Arbitrary Convex and Concave Rectilinear Module Packing Using TCG *

Jai-Ming Lin[1], *Hsin-Lung Chen*[1], *and Yao-Wen Chang*[2]

[1] Department of Computer and Information Science, National Chiao Tung University, Hsinchu 300, Taiwan

[2] Department of Electrical Engineering & Graduate Institute of Electronics Engineering, National Taiwan University, Taipei 106, Taiwan

Abstract

In this paper, we deal with arbitrary convex and concave rectilinear module packing using the Transitive Closure Graph (TCG) representation. The geometric meanings of modules are transparent to TCG and 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 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. 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.

1 Introduction

As technology advances, design complexity is increasing at a dramatic pace. To handle such design complexity, hierarchical design and reuse of IP modules become popular. This trend makes the number of modules increase significantly, and often modules are not rectangular. Therefore, it is desirable to consider the placement for arbitrarily shaped rectilinear modules to optimize silicon area utilization.

1.1 Previous Work

Placement/floorplanning with rectilinear modules has been extensively studied in the literature [1, 2, 3, 4, 7, 11, 15, 16]. Lee in [7] represented arbitrarily shaped rectilinear modules with a set of four linear profiles which describe the contours of a module viewed from four sides. They minimized silicon size by performing a bounded 2D 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 [7], most previous works 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: BSG [2, 4, 11], sequence pair [1, 3, 16], B*-tree [15], O-tree [13], and CBL [9].

Kang and Dai in [2] 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 [11] 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 [4] used BSG and sequence pair to solve the topology constrained module packing for a specific class of rectilinear modules, named *ordered convex rectilinear modules*, and extended the method to handle arbitrary rectilinear modules.

Xu et al. in [16] explored the conditions of feasible sequence pairs 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 [1]. Kang and Dai in [3] derived three necessary and sufficient conditions for recovering the shapes of convex rectilinear modules. Similarly, they also needed a post processing to recover the original shapes of rectilinear modules after packing. Recently, Fujiyoshi and Murata in [1] 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 maintain the feasibility of a sequence pair, without resorting to a post processing for fixing misplaced submodules. However, the constraint graphs are no longer acyclic after their augmentation, resulting in a longer running time for packing ($O(m^3)$ time, where m is the number of modules).

Wu et al. in [15] handled rectilinear modules using the B*-tree representation. A rectilinear module can easily be represented using B*-tree by partitioning the module into a set of rectangular submodules. However, they have to re-partition a rectilinear module whenever the rectilinear module is rotated. Besides, they need a post-processing to adjust submodules to maintain the shapes of rectilinear modules. Pang et al. [13] extended the O-tree representation to handle rectilinear modules. Recently, Ma et al. [9] used CBL to deal with the placement abutment constraint and extended the method to deal with L/T-shaped modules.

1.2 Our Contribution

In this paper, we deal with arbitrary convex and concave rectilinear module packing 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 TCGs 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 moduls 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 arbitrary shaped modules presented in [1]. 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 utilization (average dead space = 5.44%) than the previous work [16] (average dead space = 7.65%).

The remainder of this paper is organized as follows. Section 2 formulates the floorplan/placement design problem with rectilinear modules. Section 3 reviews the TCG representation. Section 4 presents the feasible TCG and packing algorithm for convex and some concave rectilinear modules. Section 5 introduces the perturbation algorithm for rectilinear modules. Section 6 extends TCG to deal with general rectilinear modules. Experimental results are reported in Section 7, and concluding remarks are given in Section 8.

2 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 *non-sliceable*.

Theorem 1 *All convex modules must be sliceable.*

Corollary 1 *All non-sliceable modules are concave modules.*

As the non-sliceable module shown in Figure 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 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 rectangular submodules $\{b_{i_1}, b_{i_2}, \ldots, b_{i_k}\}$ ordered from bottom to top (from left to right). Figure 1 shows a non-sliceable rectilinear module with four zones z_1, z_2, z_3,

*This work was partially supported by the National Science Council of Taiwan ROC under Grant No. NSC-90-2215-E-009-117. E-mail: {gis87808, is85019}@cis.nctu.edu.tw; ywchang@cc.ee.ntu.edu.tw.

