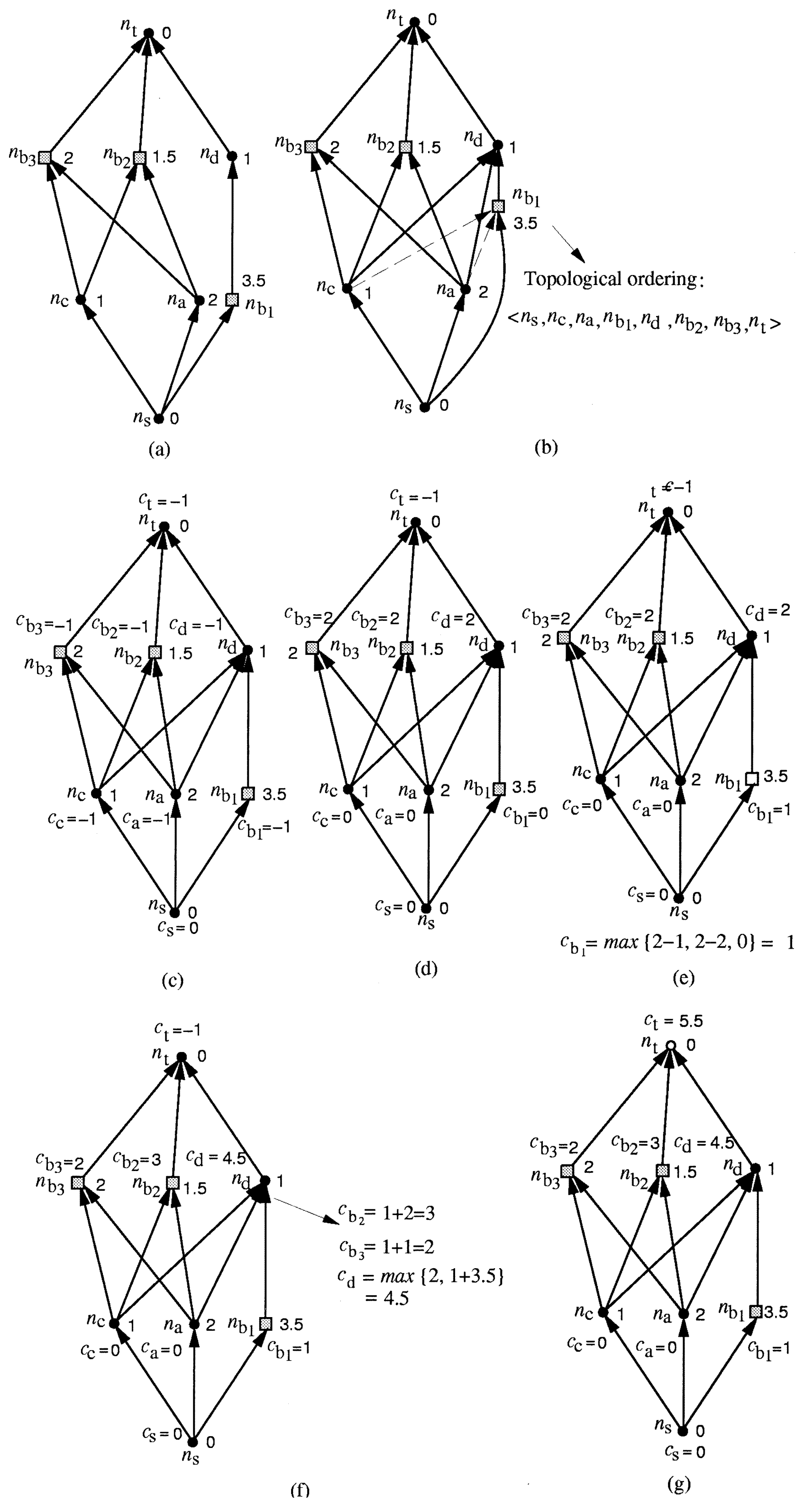
(a)
(b)
Topological ordering:
$<n_s, n_c, n_a, n_{b_1}, n_d, n_{b_2}, n_{b_3}, n_t>$
(c)
(d)
(e)
$c_{b_1} = max\{2-1, 2-2, 0\} = 1$
(f)
$c_{b_2} = 1+2=3$
$c_{b_3} = 1+1=2$
$c_d = max\{2, 1+3.5\} = 4.5$
(g)
$c_t = 5.5$

Fig. 6. An example computation of Y coordinates. (a) The augmented C_v for the C_v shown in Fig. 5(a). (b) The new augmented C_v after adding the edges (n_c, n_{b_1}) and (n_a, n_{b_1}) to make $F_{\text{in}}(n_{b_1}) = F_{\text{in}}(n_{b_2}) = F_{\text{in}}(n_{b_3})$. A possible topological order in the new augmented C_v is given as $\langle n_s, n_c, n_a, n_{b_1}, n_d, n_{b_2}, n_{b_3}, n_t \rangle$. (c) The c-values of the initial configuration. (d) The c-values after relaxing the nodes n_s, n_c, and n_a. (e) Before relaxing n_{b_1}, we need to compute c_{b_1}, c_{b_2}, and c_{b_3}. $c_{b_1} = \max\{c_{b_2} + r(b_2, b_1), c_{b_3} + r(b_3, b_1), c_{b_1}\} = 1$. (f) $c_{b_2} = c_{b_1} + r(b_1, b_2) = 3$ and $c_{b_3} = c_{b_1} + r(b_1, b_3) = 2$. (g) The final configuration after relaxing $n_{b_1}, n_d, n_{b_2}, n_{b_3}$, and n_t.

augmented C_h (C_v). The coordinates of the modules/submodules are computed as follows. For each node n_i in the augmented graph, we make $c_s = 0$ for the source n_s and $c_i = -1$ for any other node n_i. We then relax the c-value of a node n_i in the augmented C_h (C_v) as follows. For each node $n_j \in F_{\text{out}}(n_i)$, we make $c_j = \max\{c_j, c_i + W_i\}$ $(c_j = \max\{c_j, c_i + H_i\})$. However, when any node n_{b_i} of a submodule b_{b_i} in a rectilinear module b_b is encountered, the location of the submodule is given by $c_{b_i} = \max\{c_{b_i}, \max_{j=1,\ldots,p, j\neq i}\{c_{b_j} + r(b_j, b_i)\}\}$. Then, the c-values of other submodules b_{b_j}s, $j = 1, \ldots, p$ and $j \neq i$, are given by $c_{b_j} = c_{b_i} + r(b_i, b_j)$ according to the relative positions between submodules b_{b_j} and b_{b_i} before relaxing n_{b_i}.

Fig. 6(a) shows the augmented C_v for the C_v shown in Fig. 5(a). We first add the edges (n_a, n_{b_1}) and (n_c, n_{b_1}) to the augmented C_v to make $F_{\text{in}}(n_{b_1}) = F_{\text{in}}(n_{b_2}) = F_{\text{in}}(n_{b_3})$. [See Fig. 6(b).] (Note that we do not add edges (n_s, n_{b_2}) and (n_s, n_{b_3}) since they source from n_s.) After obtaining a topological order, say $\langle n_s, n_c, n_a, n_{b_1}, n_d, n_{b_2}, n_{b_3}, n_t\rangle$, of the new augmented C_v, we remove the added edges and start to compute the Y coordinates for all nodes based on the topological order. [See Fig. 6(c)–(g).] Fig. 6(c) shows the initial configuration, in which all nodes have the c-value -1 except that $c_s = 0$. Fig. 6(d) illustrates the configuration after relaxing the nodes n_s, n_c, and n_a. We then start to process n_{b_1}. Since n_{b_1} corresponds to a submodule of the rectilinear module b_b first encountered and there exists no edge among nodes n_{b_1}, n_{b_2}, and n_{b_3}, we shall first determine c_{b_1} [see Fig. 6(e)] and then compute c_{b_2} and c_{b_3} [see Fig. 6(f)] before relaxing n_{b_1} to maintain the relative positions of the rectilinear module b_b. Here, $c_{b_1} = \max\{c_{b_2} + r(b_2, b_1), c_{b_3} + r(b_3, b_1), c_{b_1}\} = \max\{2-2, 2-1, 0\} = 1$, $c_{b_2} = c_{b_1} + r(b_1, b_2) = 1 + 2 = 3$, and $c_{b_3} = c_{b_1} + r(b_1, b_3) = 1 + 1 = 2$. We then relax the node n_{b_1}, resulting in $c_d = \max\{c_d, c_{b_1} + H_{b_1}\} = \max\{2, 1+3.5 = 4.5\}$. The relaxation process continues for the nodes n_d, n_{b_2}, n_{b_3}, and finally n_t, resulting in the final configuration shown in Fig. 6(g), in which all Y coordinates have been determined.

