TCG: A Transitive Closure Graph-Based Representation for Non-Slicing Floorplans *

Jai-Ming Lin and Yao-Wen Chang
{gis87808, ywchang}@cis.nctu.edu.tw
Department of Computer and Information Science, National Chiao Tung University, Hsinchu 300, Taiwan

Abstract

In this paper, we propose a transitive closure graph-based representation for general floorplans, called *TCG*, and show its superior properties. TCG combines the advantages of popular representations such as sequence pair, BSG, and B*-tree. Like sequence pair and BSG, but unlike O-tree, B*-tree, and CBL, TCG is P-admissible. Like B*-tree, but unlike sequence pair, BSG, O-tree, and CBL, TCG does not need to construct additional constraint graphs for the cost evaluation during packing, implying faster runtime. Further, TCG supports incremental update during operations and keeps the information of boundary modules as well as the shapes and the relative positions of modules in the representation. More importantly, the geometric relation among modules is transparent not only to the TCG representation but also to its *operations*, facilitating the convergence to a desired solution. All these properties make TCG an effective and flexible representation for handling the general floorplan/placement design problems with various constraints. Experimental results show the promise of TCG.

1 Introduction

Due to the increase in design complexity, circuit size is getting larger in modern VLSI design. To handle the design complexity, hierarchical design and reuse of IP modules become popular, which makes floorplanning/placement much more important than ever. The major objective of floorplanning/placement is to allocate the modules of a circuit into a chip to optimize its area and timing. The realization of floorplanning/placement relies on a representation which describes geometric relations among modules. The representation has a great impact on the feasibility and complexity of floorplan designs. Thus, it is of particular significance to develop an efficient, effective, and flexible representation for floorplan/placement designs.

1.1 Previous Work

Floorplans can be classified into two categories, the slicing structure [10], [16] and the non-slicing structure [1], [2], [3], [6], [7], [8], [9], [11], [12], [15]. A slicing structure can be obtained by recursively cutting rectangles horizontally or vertically into smaller rectangles; it is a non-slicing structure, otherwise. Otten first proposed a binary-tree representation for slicing floorplan design [10]. Wong and Liu later in [16] presented a normalized Polish expression to represent a slicing floorplan. The slicing structure has several advantages such as smaller solution space, implying faster runtime for floorplan design. However, most of real designs are non-slicing. Researchers in [11], [15] attempted to extend the tree representation to the non-slicing floorplans with special topologies, e.g., the wheel structure.

*This work was partially supported by the National Science Council of Taiwan ROC under Grant No. NSC-90-2215-E-009-117.

DAC 2001, June 18-22, 2001, Las Vegas, Nevada, USA.

For the general non-slicing structure, Ohtsuki in [8] used a pair of (horizontal and vertical) directed acyclic graphs, named *polar graphs*, to represent a topological placement. A similar representation, named *adjacency graphs* [13], was also widely used. Recently, Onodera et al. in [9] used a branch-and-bound method to exhaustively search an optimal solution for module placement. Since the method is quite time-consuming, the size of tractable modules is limited.

Recently, there is much work in the literature on non-slicing floorplan representations, e.g., *sequence pair* [6], *BSG* [7], *O-tree* [2], *B*-tree* [1], and *CBL* [3]. Murata et al. in [6] used two sequences of permutations, namely sequence pair, to represent the geometric relations of modules for non-slicing floorplan design. They defined the *P-admissible solution space*, which satisfies the following four requirements: (1) the solution space is finite, (2) every solution is feasible, (3) packing and cost evaluation can be performed in polynomial time, and (4) the best evaluated packing in the space corresponds to an optimal placement [6]. Sequence pair is P-admissible and is flexible for general floorplan/placement design; however, it is harder to handle the floorplan/placement problems with position constraints, e.g., boundary modules, pre-placed modules, range constraints, etc. Further, two constraint graphs need to be constructed for cost evaluation for each perturbation, consuming a significantly larger run time. Tang and Wong [14] recently presented an efficient packing scheme, called *FAST-SP*, to evaluate the cost of a sequence pair by computing its common subsequence. Nakatake et al. in [7] proposed a flexible bounded slicing grid representation, called BSG. BSG is also P-admissible. However, BSG itself has many redundancies since there could be multiple representations corresponding to one packing, implying a larger solution space and thus longer search time to find an optimal solution.

For tree-based methods, Guo et al. in [2] first proposed the O-tree representation for a left and bottom compacted placement. In an 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. The placement after an O-tree packing might not be compacted. A sequence of compaction operations along the x and/or y directions may be needed to make all modules compacted to the left and bottom, possibly resulting in a mismatch between the original representation and its placement. Chang et al. recently 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 visible module above and with the same x coordinate. Similar to O-tree, the representation could be changed after packing since the space intended for placing a module may be occupied by a previously placed module. Therefore, given an O-tree or a B*-tree, it may not be feasible to find a placement corresponding to its original representation, and thus they are not P-admissible. Although the operations of the tree-based representations are more efficient and their solution spaces are smaller than sequence pair and BSG, the geometric relation of any pair of modules cannot be derived directly from the representations, unless the corresponding nodes are siblings or lineal relatives. Even for the nodes that are lineal relatives or siblings, the geometric relation defined by the representation might be changed after packing. Hence, it may be harder for the tree-based representations to handle some problems with relative position constraints, such as boundary and range constraints, etc.