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 (W_{i_j}), H_i (H_{i_j}), A_i (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 minimize the resulting area (i.e., the minimum bounding rectangle of P).

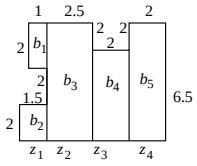


Figure 1: A concave rectilinear module consisting of four zones z_1, z_2, z_3, z_4, where $z_1 = \{b_1, b_2\}, z_2 = \{b_3\}, z_3 = \{b_4\}$, and $z_4 = \{b_5\}$.

3 Review of TCG

We first review the TCG representation presented in [8]. 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 (C_v) denotes that module b_i is left to (below) module b_j. TCG has the following three *feasibility properties* [8]:

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 (C_v) is equal to C_h (C_v) itself.[1]

Figure 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. Figure 2(b) shows the TCG $= (C_h, C_v)$ corresponding to the placement of Figure 2(a). The value associated with a node in C_h (C_v) gives the width (height) of the corresponding module, and the edge (n_i, n_j) in C_h (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* [6] on TCG, 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. Figure 2(c) shows the augmented TCG for the TCG shown in Figure 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.

4 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 non-sliceable rectilinear modules in Section 6.)

[1] 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 $G\}$.

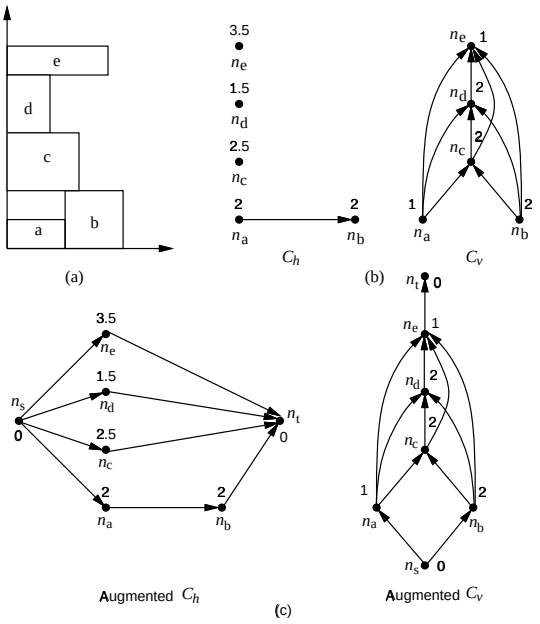


Figure 2: (a) A placement. (b) TCG. (c) Augmented TCG (augmented C_h and C_v).

4.1 Feasible TCG

We have shown in the previous section 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 Figure 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 the vertical (horizontal) boundaries, we can construct a set of reduction edges $(n_{b_j}, n_{b_{j+1}})$, $j = 1, \ldots, p-1$, and corresponding closure edges in C_h (C_v) since $b_{b_j} \vdash b_{b_{j+1}}$ ($b_{b_j} \perp b_{b_{j+1}}$), $j = 1, \ldots, p-1$. (See Figures 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 $(n_{b_j}, n_{b_{j+1}})$, $j = 1, \ldots, p-1$, 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 $(n_{b_j}, n_{b_{j+1}})$, $j = 1, \ldots, p-1$, remains as a reduction one.

Figures 3(a)–(h) show eight possible situations of rectilinear module b_b after rotation and flip. As shown in Figures 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 Figures 3(a)–(d) ((e)–(h)), the corresponding TCG is illustrated in Figure 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 $< n_{b_j}, n_x, \ldots, n_y, n_{b_{j+1}} >$ from n_{b_j} to $n_{b_{j+1}}$). Figure 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 Figure 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 $<$