Theorem 2: Given a feasible TCG for a sliceable rectilinear module, the packing scheme proposed above gives a feasible placement in $O(m'^2)$ time, where m' is the number of rectangular modules and submodules.

Proof: It was shown in [11] that TCG gives a feasible placement for rectangular modules if we apply a longest algorithm on TCG to compute the x (y) coordinates of the modules based on their topological ordering in C_h (C_v). Here, we need to guarantee that all submodules in each rectilinear module will abut and the relative positions between submodules are maintained after packing. Before applying the topological sorting algorithm, we modify the TCG to make $F_{\text{in}}(n_{b_j}) = F_{\text{in}}(n_{b_{j+1}})$, $j = 1, \ldots, p-1$, for each rectilinear module b_b with submodules b_{b_i}, $i = 1, \ldots, p$, by adding edges into the augmented C_v (C_h). We claim that 1) the resulting graph is directed acyclic after adding the additional edges; and 2) a topological order obtained from the new graph is also one in the original graph.

Let n_{b_p} and n_{b_q} denote two nodes in the resulting graph, which are associated with two submodules of the rectilinear module b_b. If there exist no edge between n_{b_p} and n_{b_q}, we add an additional edge (n_k, n_{b_q}) to the graph, where n_k is the predecessor of n_{b_p}. Assume that the resulting graph is cyclic. There must exist a path $\langle n_{b_q}, \ldots, n_k\rangle$ from n_{b_q} to n_k in the original graph. This implies that there exists a path $\langle n_{b_q}, \ldots, n_k, n_{b_p}\rangle$. Since there exists a path from n_{b_q} to n_{b_p} in the original transitive closure graph, there must exist an edge between n_{b_q} to n_{b_p}, contradicting the assumption that there exists no edge between n_{b_p} and n_{b_q}. Since we only add edges (without deleting any edge) in the original graph and the new graph is still directed acyclic, it is clear that a topological order obtained from the new graph is also one in the original graph.

Now, we show that the relative positions between submodules of each rectilinear module can be maintained. Without loss of generality, given a rectilinear module b_b with submodules b_{b_i}, $i = 1, \ldots, p$, obtained by slicing along its vertical boundaries, we show that the vertical relative positions between b_{b_i}s will be kept. By our procedure, the y coordinates of nodes n_ks, predecessors of n_{b_i}s, have been determined before the coordinate of b_b is computed. The y coordinates (c_{b_i}s) of n_{b_i}s are known by relaxing n_ks. We can compute the coordinates of a submodule b_{b_i} from submodules b_{b_j}s, $j = 1, \ldots, p$, if b_{b_j}s are placed at c_{b_i}s, and choose the maximum value as the coordinate of b_{b_i}. After the location of b_{b_i} is determined, the coordinates of b_{b_j}s, $j = 1, \ldots, p$ and $i \neq j$, can be computed by their relative positions to the submodule b_{b_i}. Therefore, the vertical relative positions among b_{b_i}s are maintained. Similar claims hold for the rectilinear module sliced along horizontal boundaries.

Since a (simple) graph can have at most $O(m'^2)$ edges, the time complexity of adding additional edges is also $O(m'^2)$. Besides, it takes $O(m'^2)$ time to compute a longest path on an (acyclic) constraint graph. Therefore, the time complexity of our packing scheme is $O(m'^2)$ time. ■

It should be noted that the SP-based packing scheme presented in [2] needs $O(m'^3)$ time.

V. Algorithm

Our algorithm is based on simulated annealing [8]. Given an initial solution represented by a TCG, we perturb the TCG to obtain a new TCG. The perturbation continues to search for a "good" configuration until a predefined termination condition is satisfied. To ensure the correctness of rectilinear module packing, the new TCG for each rectilinear module must satisfy the TCG feasibility conditions described in Section III and the inseparability constraint presented in Section IV-A. To identify a feasible TCG for perturbation, we need to identify reduction edges.

A. Reduction Edge Identification

Since TCG is formed by directed acyclic transitive closure graphs, given an arbitrary node n_i in one transitive closure graph, there exists at least one reduction edge (n_i, n_j), where $n_j \in F_{\text{out}}(n_i)$. For nodes $n_k, n_l \in F_{\text{out}}(n_i)$, the edge (n_i, n_k) cannot be a reduction edge if $n_k \in F_{\text{out}}(n_l)$. Hence, we remove those nodes in $F_{\text{out}}(n_i)$ that are fan-outs of others. The edges between n_i and the remaining nodes in $F_{\text{out}}(n_i)$ are reduction edges. For the C_h of Fig. 5(a), $F_{\text{out}}(n_c) = \{n_a, n_{b_1}, n_d\}$. Since n_{b_1} and n_d belong to $F_{\text{out}}(n_a)$, edges (n_c, n_{b_1}) and (n_c, n_d) are closure edges while (n_c, n_a) is a reduction one. The time complexity for finding such a reduction edge is $O(m'^2)$, where m' is the number of rectangular modules and submodules [11].

B. *Solution Perturbation*

We apply the following eight operations to perturb a TCG.

- *Rotation:* Rotate a rectangular module.
- *Swap:* Swap two nodes associated with rectangular modules in both C_h and C_v.
- *Reverse:* Reverse a *reduction edge* in C_h or C_v.
- *Move:* Move a *reduction edge* from one transitive closure graph (C_h or C_v) to the other.
- *Transpositional Move:* Move a *reduction edge* from one transitive closure graph (C_h or C_v) to the other and then transpose the two nodes associated with the edge. It is clear later that this operation is different from performing Move and then Reverse.
- *Perpendicular Flip:* Flip a rectilinear module about the axis perpendicular to the cut lines for obtaining its submodules.
- *Parallel Flip:* Flip a rectilinear module about the axis parallel to its cut lines.
- *Twirl:* Rotate a rectilinear module.

Note that Rotation, Swap, Reverse, and Move are first introduced in [11], which can be performed in respective $O(1)$, $O(1)$, $O(m'^2)$, and $O(m'^2)$ times, where m' is the number of modules and submodules. Further, the resulting graph after performing any of these operations on a TCG is still a TCG.

Rotation, Swap, Perpendicular Flip, and Parallel Flip will not change the topology of TCG, and thus the inseparability constraint will not be violated, either. However, Reverse, Move, Transpositional Move, and Twirl will, and we may need to update TCG after performing Reverse, Move, Transpositional Move, and Twirl. Further, in order to guarantee that the inseparability constraint will not be violated, we need feasibility detection during the operation. We first detail the operations in the following.

1) Rotation: To rotate a *rectangular module* b_i, we only need to exchange the weights of the corresponding node n_i in C_h and C_v.

Lemma 1: The inseparability constraint of a TCG will not be violated for the Rotation operation.

Proof: Since the configuration of a TCG remains the same for the Rotation operation, the inseparability constraint will not be violated. ■

2) Swap: To swap nodes n_i and n_j of two rectangular modules b_i and b_j, we only need to exchange the nodes n_i and n_j in both C_h and C_v.

Lemma 2: The inseparability constraint of a TCG will not be violated for the Swap operation.

Proof: Since we only exchange the nodes of rectangular modules and the topology of the TCG remains the same after the Swap operation, the inseparability constraint will not be violated. ■

3) Reverse: The Reverse operation reverses the direction of a *reduction* edge (n_i, n_j) in a transitive closure graph, where n_i and n_j are not associated with two submodules of the same rectilinear module. For two modules b_i and b_j, $b_i \vdash b_j$ $(b_i \perp b_j)$ if there exists a reduction edge (n_i, n_j) in C_h (C_v); after reversing the edge (n_i, n_j), we have the new geometric relation $b_j \vdash b_i$ $(b_j \perp b_i)$.

To reverse a reduction edge (n_i, n_j) in a transitive closure graph, we first delete the edge from the graph, and then add the edge (n_j, n_i) to the graph. For each node $n_k \in F_{\text{in}}(n_j) \cup \{n_j\}$ and $n_l \in F_{\text{out}}(n_i) \cup \{n_i\}$ in the new graph, we shall check whether the edge (n_k, n_l) exists in the new graph. If the graph contains the edge, we do nothing; otherwise, we need to add the edge to the graph and delete the corresponding edge (n_k, n_l) or (n_l, n_k) in the other transitive closure graph, if any, to maintain the properties of the TCG.