Figure 1(a) shows a placement, and Figures 1(b) and (c) show an O-tree encoded by $(T, \pi) = (0011001101, abcde)$ and a B*-tree corresponding to the placement, respectively. For O-trees and B*-trees, we cannot derive any geometric relation between two modules from the representations unless the two nodes are siblings or on the same path. For example, even though module b is adjacent to module d, we cannot derive any geometric relation from the representations. Also, we do not know which nodes represent the top and right boundary modules from the representations.

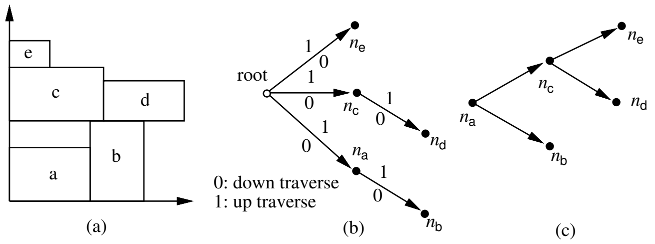

Figure 1: (a) A placement. (b) O-tree. (c) B*-tree.

Recently, Hong et al. in [3] proposed a corner block list (CBL) representation for non-slicing floorplans. Like sequence pair and BSG, but unlike the tree-based representations, CBL can represent general uncompacted non-slicing floorplans. CBL has smaller feasible solution space and a faster packing scheme; however, it is not P-admissible since it cannot guarantee a feasible solution in each perturbation, and many infeasible solutions may be generated before a feasible solution is found.

1.2 Our Contribution

We propose in this paper a transitive closure graph-based representation for non-slicing floorplans, named *TCG*, and show its superior properties. TCG uses a horizontal and a vertical transitive closure graphs to describe the horizontal and vertical relations for each pair of modules. It combines the advantages of sequence pair, BSG, and B*-tree. Like sequence pair and BSG, but unlike O-tree, B*-tree, and CBL, TCG satisfies the four properties of P-admissibility [6]: (1) its solution space is $(m!)^2$ and thus finite, where m is the number of modules, (2) it guarantees a *unique* feasible packing for each representation (note that the tree-based and the CBL representations do not guarantee this property), (3) packing and cost evaluation can be performed in $O(m^2)$ time, and (4) the best evaluated packing in the solution space corresponds to an optimum placement. Like B*-tree, but unlike sequence pair, BSG, O-tree, and CBL, TCG does not need to construct additional constraint graphs for the cost evaluation during packing, implying faster runtime. Further, TCG supports incremental update during operations, and keeps the information of boundary modules as well as the shapes and the relative positions of modules in the representation. More importantly, the geometric relation among modules is transparent not only to the TCG representation but also to its *operations* (i.e., the effect of an operation on the change of the geometric relation is known *before* packing), facilitating faster convergence to a desired solution. All these properties make TCG an effective and flexible representation for handling the general floorplan/placement design problems with various constraints such as boundary constraints, etc. Experimental results show the promise of TCG. For area optimization, TCG achieved average improvements of 2.22%, 2.04%, 1.18%, and 3.54%, compared to O-tree, enhanced O-tree, B*-tree, and CBL, respectively. Optimizing wirelength, TCG obtained respective average improvements of 3.56% and 3.18%, compared to O-tree and enhanced O-tree. (Note that B*-tree and CBL do not report the results for optimizing wirelength alone.) The runtime requirements of TCG are much smaller than O-tree and B*-tree, and are comparable to enhanced O-tree. We summarize in Table 1 the properties of several recently published representations for non-slicing floorplans.

The remainder of this paper is organized as follows. Section 2 formulates the floorplan/placement design problem. Section 3 presents the procedures to derive a TCG from a placement and construct a placement from a TCG. Section 4 introduces the operations to perturb a TCG. Experimental results are reported in Section 5. Finally, we conclude our work and discuss future research directions in Section 6.

2 Problem Definition

Let $B = \{b_1, b_2, ..., b_m\}$ be a set of m rectangular modules whose width, height, and area are denoted by W_i, H_i, and A_i, $1 \leq i \leq m$. Each module is free to rotate. Let (x_i, y_i) denote the coordinate of the bottom-left corner of rectangle b_i, $1 \leq i \leq m$, on a chip. A placement $\mathcal{P}$ is an assignment of (x_i, y_i) for each b_i, $1 \leq i \leq m$, such that no two modules overlap. The goal of floorplanning/placement is to minimize the area (i.e., the minimum bounding rectangle of $\mathcal{P}$) and wirelength (i.e., the summation of half bounding box of interconnections) induced by a placement.