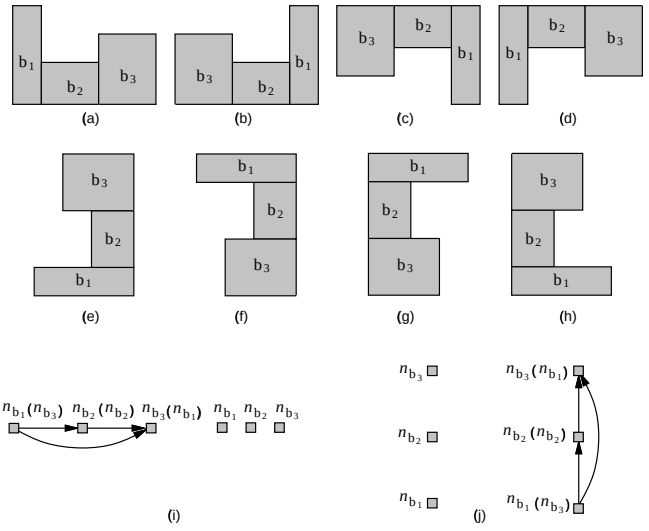


Figure 3: (a)–(h) Eight situations of the submodules for the rectilinear module b_b. (i) The subgraph of C_h and C_v for the rectilinear modules shown in (a)–(d). (j) The subgraph of C_h and C_v for the rectilinear modules shown in (e)–(h).

$n_{b_3}, n_d, n_{b_2} >$ 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 Figure 4(c), there exist two paths $< n_{b_3}, n_d, n_{b_1} >$ and $< n_{b_3}, n_{b_2}, n_{b_1} >$ from n_{b_3} to n_{b_1} in C_v. Suppose that $H_d > H_{b_2}$. 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}. Same as [13, 17], we can resolve it 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).

4.2 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 3. Figure 5 illustrates the difference between the packings for a rectangular and a rectilinear modules. Figure 5(a) shows a given TCG with four rectilinear modules, b_a, b_b, b_c, and b_d, where b_b is a non-rectangular module whose shape is illustrated in Figure 5(b). Figures 5(c) and (d) show 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 Figure 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, Figure 5(d) gives a correct packing while Figure 5(c) does not. As mentioned in Section 3, all modules should be packed in the topological order in C_h (C_v). To process a non-rectangular 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 Figure 5(a), we should determine the Y coordinates of modules b_a and b_c before processing the non-rectangular 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 $F_{in}(n_i)$ ($F_{out}(n_i)$), be the nodes n_j's 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

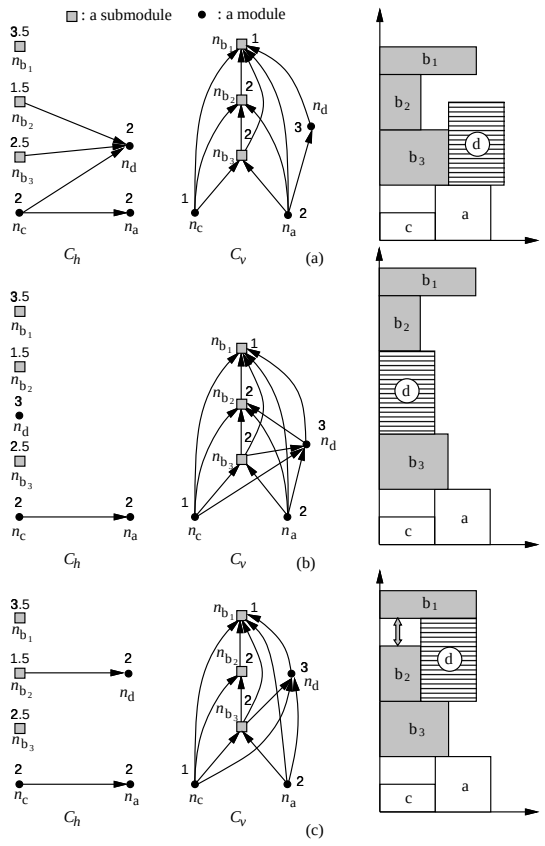


Figure 4: (a) A feasible TCG for a rectilinear module b_b and its corresponding placement. (b) A TCG that violates the inseparability constraint and its corresponding placement. (c) Expanding submodule b_{b_2} to abut with b_{b_1}. (In the figures, a quadratic node corresponds to a submodule while a round one corresponds to a rectangular module.)

edges into the augmented C_h (C_v) to make $F_{in}(n_{b_j}) = F_{in}(n_{b_{j+1}})$, $j = 1, \ldots, p-1$, except the edges emanating from n_s. (See Figure 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 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 $n_j \in F_{out}(n_i)$, 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} of a rectilinear module 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_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_j and b_i before relaxing n_{b_i}.