4) Move: The Move operation moves a *reduction* edge (n_i, n_j) in a transitive closure graph to the other, where n_i and n_j are not associated with two submodules of the same rectilinear module. Move switches the geometric relation of the two modules b_i and b_j between a horizontal relation and a vertical one. For two modules b_i and b_j, $b_i \vdash b_j$ $(b_i \perp b_j)$ if there exists a reduction edge (n_i, n_j) in C_h (C_v); after moving the edge (n_i, n_j) to C_v (C_h), we have the new geometric relation $b_i \perp b_j$ $(b_i \vdash b_j)$.

To move a reduction edge (n_i, n_j) from a transitive closure graph G to the other G' in a TCG, we first delete (n_i, n_j) from G and then add (n_i, n_j) to G'. Similar to the Reverse operation, for each node $n_k \in F_{\text{in}}(n_i) \cup \{n_i\}$ and $n_l \in F_{\text{out}}(n_j) \cup \{n_j\}$, we shall check whether the edge (n_k, n_l) exists in G'. If G' contains the edge, we do nothing; otherwise, we need to add the edge to G' and delete the corresponding edge (n_k, n_l) or (n_l, n_k) in G, if any, to maintain the properties of the TCG.

5) Transpositional Move: The Transpositional Move operation removes a *reduction* edge (n_i, n_j) from a transitive closure graph, and adds an edge (n_j, n_i) to the other, where n_i and n_j are not associated with two submodules of the same rectilinear module. Transpositional Move switches the geometric relation of the two modules b_i and b_j between a horizontal relation and a vertical one and changes the ordering of the two modules b_i and b_j in their geometric relation. For two modules b_i and b_j, $b_i \vdash b_j$ $(b_i \perp b_j)$ if there exists a reduction edge (n_i, n_j) in C_h (C_v); after transpositionally moving the edge (n_i, n_j) to C_v (C_h), we have the new geometric relation $b_j \perp b_i$ $(b_j \vdash b_i)$.

To transpositionally move a reduction edge (n_i, n_j) from a transitive closure graph G to the other G' in a TCG, we first delete (n_i, n_j) from G and add (n_j, n_i) to G'. Similar to the Move operation, for each node $n_k \in F_{\text{in}}(n_j) \cup \{n_j\}$ and $n_l \in F_{\text{out}}(n_i) \cup \{n_i\}$, we shall check whether the edge (n_k, n_l) exists in G'. If G' contains the edge, we do nothing; otherwise, we need to add the edge to G' and delete the corresponding edge (n_k, n_l) or (n_l, n_k) in G, if any, to maintain the properties of the TCG. We have the following theorem.

Theorem 3: TCG is closed under the Transpositional Move operation, and such an operation takes $O(m'^2)$ time, where m' is the number of modules and submodules.

Proof: We first show that the resulting graphs C_h and C_v of a TCG satisfy the three properties of TCG after performing the Transpositional Move operation.

Without loss of generality, we focus on the case for transpositionally moving a reduction edge (n_i, n_j) from C_h to C_v. For Property 1, suppose that the resulting C_v is not acyclic after we delete a reduction edge (n_i, n_j) from C_h and add an edge (n_j, n_i) to C_v. There must exist a path from n_i to n_j in the original C_v. This implies that the edge (n_i, n_j) is also in the original

C_v since C_v is a transitive closure graph. This is a contradiction since (n_i, n_j) cannot both exist in the original TCG (Property 2). Therefore, the new C_v must be acyclic. The new C_h must also be acyclic since we do not add any edge into the original C_h. For Property 2, each pair of nodes must be connected by exactly one edge either in the new C_h or in the new C_v after the operation because the corresponding edge will be deleted from C_h after the edge (n_j, n_i) is added to C_v. For Property 3, suppose that the new C_v is not a transitive closure of itself. Then, there exists a path $\langle n_x, \ldots, n_j, n_i, \ldots, n_y \rangle$ in the new C_v, but the C_v does not contain the closure edge (n_x, n_y). During the operation, for each node $n_k \in F_{\text{in}}(n_j) \cup \{n_j\}$ and $n_l \in F_{\text{out}}(n_i) \cup \{n_i\}$ in C_v, we add the edges (n_k, n_l)s to the new C_v and delete them from C_h. Therefore, at least one of the edges (n_x, n_j) and (n_i, n_y) does not exist in the original C_h; otherwise, we would have added the closure edge (n_x, n_y) into the new C_v during the Transpositional Move operation. This implies that the original C_v is not a transitive closure graph, contradicting to our assumption. It is clear that the deleted edges of C_h are closure edges of the new C_v, which cannot be the closure edges in C_h. Therefore, the new C_h is still a transitive closure graph of itself.

The time complexity is dominated by checking whether the edges (n_k, n_l)s ($n_k \in F_{\text{in}}(n_j) \cup \{n_j\}$ and $n_l \in F_{\text{out}}(n_i) \cup \{n_i\}$) exist in the new graph and by inserting and deleting the corresponding edges. Since there are at most $O(m')$ n_ks and $O(m')$ n_ls, the operation takes $O(m'^2)$ time in total. ■

6) Perpendicular Flip: The Perpendicular Flip operation flips a rectilinear module b_b about the axis perpendicular to the cut lines for obtaining its submodules by changing the difference of relative positions from $r(b_i, b_j)$ to $W_{b_i} - W_{b_j} - r(b_i, b_j)$ ($H_{b_i} - H_{b_j} - r(b_i, b_j)$) for horizontal (vertical) cut lines, where $i, j = 1, \ldots, p$ and $j \neq i$. Fig. 7(b) shows the resulting C_h, C_v, and placement after perpendicularly flipping the rectilinear module b_b. $r(b_1, b_3) = 1$, $r(b_1, b_2) = 2$, $r(b_2, b_1) = -2$, $r(b_2, b_3) = -1$, $r(b_3, b_1) = -1$, and $r(b_3, b_2) = 1$ in Fig. 7(b) while $r(b_1, b_3) = r(b_1, b_2) = r(b_2, b_1) = r(b_2, b_3) = r(b_3, b_1) = r(b_3, b_2) = 0$ in Fig. 7(a).

Since Perpendicular Flip changes only the relative positions of each pair of submodules, the topology of TCG remains the same after the operation. We have the following theorem and lemma.

Theorem 4: TCG is closed under the Perpendicular Flip operation, and such an operation takes $O(p^2)$ time, where p is the number of submodules in the rectilinear module.

Proof: Since the Perpendicular Flip operation only exchanges the differences of relative positions among submodules in a rectilinear module without changing the configuration of TCG, the resulting graphs are still a TCG.

It takes $O(1)$ time to exchange the value for the relative position between two submodules; therefore, we need $O(p^2)$ time to update these values for each pair of submodules if there are p submodules in a rectilinear module. ■

Lemma 3: The inseparability constraint of a TCG will not be violated for Perpendicular Flip operation.

Proof: Since the configuration of TCG remains the same after the Perpendicular Flip operation, the inseparability constraint will not be violated. ■

7) Parallel Flip: The Parallel Flip operation flips a rectilinear module b_b about the axis parallel to the cut lines for obtaining its submodules by swapping the nodes n_{b_k} and $n_{b_{p-k+1}}$, $k = 1, \ldots, \lfloor p/2 \rfloor$, in both C_h and C_v. Given a rectilinear module b_b consisting of submodules b_{b_i}, $i = 1, \ldots, p$, formed by vertical (horizontal) cuts (i.e., $b_{b_j} \vdash b_{b_{j+1}}$ $(b_{b_j} \perp b_{b_{j+1}})$, there exist reduction edges $(n_{b_j}, n_{b_{j+1}})$ and their corresponding closure edges in C_h (C_v), where $j = 1, \ldots, p-1$. After swapping the nodes n_{b_k} and $n_{b_{p-k+1}}$, $k = 1, \ldots, \lfloor p/2 \rfloor$, in both of C_h and C_v, we have the new reduction edges $(n_{b_l}, n_{b_{l-1}})$ and their corresponding closure edges in C_h (C_v), implying $b_{b_l} \vdash b_{b_{l-1}}$ $(b_{b_l} \perp b_{b_{l-1}})$, where $l = p, \ldots, 2$.

Fig. 7(c) shows the resulting C_h, C_v, and placement after parallel flipping the rectilinear module b_b of Fig. 7(b). Notice that after swapping the nodes n_{b_1} and n_{b_3} in both C_h and C_v of Fig. 7(b), we introduce the new edges (n_{b_1}, n_{b_2}), (n_{b_2}, n_{b_3}), and (n_{b_1}, n_{b_3}) in C_v of Fig. 7(c), implying $b_1 \perp b_2$ and $b_2 \perp b_3$.

Parallel Flip needs to perform the Swap operation $\lfloor p/2 \rfloor$ times, where p is the number of submodules in a rectilinear module, and each Swap operation guarantees to perturb into a unique feasible TCG in $O(1)$ time. We have the following theorem and lemma.

Theorem 5: TCG is closed under the Parallel Flip operation, and such an operation takes $O(p)$ time, where p is the number of submodules in the rectilinear module.

Proof: Since the Parallel Flip operation only reverses the sequence of the nodes n_{b_j}, $j = 1, \ldots, p$, of a rectilinear module b_b in both C_h and C_v without changing the topology of TCG, the resulting graphs are still a TCG.

The Parallel Flip operation has to perform the Swap operation for each pair of the symmetric nodes n_{b_j} and $n_{b_{p-k+1}}$ in both C_h and C_v, $k = 1, \ldots, \lfloor p/2 \rfloor$, and a Swap operation takes $O(1)$ time [11]. There are $\lfloor p/2 \rfloor$ pairs of nodes; there fore, the time complexity is $O(p)$. ■

Lemma 4: The inseparability constraint of a TCG will not be violated in Parallel Flip operation.

Proof: The Parallel Flip operation only reverses the sequence of nodes n_{b_j}, $j = 1, \ldots, p$, of a rectilinear module b_b without changing the topology of a TCG, the inseparability constraint will not be violated. ■

8) Twirl: The Twirl operation rotates a rectilinear module b_b by exchanging the weights of the corresponding nodes n_{b_i}, $i = 1, \ldots, p$, in C_h and C_v, and transpositionally moving the reduction edges $(n_{b_j}, n_{b_{j+1}})$, $j = 1, \ldots, p-1$, from a transitive closure graph to the other. Given a rectilinear module b_b consisting of submodules b_{b_i} obtained by vertical (horizontal) cuts (i.e., $b_{b_j} \vdash b_{b_{j+1}}$ $(b_{b_j} \perp b_{b_{j+1}})$); after exchanging the weights and transpositionally moving the reduction edges, we rotate the rectangular submodules b_{b_i}, $i = 1, \ldots, p$, and make $b_{b_l} \perp b_{b_{l-1}}$ $(b_{b_l} \vdash b_{b_{l-1}})$, $l = p, \ldots, 2$.

Fig. 7(d) shows the resulting C_h, C_v, and placement after twirling the b_b of Fig. 7(c). To transpositionally move the reduction edge (n_{b_1}, n_{b_2}) from C_v to C_h, we first remove the reduction edge (n_{b_1}, n_{b_2}) from C_v, and add (n_{b_2}, n_{b_1}) to C_h. Since $\{n_{b_2}\} \cup F_{\text{in}}(n_{b_2}) = \{n_{b_2}\}$ and $\{n_{b_1}\} \cup F_{\text{out}}(n_{b_1}) = \{n_{b_1}\}$ in C_h, for each node $n_k \in \{n_{b_2}\}$ and $n_l \in \{n_{b_1}\}$, we shall check whether the edge (n_k, n_l) exists. Since the edge (n_{b_2}, n_{b_1}) is already added during the transpositional move, we

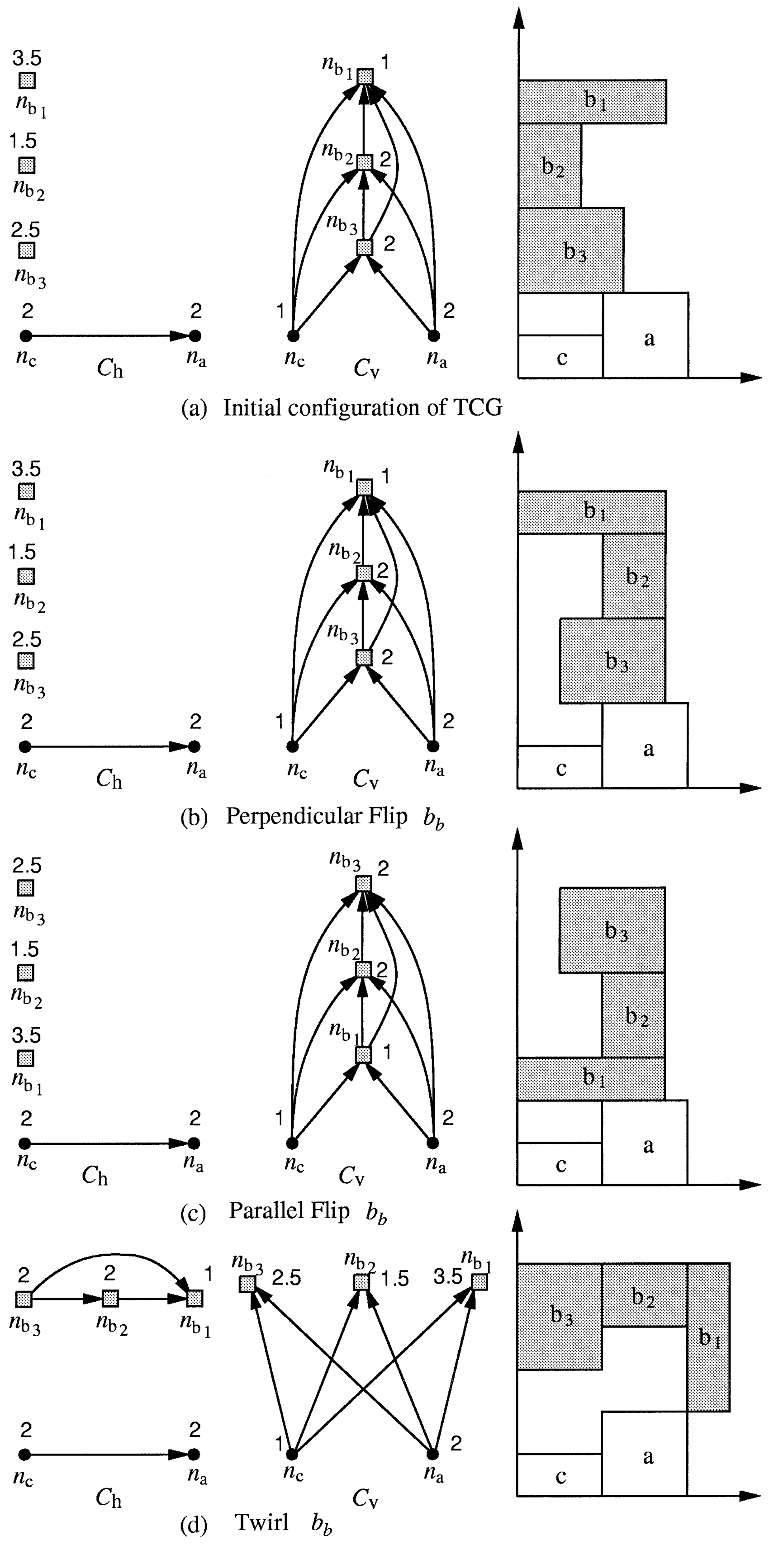

Fig. 7. Examples of perturbations. (a) The initial TCG (C_h and C_v) and its corresponding placement. (b) The resulting TCG and placement after perpendicularly flipping the rectilinear module b_b shown in (a). (c) The resulting TCG and placement after parallel flipping the rectilinear module b shown in (b). (d) The resulting TCG and placement after twirling the rectilinear module b by $-90°$ shown in (c).

do nothing. Similarly, to transpositionally move the reduction edge (n_{b_2}, n_{b_3}) from the new C_v to the new C_h, we remove the reduction edge (n_{b_2}, n_{b_3}) from the new C_v and add (n_{b_3}, n_{b_2}) to the new C_h. Since $\{n_{b_3}\} \cup F_{\text{in}}(n_{b_3}) = \{n_{b_3}\}$ and $\{n_{b_2}\} \cup F_{\text{out}}(n_{b_2}) = \{n_{b_1}, n_{b_2}\}$ in the new C_h and (n_{b_3}, n_{b_2}) is already added, we only need to add the edge (n_{b_3}, n_{b_1}). Finally, we exchange the weights of each node n_{b_i}, $i = 1, 2, 3$, in C_h and C_v of Fig. 7(c). Fig. 7(d) illustrates the resulting TCG.