Figure 6(a) shows the augmented C_v for the C_v shown in Figure 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 $F_{in}(n_{b_1}) = F_{in}(n_{b_2}) = F_{in}(n_{b_3})$ (see Figure 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 $< n_s, n_c, n_a, n_{b_1}, n_d, n_{b_2}, n_{b_3}, n_t >$, 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 Figures 6(c)–(g)). Figure 6(c) shows the initial configuration, in which all nodes have the c-value -1 except that $c_s = 0$. Figure 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 Figure 6(e)) and then compute c_{b_2} and c_{b_3} (see Figure 6(f)) before relaxing n_{b_1} to maintain the relative positions of the rectilinear

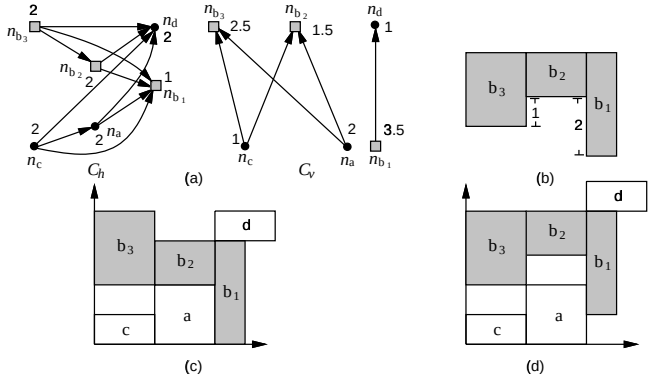


Figure 5: (a) A TCG. (b) The rectilinear module b_b consists of three submodules b_{b_1}, b_{b_2}, and b_{b_3}. (c) An incorrect packing resulting from the original packing procedure. (d) A correct packing. (Note that this packing is suboptimal; we will show in next section how to perturb the TCG to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

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 Figure 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.*

It should be noted that the sequence pair-based packing scheme presented in [1] needs $O(m'^3)$ time.

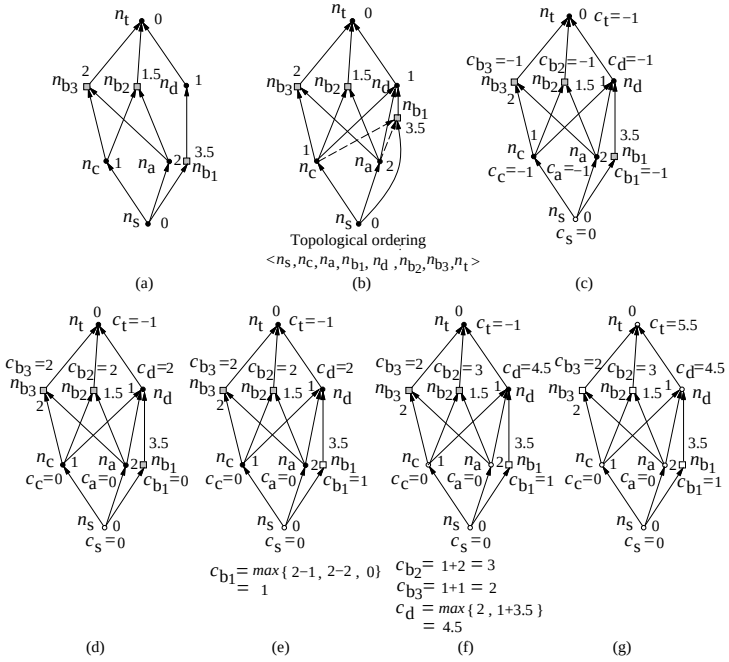


Figure 6: An example computation of Y coordinates. (a) The augmented C_v for the C_v shown in Figure 5(a). (b) The new augmented C_v after adding the edges (n_c, n_{b_1}) and (n_a, n_{b_1}) to make $F_{in}(n_{b_1}) = F_{in}(n_{b_2}) = F_{in}(n_{b_3})$. A possible topological order in the new augmented C_v is given as $< n_s, n_c, n_a, n_{b_1}, n_d, n_{b_2}, n_{b_3}, n_t >$. (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.

5 Algorithm

Our algorithm is based on simulated annealing [5]. 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 3 and the inseparability constraint presented in Section 4.1. To identify a feasible TCG for perturbation, we need to identify reduction edges.