The Twirl operation needs to perform Rotation and Transpositional Move p and $p-1$ times, and each Rotation and Transpositional Move operation guarantees to perturb into a feasible TCG in respective $O(1)$ and $O(m'^2)$ times, where p is the number of submodules in a rectilinear module and m' is the number of rectangular modules and submodules. We have the following theorem.

Theorem 6: TCG is closed under the Twirl operation, and such an operation takes $O(m'^2 p)$ time, where p is the number of submodules in a rectilinear module and m' is the number of rectangular modules and submodules.

Proof: By Lin and Chang [11] and Theorem 3, TCG is close under the Rotation and Transpositional Move operations. The resulting graphs are still a TCG after performing the Rotation and Transpositional Move operations p and $p-1$ times, respectively, where p is the number of sumbodules in the rectilinear module b_b. The time complexities of Rotation and Transpositional Move operations are $O(1)$ and $O(m'^2)$, respectively. Therefore, the time complexity of the Twirl operation is $O(m'^2 p)$. ■

C. Feasibility Detection

To maintain the shapes of rectilinear modules, TCG must satisfy the inseparability constraint for each rectilinear module. Among the eight operations, only Reverse, Move, Transpositional Move, and Twirl could violate the constraints, which can easily be detected during perturbation. When we move an edge (n_i, n_j) or reverse/transpositionally move (n_j, n_i), the inseparability constraint will be violated if $n_{b_l} \in F_{\text{in}}(n_i) \cup \{n_i\}$ and $n_{b_k} \in F_{\text{out}}(n_j) \cup \{n_j\}$, where $|l-k| = 1$ since (n_{b_l}, n_{b_k}) would become a closure edge. Since Twirl consists of Rotate and Transpositional Move operations, the inseparability constraint will not be violated if the inseparability constraint is satisfied for each Transpositional Move operation. By doing the feasibility detection during the Reverse, Move, or Transpositional Move operation, we can guarantee that the resulting TCG is still a TCG for rectilinear modules. We thus have the following theorem.

Theorem 7: The inseparability constraint of a TCG is not violated for the Reverse, Move, Transpositional Move, or Twirl operation with the feasibility detection.

Proof: The inseparability constraint will be violated if a reduction edge (n_{b_p}, n_{b_q}) is converted into a closure one. Without loss of generality, we focus on the case for reversing a reduction edge (n_i, n_j) in C_h. During the operation, we should check the existence of each edge (n_l, n_k) in C_h, where nodes $n_l \in F_{\text{in}}(n_j) \cup n_j$ and $n_k \in F_{\text{out}}(n_i) \cup n_i$. If such edge does not exist, we would add the edge to C_h and delete the corresponding edge (n_l, n_k) or (n_k, n_l) in C_v. Each newly added edge in C_h must be a closure edge; therefore, if nodes $n_l = n_{b_p}$ and $n_k = n_{b_q}$ for a reduction edge (n_{b_p}, n_{b_q}), the edge will become a closure edge (see the example of Fig. 4(b)). Similar claims hold for the Move and Transpositional Move operations.

The Twirl operation performs the Transpositional Move operation $p-1$ times on each edge $(n_{b_j}, n_{b_{j+1}})$, $j = 1, \ldots, p-1$, for a rectilinear module with p submodules. Therefore, if we can guarantee that the inseparability constraint of TCG will not be violated for each Transpositional Move operation, the inseparability constraint of the resulting TCG will not be violated. Besides, we also have to show that the inseparability constraint of b_b will be maintained during the Twirl operation. Without loss of generality, we transpositionally move the reduction edges $(n_{b_j}, n_{b_{j+1}})$s, $j = 1, \ldots, p-1$, from C_h to C_v in twirling a rectilinear module b_b. We first delete the edge (n_{b_1}, n_{b_2}) from C_h and add the edge (n_{b_2}, n_{b_1}) to C_v. Similarly, the edge (n_{b_2}, n_{b_3}) is deleted from C_h and a new edge (n_{b_3}, n_{b_2}) is added to C_v. Since $n_{b_3} \in F_{\text{in}}(n_{b_3}) \cup \{n_{b_3}\}$ and $n_{b_1} \in F_{\text{out}}(n_{b_2}) \cup \{n_{b_2}\}$, the edge (n_{b_3}, n_{b_1}) will be added to C_v after transpositionally moving the edge (n_{b_2}, n_{b_3}). (n_{b_2}, n_{b_1}) remains a reduction edge since all newly added edges that start from n_{b_3} or an ancestor of n_{b_3} cannot make (n_{b_2}, n_{b_1}) a closure one. By performing such operation $p-1$ times, the reduction edges $(n_{b_{j+1}}, n_{b_j})$s and the corresponding closure edges will be added to C_v and there exists no edge among nodes n_{b_j} and $n_{b_{j+1}}$ in C_v. Therefore, the inseparability constraint of a TCG is not violated for the Twirl operation with the feasibility detection. ■

Fig. 8(a) shows the resulting C_h, C_v, and placement after moving the edge (n_a, n_{b_1}) from C_v shown in Fig. 7(d) to C_h. Since $\{n_a\} \cap F_{\text{in}}(n_a) = \{n_a, n_c\}$ and $\{n_{b_1}\} \cap F_{\text{out}}(n_{b_1}) = \{n_{b_1}\}$ in C_h, we shall check the edge (n_c, n_{b_1}) for the inseparability constraint. The inseparability constraint will not be violated because b_c and b_{b_1} are not adjacent submodules of the same rectilinear module. Fig. 8(b) shows the resulting C_h, C_v, and placement after transpositionally moving the edge (n_a, n_{b_3}) from the C_v of Fig. 8(a) to C_h. Since $\{n_{b_3}\} \cap F_{\text{in}}(n_{b_3}) =$

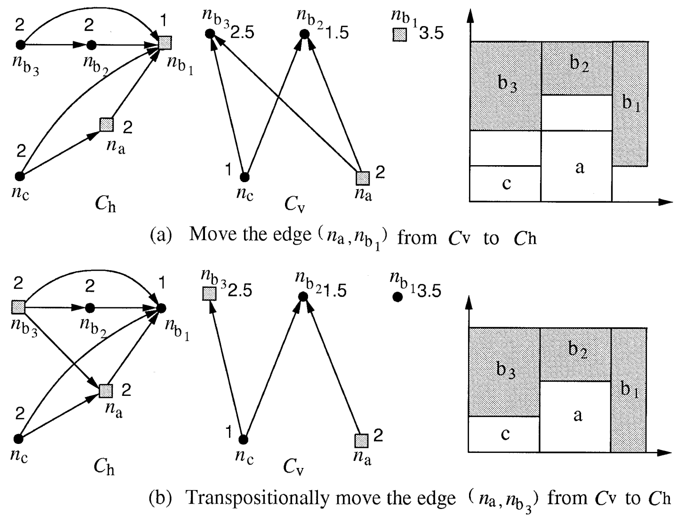
2
2
1
n_{b_3}
n_{b_2}
n_{b_1}
2
n_a
2
n_c
C_h
n_{b_3} 2.5
n_{b_2} 1.5
n_{b_1} 3.5
1
n_c
C_v
2
n_a
b_3
b_2
b_1
a
c
(a) Move the edge (n_a, n_{b_1}) from C_v to C_h
(b) Transpositionally move the edge (n_a, n_{b_3}) from C_v to C_h

Fig. 8. Examples of perturbations (continued from Fig. 7). (a) The resulting TCG and placement after moving the reduction edge (n_a, n_{b_1}) from the C_v of Fig. 7(d) to C_h. (b) The resulting TCG and placement after transpositionally moving the reduction edge (n_a, n_{b_3}) from the C_v shown in (e) to C_h.

```
Algorithm: Simulated_Annealing_Floorplanning($P, \epsilon, r, k$)
1 begin
2 $E \leftarrow$ TCG; /* initial solution */
3 $Best \leftarrow E$; $T \leftarrow \frac{\Delta_{avg}}{ln(P)}$; $N = km$;
4 repeat
5   $MT \leftarrow uphill \leftarrow reject \leftarrow 0$;
6   repeat
7     SelectMove($M$);
8     Case $M$ of
9       $M1$: $NE \leftarrow Rotation(E)$;
10      $M2$: $NE \leftarrow Swap(E)$;
11      $M3$: $NE \leftarrow Reverse(E)$;
12        if (Feasibility Detection(NE)==Fail)
13          Jump to Line 7;
14      $M4$: $NE \leftarrow Move(E)$;
15        if (Feasibility Detection(NE)==Fail)
16          Jump to Line 7;
17      $M5$: $NE \leftarrow TranspositionalMove(E)$;
18        if (Feasibility Detection(NE)==Fail)
19          Jump to Line 7;
20      $M6$: $NE \leftarrow PerpendicularFlip(E)$;
21      $M7$: $NE \leftarrow ParallelFlip(E)$;
22      $M8$: $NE \leftarrow Twirl(E)$;
23        if (Feasibility Detection(NE)==Fail)
24          Jump to Line 7;
25    $MT \leftarrow MT + 1$; $\Delta cost \leftarrow cost(NE) - cost(E)$;
26    if ($\Delta cost \leq 0$) or ($Random < e^{\frac{-\Delta cost}{T}}$)
27    then
28      if ($\Delta cost > 0$) then $uphill \leftarrow uphill + 1$;
29      $E \leftarrow NE$;
30      if $cost(E) < cost(Best)$ then $Best \leftarrow E$;
31    else $reject \leftarrow reject + 1$;
32  until ($uphill > N$) or ($MT > 2N$);
33  $T = rT$; /* reduce temperature */
34 until ($\frac{reject}{MT} > 0.95$) or ($T < \epsilon$);
35 end
```

Fig. 9. Floorplanning design algorithm for rectilinear modules using TCG.

$\{n_{b_3}\}$ and $\{n_a\} \cap F_{\text{out}}(n_a) = \{n_a, n_{b_1}\}$ in C_h, we shall check (n_{b_3}, n_{b_1}) for the constraint. Since b_{b_1} and b_{b_3} are not adjacent submodules, the inseparability constraint will not be violated.

D. Floorplan Design Algorithm

In this subsection, we describe our rectilinear module placement algorithm (see Fig. 9 for the pseudocode). Our algorithm is based on the simulated annealing method [8]. We first initialize a TCG as the current state E as well as the best state (*Best*). We set the initial temperature $T = \Delta_{avg}/ln(P)$, where P is the initial probability of accepting uphill moves and the value is very close to 1. Then, we set $N = km$, where m and k are the number of modules and a user defined value, respectively. T is gradually cooled down until the value is lower than a predefined value ϵ, or the rejection rate is larger than 0.95. Let *uphill* denote the number of bad moves. In each temperature, the following process is repeated until *uphill* is larger than N or the procedure runs more than $2N$ times. We first randomly pick one operation from the eight operations proposed in Section V-B. As mentioned in Section V-C, if the selected operation is Reverse, Move, Transpositional Move, or Twirl and the inseparability constraint of TCG is violated during the operation, we give up the operation and reselect a new one; otherwise, a new state NE can be obtained by applying the operation to the current state E. If the new state NE gives a better cost than E (i.e., $\Delta cost = cost(NE) - cost(E) \leq 0$) or $e^{-\Delta cost/T}$ is smaller than a random value *Random*, the new state NE is selected as the current state E; otherwise, the current state E remains unchanged.

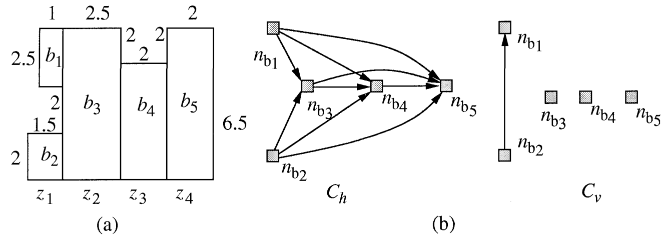

Fig. 10. (a) A nonsliceable rectilinear module. (b) The components in C_h and C_v for the rectilinear module of (a).

VI. TCG for Nonsliceable Rectilinear Modules

Due to the limitation of space, we briefly give the idea on how to deal with nonsliceable rectilinear modules. For a nonsliceable rectilinear module, each zone may contain more than one submodule. Therefore, to maintain the shape of a rectilinear module, we need to keep the relative positions of the submodules in a zone as well as between zones. Given a nonsliceable rectilinear module b_b with p zones by slicing b_b from left to right (or from bottom to top) along vertical (horizontal) boundaries, for each pair of submodules b_{b_i} and b_{b_j} in different zones with b_{b_i} left to (below) b_{b_j}, we introduce an edge (n_{b_i}, n_{b_j}) in C_h (C_v). Also, for each pair of submodules b_{b_i} and b_{b_j} in a zone with b_{b_i} below (left to) b_{b_j}, we introduce an edge (n_{b_i}, n_{b_j}) in C_v (C_h). Similar to the inseparability constraint for sliceable rectilinear modules, for two submodules b_{b_i} and b_{b_j} in adjacent zones, we must guarantee that the corresponding edge (n_{b_i}, n_{b_j}) is a reduction edge during perturbation to maintain the shape of a rectilinear module. Let $s(b_i, b_j)$ denote the spacing between two submodules b_{b_i} and b_{b_j} in the x (y) direction if $b_{b_i} \vdash b_{b_j}$ $(b_{b_i} \perp b_{b_j})$. For example, $s(b_1, b_2) = 2$ since $b_2 \perp b_1$ and their spacing in the y direction is 2. To prevent submodules from being deformed by other modules, for every pair of submodules b_{b_i} and b_{b_j} that do not abut or are not in adjacent zones, we need to impose the following constraint:

- *Dimension Constraint:* $W_x + \cdots + W_y \leq s(b_i, b_j)$ $(H_x + \cdots + H_y \leq s(b_i, b_j))$ if there exists another path $\langle n_{b_i}, n_x, \ldots, n_y, n_{b_j}\rangle$ from n_{b_i} to n_{b_j} in addition to the edge (n_{b_i}, n_{b_j}).

As the example shown in Fig. 10, for each pair of submodules in different zones, we introduce an edge in C_h (see the C_h of Fig. 10(b)). Also, for the submodules b_{b_1} and b_{b_2} in the same zone z_1, we introduce an edge (n_{b_2}, n_{b_1}) in C_v. To maintain the shape of the rectilinear module, we must guarantee that the edges (n_{b_1}, n_{b_3}), (n_{b_2}, n_{b_3}), (n_{b_3}, n_{b_4}), and (n_{b_4}, n_{b_5}) are reduction edges during perturbation. If any of the four edges becomes a closure edge, the submodules will no longer be in adjacent zones in the resulting placement. (See Fig. 4(b) for an example.) Besides, for submodules b_{b_1} and b_{b_2} that do not abut or are not in adjacent zones, if there exists an additional path

$\langle n_{b_2}, n_x, \ldots, n_y, n_{b_1}\rangle$ from n_{b_2} to n_{b_1} in C_v, the summation of $H_x, \ldots, H_y$ cannot be larger than $s(b_1, b_2)$ to avoid submodules b_{b_1} and b_{b_2} from being deformed by modules $b_x, \ldots$, and b_y.

According to the earlier discussions, all operations introduced in Section V-B can be applied with minor modifications by considering the inseparability and dimension constraints.

VII. Experimental Results

Based on the simulated annealing method [8], we implemented the TCG-based rectilinear module placement algorithm using the TCG representation in the C++ programming language on a 433 MHz SUN Sparc Ultra-60 workstation with 1 GB memory. The experiments consist of two parts: area optimization and wirelength optimization.[1] For area optimization, we first compared our method with that presented in [21] based on the same circuits generated by Xu *et al.* To generate L- and T-shaped rectilinear modules for experimentation, they combined two (three) rectangular modules in the MCNC benchmark ami49 to form an L-shaped (T-shaped) module. (Note that all previous works on rectilinear modules generated circuits by themselves without making comparisons with others. Therefore, most of the data are not available to us.) The parameters used for simulated annealing are as follows: initial temperature $= 0.9°$, termination temperature $= 30°$, and the probabilities for operations $M1$, $M2$, $M3$, $M4$, $M5$, $M6$, $M7$, and $M8$ are 0.125, 0.15, 0.15, 0.15, 0.15, 0.075, 0.075, and 0.125, respectively.