5.1 Reduction Edge Identification

Recall that 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 $n_j \in F_{out}(n_i)$. For nodes $n_k, n_l \in F_{out}(n_i)$, the edge (n_i, n_k) cannot be a reduction edge if $n_k \in F_{out}(n_l)$. Hence, we remove those nodes in $F_{out}(n_i)$ that are fan-outs of others. The edges between n_i and the remaining nodes in $F_{out}(n_i)$ are reduction edges. For the C_h of Figure 5(a), $F_{out}(n_c) = \{n_a, n_{b_1}, n_d\}$. Since n_{b_1} and n_d belong to $F_{out}(n_a)$, 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 [8].

5.2 Solution Perturbation

We apply the following eight operations to perturb a TCG:

- *Rotation:* Rotate a rectangular module.
- *Swap:* Swap two nodes associated with two 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 [8], 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 do not change the topology of TCG, and thus the inseparability constraint will not be violated, either. However, Revere, Move, Transpositional Move, and Twirl will. We thus may need to update TCG after performing Revere, Move, Transpositional Move, and Twirl. Further, in order to satisfy the inseparability constraint, we need feasibility detection during the operation. We first detail the operations in the following.

5.2.1 Rotation

To rotate a *rectangular module* b_i, we only need to exchange the weights of the corresponding nodes n_i in C_h and C_v.

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

5.2.2 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.*

5.2.3 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 $n_k \in F_{in}(n_j) \cup \{n_j\}$ and $n_l \in F_{out}(n_i) \cup \{n_i\}$ 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.

5.2.4 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 $n_k \in F_{in}(n_i) \cup \{n_i\}$ and $n_l \in F_{out}(n_j) \cup \{n_j\}$, 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.

5.2.5 Transpositional Move

The Transpositional Move operation removes a *reduction* edge (n_i, n_j) from a transitive closure graph, and add 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 $n_k \in F_{in}(n_j) \cup \{n_j\}$ and $n_l \in F_{out}(n_i) \cup \{n_i\}$, 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.*

5.2.6 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 $j \neq i$. Figure 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 Figure 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 Figure 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.*

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

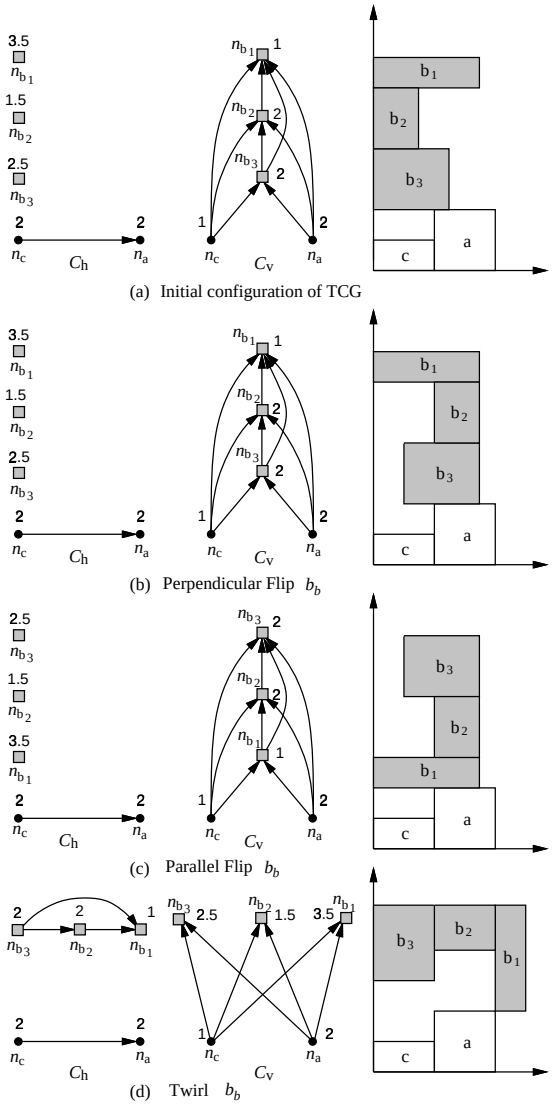


Figure 7: Examples of perturbations. (a) The initial TCG (C_h and C_v) and its corresponding placement. (b) The resulting TCG and placement after perpendicularly flipping the rectilinear module b_b shown in (a). (c) The resulting TCG and placement after parallel flipping the rectilinear module b shown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

5.2.7 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 $l = p, \ldots, 2$.

Figure 7(c) shows the resulting C_h, C_v, and placement after parallel flipping the rectilinear module b_b of Figure 7(b). Notice that after swapping the nodes n_{b_1} and n_{b_3} in both C_h and C_v of Figure 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 Figure 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.*

Lemma 4 *The inseparability constraint of a TCG will not be violated for the Parallel Flip operation.*

5.2.8 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 $b_{b_l} \perp b_{b_{l-1}}$ ($b_{b_l} \vdash b_{b_{l-1}}$), $l = p, \ldots, 2$.

Figure 7(d) shows the resulting C_h, C_v, and placement after twirling the b_b of Figure 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 $\{n_{b_2}\} \cup F_{in}(n_{b_2}) = \{n_{b_2}\}$ and $\{n_{b_1}\} \cup F_{out}(n_{b_1}) = \{n_{b_1}\}$ 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 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 $\{n_{b_3}\} \cup F_{in}(n_{b_3}) = \{n_{b_3}\}$ and $\{n_{b_2}\} \cup F_{out}(n_{b_2}) = \{n_{b_1}, n_{b_2}\}$ 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 Figure 7(c). Figure 7(d) illustrates the resulting TCG.

The Twirl operation needs to perform respective Rotate and Transpositional Move p and $p-1$ times, and each Rotate 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.

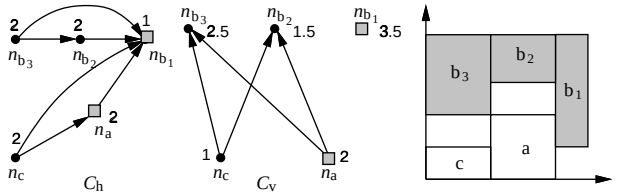
Theorem 6 *TCG is closed under the Twirl operation, and such an operation takes $O(m'^2 p)$ time, where p is the number of submodules in a rectilinear module and m' is the number of rectangular modules and submodules.*

5.3 Feasibility Detection

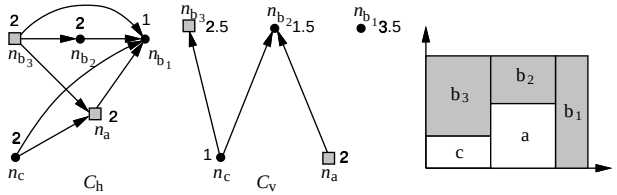
To maintain the shape of a rectilinear module, TCG must satisfy the inseparability constraint for each rectilinear module. Among the eight operations, only Reverse, Move, Transpositional Move, and Twirl could violate the constraints, which can easily be detected during perturbation. When we reverse an edge (n_i, n_j) or move/transpositionally move (n_j, n_i), the inseparability constraint will be violated if $n_{b_l} \in F_{in}(n_i) \cup \{n_i\}$ and $n_{b_k} \in F_{out}(n_j) \cup \{n_j\}$, where $|l-k| = 1$, since (n_{b_l}, n_{b_k}) would become a closure edge. Since Twirl consists of the 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 module. 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.*

Figure 8(e) shows the resulting C_h, C_v, and placement after moving the edge (n_a, n_{b_1}) from C_v shown in Figure 7(d) to C_h. Since $\{n_a\} \cap F_{in}(n_a) = \{n_a, n_c\}$ and $\{n_{b_1}\} \cap F_{out}(n_{b_1}) = \{n_{b_1}\}$ 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. Figure 8(f) shows the resulting C_h, C_v, and placement after transpositionally moving the edge (n_a, n_{b_3}) from the C_v of Figure 8(e) to C_h. Since $\{n_{b_3}\} \cap F_{in}(n_{b_3}) = \{n_{b_3}\}$ and $\{n_a\} \cap F_{out}(n_a) = \{n_a, n_{b_1}\}$ 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.



(e) Move the edge (n_a, n_{b_1}) from C_v to C_h



(f) Transpositionally move the edge (n_a, n_{b_3}) from C_v to C_h

Figure 8: Examples of perturbations (continued from Figure 7). (e) The resulting TCG and placement after moving the reduction edge (n_a, n_{b_1}) from the C_v of Figure 7(d) to C_h. (f) 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.