Columns 2, 3, and 4 in Table I list the respective numbers of rectangular, L-shaped, and T-shaped modules. ami49_L consists of seven rectangular modules and 21 L-shaped modules, and ami49_LT consists of six rectangular modules, 20 L-shaped modules, and one T-shaped module.[2] As shown in the table, our method achieved significantly better area utilization for ami49_L and ami49_LT, compared to Xu *et al.* [21]. Further, our method is also very efficient (see Column 10 for the runtimes). Figs. 11 and 12 show the placements for ami49_L and ami49_LT. In addition to the two circuits used in Xu *et al.* [21], we also construct three circuits based on ami49. Their configurations are listed in rows 3, 4, and 5 of Table I. The experimental results show that our TCG-based algorithm consistently obtains good results; the dead spaces are all smaller than 6%. Fig. 13 shows the placement for ami49_1.

In addition to L-shaped and T-shaped modules, we also generated two cases with arbitrarily shaped modules, such as H-, $+$-, $\sqcup$-, stair-shaped, etc., to show the flexibility of our method. Our test cases were generated by cutting a rectangle into a set of modules. Fig. 14(a) and (b) [see also Fig. 15(a) and (b)] shows the optimum placement and the resulting placement generated by our methods, respectively. There are six (22) rectangular, two (1) L-shaped, and nine (6) arbitrarily shaped modules in Fig. 14 (see Fig. 15). The dead space is 9.375% (6.944%) and runtime is 1224 (1409) s.

For timing optimization, we estimated the wirelength of a net by half the perimeter of the minimum bounding box enclosing

[1]Although our experiments only "demonstrate" the optimization of the cost metric defined by area or wirelength, the TCG-based approach readily applies to other considerations.

[2]In addition to the two modified ami49 benchmark circuits, Xu et al. [21] also experimented on a small randomly generated test case with 2 rectangular and 4 L-shaped modules. Unlike the two modified ami49 benchmark circuits that can be re-generated (since their module IDs are given in the paper), however, we are unable to re-construct the small randomly generated test case. Therefore, we focus on the comparison with the two modified ami49 benchmark circuits.

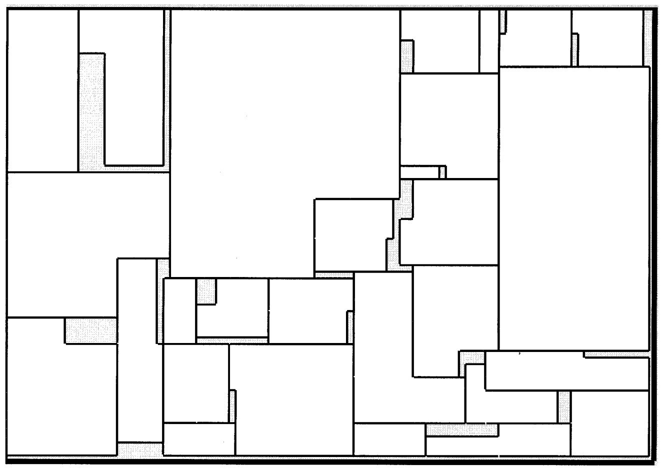
Fig. 11. Resulting placement of ami49_L (area $= 37.24$ mm^2).

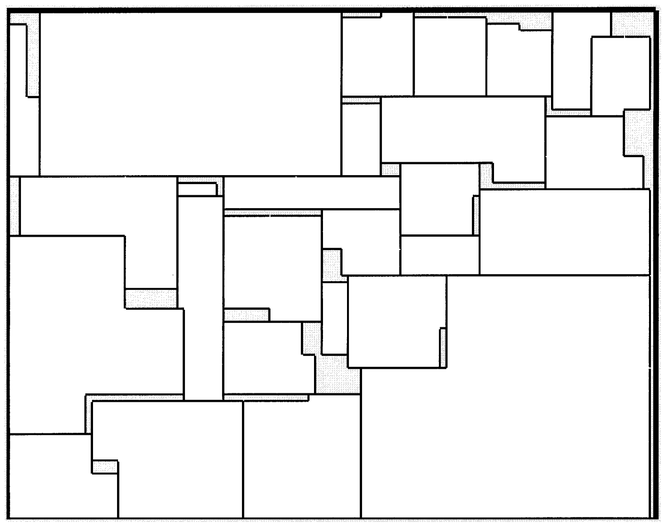
Fig. 12. Resulting placement of ami49_LT (area $= 37.37$ mm^2).

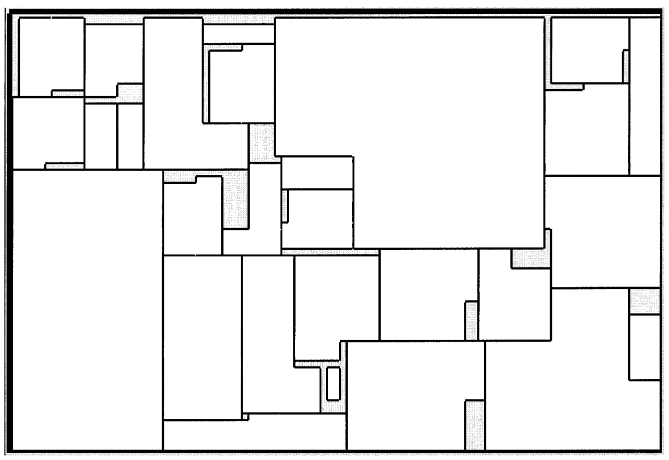
Fig. 13. Resulting placement of ami49_1 (area $= 37.40$ mm^2).

TABLE I

AREA AND RUNTIME COMPARISONS BETWEEN XU *et al.* [21] (ON SUN SPARC ULTRA I WITH 233 MHZ) AND TCG (ON SUN SPARC ULTRA 60 WITH 433 MHZ). (NOTE THAT [21] DOES NOT REPORT RUNTIMES FOR AMI49_L AND AMI49_LT. THE RUNTIME FOR AMI49_L IS TAKEN FROM ITS JOURNAL VERSION [22], BUT [22] DOES NOT REPORT THE RESULT FOR AMI49_LT.) THE OPTIMAL AREA OF AMI49 IS 35.445 mm^2

Circuit	# of rectangular modules	# of L-shaped modules	# of T-shaped modules	Xu et al. [21]			TCG		
				Area (mm^2)	Dead space (%)	Time (sec)	Area (mm^2)	Dead space (%)	Time (sec)
ami49_L	7	21	0	37,39	5.20	1200–	37,24	4.83	448
ami49_LT	6	20	1	39,43	10.09	NA	37,37	5.16	2073
ami49_1	8	19	1	-	-	-	37,40	5.22	991
ami49_2	18	11	3	-	-	-	37,51	5.51	603
ami49_3	9	11	6	-	-	-	37,61	5.76	620
Comparison				-	1.54	-	-	1.00	-

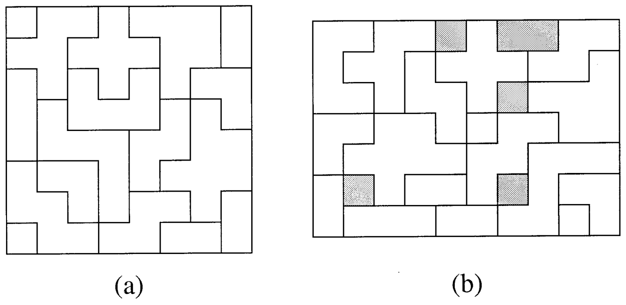
Fig. 14. (a) The optimal placement (area = 64). (b) The resulting placement of (a) (area = 70).

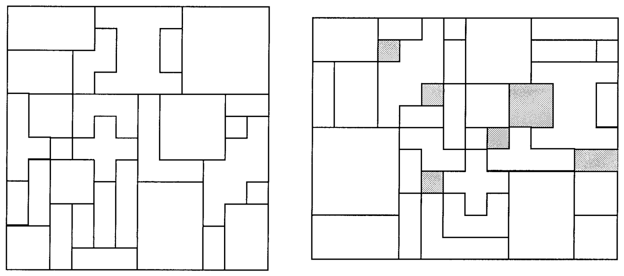
Fig. 15. (a) The optimal placement (area = 144). (b) The resulting placement of (a) (area = 154).