6 TCG for Non-Sliceable Rectilinear Modules

Due to the page limit, we briefly give the idea on how to deal with non-sliceable rectilinear modules. For a non-sliceable 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 non-sliceable 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*: $W_x + \ldots + W_y \leq s(b_i, b_j)$ ($H_x + \ldots + H_y \leq s(b_i, b_j)$) if there exists another path $< n_{b_i}, n_x, \ldots, n_y, n_{b_j} >$ from n_{b_i} to n_{b_j} in addition to the edge (n_{b_i}, n_{b_j}).

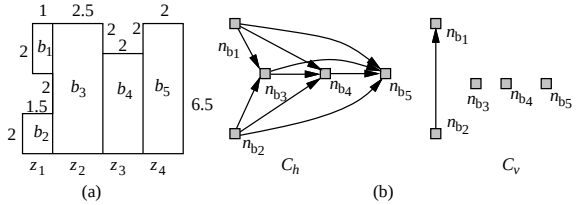


Figure 9: (a) A non-sliceable rectilinear module. (b) The components in C_h and C_v for the rectilinear module of (a).

As the example shown in Figure 9, for each pair of submodules in different zones, we introduce an edge in C_h (see the C_h of Figure 9(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 Figure 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 $< n_{b_2}, n_x, \ldots, n_y, n_{b_1} >$ from n_{b_2} to n_{b_1} in C_v, the summation of H_x, ..., 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, ..., and b_y.

According to the above discussions, all operations introduced in Subsection 5.2 can be applied with minor modifications by considering the inseparability and the dimension constraints.

7 Experimental Results

Based on the simulated annealing method [5], 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 package is available at http://cc.ee.ntu.edu.tw/∼ywchang/research.html. We compared our method with that presented in [16] 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.)

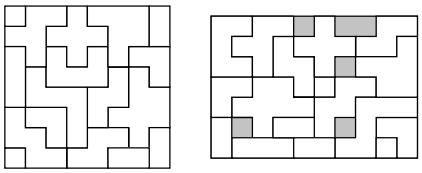
Circuit	# of R	# of L	# of T	Xu et al. [16] area	Xu et al. [16] Time (sec)	TCG area	TCG Time (sec)
ami49_L	7	21	0	37,39	1200–	37,30	203
ami49_LT	6	20	1	39,43	NA	37,37	2073
ami49_1	8	19	1	-	-	37,40	991
ami49_2	18	10	3	-	-	37,51	603
ami49_3	9	11	6	-	-	37,61	620

Table 1: Area and runtime comparisons between Xu et al. [16] (on Sun Sparc Ultra I with 233 MHz) and TCG (on Sun Sparc Ultra60 with 433 MHz). (Note that [16] does not report runtimes for ami49_L and ami49_LT. The runtime for ami49_L is taken from its journal version [17], but [17] does not report the result for ami49_LT.) The area of all modules in ami49 is $35.445mm^2$.

Columns 2, 3, and 4 in Table 1 list the respective numbers of rectangular, L-shaped, T-shaped modules (denoted by # of R, L, and T). ami49_L consists of 7 rectangular modules and 21 L-shaped modules, and ami49_LT consists of 6 rectangular modules, 20 L-shaped modules, and 1 T-shaped module.[2] The total area of each circuit is shown in Column 5. As shown in the table, our method achieved significantly better area utilization for ami49_L and ami49_LT, compared to Xu et al. [16]. Further, our method is also very efficient (see Column 11 for the running times). Figure 10 shows the placement for ami49_LT. In addition to the two circuits used in Xu et al. [16], we also construct three circuits based on ami49. Their configurations are listed in rows 3, 4, and 5 of Table 1. The experimental results show that our TCG-based algorithm consistently obtains good results; the dead spaces are all smaller than 6%.

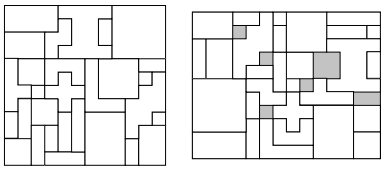
Figure 10: Resulting placement of ami49_LT (area = $37.37mm^2$).

In addition to L-shaped and T-shaped modules, we also generated two cases with arbitrarily shaped modules, such as H-, +-, ⊔-, stair-shaped, etc., to show the flexibility of our method. Our test cases were generated by cutting a rectangle into a set of modules. Figures 11 (12)(a) and (b) show the optimum placement and the resulting placement generated by our methods, respectively. There are 6 (22) rectangular, 2 (1) L-shaped, and 9 (6) arbitrarily shaped modules in Figure 11 (12). The dead space is 9.375% (6.944%), and the running time is 1224 (1409) sec. Note that the test cases are two of most complex benchmarks ever reported in the literature.



(a) (b)

Figure 11: (a) The optimal placement (area=64). (b) The resulting placement of (a) (area = 70).



(a) (b)

Figure 12: (a) The optimal placement (area=144). (b) The resulting placement of (a) (area=154).

8 Concluding Remarks

We have presented a TCG-based algorithm to deal with rectilinear module packing. TCG is the first *general* graph representation with the feasibility guarantee for perturbations. 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 flexible and effective.

References

[1] K. Fujiyoshi and H. Murata, "Arbitrary Convex and Concave rectilinear Block Packing Using Sequence-Pair," *IEEE TCAD*, vol. 19, no. 2, pp. 224–233, Feb. 2000.

[2] M. Kang and W. Dai., "General Floorplanning with L-shaped, T-shaped and Soft Blocks Based on Bounded Slicing Grid Structure," *Proc. ASP-DAC*, pp. 265–270, 1997.

[3] M. Z. Kang and W. W.-M Dai., "Arbitrary Rectilinear Block Packing Based on Sequence Pair," *Proc. ICCAD*, pp. 259–266, 1998.

[4] M. Z. Kang and W. W.-M. Dai, "Topology Constrained Rectilinear Block Packing for Layout Reuse," in *Proc. ISPD*, pp. 179–186, 1998.

[5] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, no. 4598, May 13, 1983, pp.671–680.

[6] E. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart, and Winston, 1976.

[7] T. C. Lee, "An Bounded 2D Contour Searching Algorithm for Floorplan Design with Arbitrarily Shaped Rectilinear and Soft Modules," *Proc. DAC*, pp. 525–530, 1993.

[8] J.-M. Lin and Y.-W. Chang, "TCG: A Transitive Closure Graph-Based Representation for Non-Slicing Floorplans," *Proc. DAC*, June 2001.

[9] Y. Ma, X. Hong, S. Dong, Y. Cai, C.-K. Cheng, and Jun Gu,"Floorplanning with Abutment Constraints and L-shaped/T-shaped Blocks based on Corner Block List, " *Proc. DAC*, pp. 770-775, 2001.

[10] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, "Rectangle-Packing Based Module Placement," *Proc. ICCAD*, pp. 472–479, 1995.

[11] S. Nakatake, M. Furuya, and Y. Kajitani, "Module Placement on BSG-Structure with Pre-Placed Modules and Rectilinear Modules," *Proc. ASP-DAC*, pp. 571–576, 1998.

[12] R. H. J. M. Otten, "Automatic Floorplan Design,"*Proc. DAC*, pp.261–267, 1982.

[13] Y. Pang, C.K. Cheng, K. Lampasert, and W. Xie, "Rectilinear Block Packing Using O-tree Representation," *Proc. ISPD*, pp. 156-161, 2001.

[14] S. M. Sait and H. Youssef, *VLSI Physical Design Automation.*

[15] G.-M. Wu, Y.-C. Chang, and Y.-W. Chang, "Rectilinear Block Placement Using B*-Trees," *Proc. ICCD*, 2000.

[16] J. Xu, P.-N. Guo, and C.-K. Cheng, "Rectilinear Block Placement Using Sequence-Pair," *Proc. ISPD*, pp. 173–178, 1998.

[17] J. Xu, P.-N. Guo, and C.-K. Cheng, "Rectilinear Block Placement Using Sequence-Pair," *IEEE Trans. Computer-Aided Design*, pp. 484–493, 1999.

[2] In additional to the two modified ami49 benchmark circuits, Xu et al. [16] 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 ID's 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.