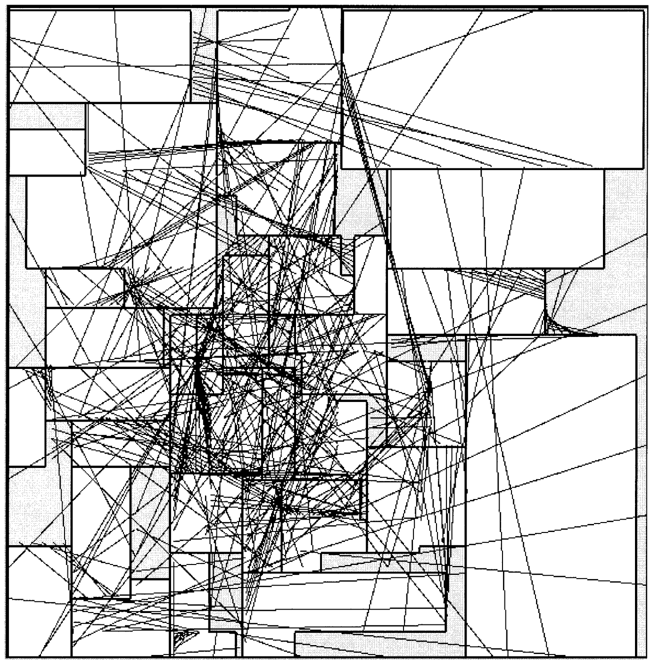
Fig. 16. Resulting placement of ami49_2 (wire = 740 mm^2).

TABLE II

WIRELENGTH AND RUNTIME OF TCG (ON SUN SPARC ULTRA60 WITH 433 MHZ). (NOTE THAT WE ARE THE FIRST WORK THAT RUNS WIRELENGTH WITH RECTILINEAR MODULES)

Circuit	# of Rect. modules	# of L-shaped modules	# of T-shaped modules	TCG	
				wirelength (mm)	Time (sec)
ami49_L	7	21	0	780	1053
ami49_LT	6	20	1	832	2103
ami49_1	8	19	1	798	750
ami49_2	18	11	3	740	605
ami49_3	9	11	6	834	535

the net. The wirelength of a placement is given by the summation of the wirelengths of all nets. Table II shows the experimental results of TCG in optimizing wirelength. (Note that our work is the first to report the results on wirelength optimization for rectilinear modules. So there is no comparative report here.) The resulting placement for ami49_2 with wirelength optimization is shown in Fig. 16.

VIII. CONCLUDING REMARKS

We have presented a TCG-based algorithm to deal with rectilinear module packing for nonslicing floorplans. We have derived necessary and sufficient conditions of TCG for rectilinear modules. Our algorithm not only can avoid infeasible packing during perturbation but also can eliminate the need of the post processing on deformed modules. All these properties make TCG an ideal representation for dealing with the floorplan/placement design with rectilinear modules. Experimental results have shown that our method is very efficient and effective.

REFERENCES

[1] Y.-C. Chang, Y.-W. Chang, G.-M. Wu, and S.-W. Wu, "B^*-trees: A new representation for nonslicing floorplans," in *Proc. DAC*, 2000, pp. 458–463.

[2] K. Fujiyoshi and H. Murata, "Arbitrary convex and concave rectilinear block packing using sequence-pair," *IEEE Trans. Comput.-Aided Design*, vol. 19, pp. 224–233, Feb. 2000.
[3] P.-N. Guo, C.-K. Cheng, and T. Yoshimura, "An O-tree representation of nonslicing floorplan and its applications," in *Proc. DAC*, 1999, pp. 268–273.
[4] X. Hong, G. Huang, Y. Cai, J. Gu, S. Dong, C.-K. Cheng, and J. Gu, "Corner block list: An effective and efficient topological representation of nonslicing floorplan," in *Proc. ICCAD*, 2000, pp. 8–12.
[5] M. Kang and W. Dai, "General floorplanning with L-shaped, T-shaped and soft blocks based on bounded slicing grid structure," in *Proc. ASP-DAC*, 1997, pp. 265–270.
[6] M. Z. Kang and W. W.-M. Dai, "Arbitrary rectilinear block packing based on sequence pair," in *Proc. ICCAD*, 1998, pp. 259–266.
[7] ——, "Topology constrained rectilinear block packing for layout reuse," in *Proc. ISPD*, 1998, pp. 179–186.
[8] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.
[9] E. Lawler, *Combinatorial Optimization: Networks and Matroids*. New York: Holt, Rinehart, and Winston, 1976.
[10] T. C. Lee, "An bounded 2D contour searching algorithm for floorplan design with arbitrarily shaped rectilinear and soft modules," in *Proc. DAC*, 1993, pp. 525–530.
[11] J.-M. Lin and Y.-W. Chang, "TCG: A transitive closure graph-based representation for nonslicing floorplans," in *Proc. DAC*, 2001, pp. 764–769.
[12] J.-M. Lin, H.-L. Chang, and Y.-W. Chang, "Arbitrary convex and concave rectilinear module packing using TCG," in *Proc. DATE*, 2002, pp. 69–75.
[13] Y. Ma, X. Hong, S. Dong, Y. Cai, C.-K. Cheng, and J. Gu, "Floorplanning with abutment constraints and L-shaped/T-shaped blocks based on corner block list," in *Proc. DAC*, 2001, pp. 770–775.
[14] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, "Rectangle-packing based module placement," in *Proc. ICCAD*, 1995, pp. 472–479.
[15] S. Nakatake, K. Fujiyoshi, H. Murata, and Y. Kajitani, "Module placement on BSG-structure and IC layout applications," in *Proc. ICCAD*, 1996, pp. 484–491.
[16] S. Nakatake, M. Furuya, and Y. Kajitani, "Module placement on BSG-structure with pre-placed modules and rectilinear modules," in *Proc. ASP-DAC*, 1998, pp. 571–576.
[17] R. H. J. M. Otten, "Automatic floorplan design," in *Proc. DAC*, 1982, pp. 261–267.
[18] Y. Pang, C.-K. Cheng, K. Lampasert, and W. Xie, "Rectilinear block packing using O-tree representation," in *Proc. ISPD*, 2001, pp. 156–161.
[19] S. M. Sait and H. Youssef, *VLSI Physical Design Automation*. New York: McGraw-Hill, 1995.
[20] G.-M. Wu, Y.-C. Chang, and Y.-W. Chang, "Rectilinear block placement using B^*-trees," in *Proc. ICCD*, 2000, pp. 351–356.
[21] J. Xu, P.-N. Guo, and C.-K. Cheng, "Rectilinear block placement using sequence-pair," in *Proc. ISPD*, 1998, pp. 173–178.
[22] ——, "Sequence-pair approach for rectilinear module placement," *IEEE Trans. Comput.-Aided Design*, vol. 18, pp. 484–493, Apr. 1999.

Jai-Ming Lin received the B.S. and M.S. degrees in computer and information science from National Chiao Tung University, Hsinchu, Taiwan, R.O.C., in 1996 and 1998, respectively. He will receive the Ph.D. degree, also in computer and information science from the same university in 2002.

He is planning on becoming a CAD engineer with Realtek Semiconductor Corp., Hsinchu Science-Based Industrial Park, Taiwan. His research interest focuses on physical design.

Hsin-Lung Chen received the B.S. degree in computer and information science from National Chiao Tung University, Hsinchu, Taiwan, R.O.C., in 2000. He will receive the M.S. degree in computer and information science from the same university in 2002.

He is planning on becoming a CAD engineer with Etron Technology, Inc., Hsinchu Science-Based Industrial Park, Taiwan. His research interest focuses on foorplan design in VLSI.

Yao-Wen Chang (S'94–M'96) received the B.S. degree from National Taiwan University, Taipei, Taiwan, R.O.C., in 1988, and the M.S. and the Ph.D. degrees from the University of Texas at Austin in 1993 and 1996, respectively, all in computer science.

Currently, he is an Associate Professor with the Department of Electrical Engineering and the Graduate Institute of Electronics Engineering, National Taiwan University, Taiwan, R.O.C. He was with the VLSI design group of IBM T. J. Watson Research Center, Yorktown Heights, NY, in summer 1994. From 1996 to 2001, he was an Associate Professor with the Department of Computer and Information Science, National Chiao Tung University, Hsinchu, Taiwan, where he received an inaugural all-university Excellent Teaching Award (ranked first in the Department) in 2000. His research interests lie in physical design automation, architectures, and systems for VLSI and combinatorial optimization.

Dr. Chang received the Best Paper Award at the 1995 IEEE International Conference on Computer Design (ICCD-95) for his work on FPGA routing, and the Best Paper Nomination at the 2002 IEEE/ACM International Conference on Computer-Aided Design (ICCAD-2002) for his work on multilevel routing. He has served on the technical program committees of several conferences, including VLSI design automation. He is a member of IEEE Circuits and Systems Society, ACM, and ACM/SIGDA.