3 Transitive Closure Graph (TCG)

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 $G\}$. The *Transitive Closure Graph* (*TCG*) representation 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 this section, we first introduce the procedure for constructing C_h and C_v from a placement. Then, we describe how to pack modules from TCG. In the last subsection, we discuss the properties and the solution space of TCG.

3.1 From a placement to its TCG

For two non-overlapped modules b_i and b_j, b_i is said to be *horizontally* (*vertically*) *related* to b_j, denoted by $b_i \vdash b_j$ ($b_i \perp b_j$), if b_i is on the left (bottom) side of b_j and their projections on the y (x) axis overlap. (Note that two modules cannot have both horizontal and vertical relations unless they overlap.) For two non-overlapped modules b_i and b_j, b_i is said to be *diagonally related* to b_j if b_i is on the left side of b_j and their projections on the x and the y axes do not overlap. In a placement, every two modules must bear one of the three relations: *horizontal relation*, *vertical relation*, and *diagonal relation*. To simplify the operations on geometric relations, we treat a diagonal relation for modules b_i and b_j as a horizontal one, unless there exists a chain of vertical relations from b_i (b_j), followed by the modules enclosed with the rectangle defined by the two closest corners of b_i and b_j, and finally to b_j (b_i), for which we make $b_i \perp b_j$ ($b_j \perp b_i$).

Figure 2(a) shows a placement with five modules a, b, c, d, and e whose widths and heights are (6, 4), (4, 6), (7, 4), (6, 3) and (3, 2), respectively. In Figure 2(a), $a \vdash b$, $a \perp c$, and module e is diagonally related to module b. There exists a chain of vertical relations formed by modules e, c, and b between the two modules e and b (i.e., $b \perp c$ and $c \perp e$). Therefore, we make $b \perp e$. Also, module e is diagonally related to module d. However, there dose not exist a chain of vertical relations between modules e and d, and thus we make $e \vdash d$.

TCG can be derived from a placement as follows. For each module b_i in a placement, we introduce a node n_i with the weight being the width (height) in C_h (C_v). If $b_i \vdash b_j$, we construct a directed edge from node n_i to node n_j (denoted by (n_i, n_j)) in C_h. Similarly, we construct a directed edge (n_i, n_j) in C_v if $b_i \perp b_j$. Given a placement with m modules, we need to perform the above process $m(m-1)/2$ times to capture all the geometric relations among modules (i.e., C_h and C_v have $m(m-1)/2$ edges in total).

As shown in Figure 2(b), for each module b_i, $i \in \{a, b, c, d, e\}$, we introduce a node n_i in C_h and also in C_v. For each node n_i in C_h (C_v), $i \in \{a, b, c, d, e\}$, we associate the node with a weight equal to the width (height) of the corresponding module b_i. Since $b_a \vdash b_b$, we construct a directed edge (n_a, n_b) in C_h. Similarly, we construct a directed edge (n_a, n_c) in C_v since $b_a \perp b_c$. This process is repeated until all geometric relations among modules are defined. As shown in Figure 2(b), each transitive closure graph has five nodes, and there are totally 10 edges in C_h and C_v (four in C_h and six in C_v). From the TCG representation shown in Figure 2(b), we know that module e

Representation	SP [6]	FAST-SP [14]	BSG [7]	O-tree [2]	B*-tree [1]	CBL [3]	TCG
Is P-admissible?	Yes	Yes	Yes	No	No	No	Yes
Need Sequence encoding?	Yes	Yes	No	Yes	No	Yes	No
Construct constraint graphs for packing?	Yes	No	Yes	Yes	No	Yes	No
Evaluate cost directly on representation?	No	No	No	No	Yes	No	Yes
Geometric relation is transparent to its operations?	No	No	No	No	No	No	Yes
Boundary information on representation?	No	No	No	No	No	No	Yes
Runtime for packing	$O(m^2)$	Amortized $O(m \lg\lg m)$	$O(m^2)$	Amortized $O(m)$	Amortized $O(m)$	Amortized $O(m)$	$O(m^2)$

Table 1: Properties of representations. Here, m is the number of modules in a placement.

is above module b because there exists a directed edge (n_b, n_e) in C_v; notice that this relation cannot be directly derived from the O-tree and B*-tree shown in Figure 1. Further, we also know directly from the TCG that module d is right to module a while the relationship is not known to O-tree and B*-tree until modules are placed. For boundary information, we know that modules a, c and e (b and d) are on the left (right) boundary since the in-degrees (out-degrees) of n_a, n_c and n_e (n_b and n_d) are zero in C_h. Similarly, it is easy to know from C_v that modules a and b (e and d) are on the bottom (top) boundary. Therefore, the floorplan/placement design with boundary constraints can be handled easily by checking the degree of a node during each perturbation.

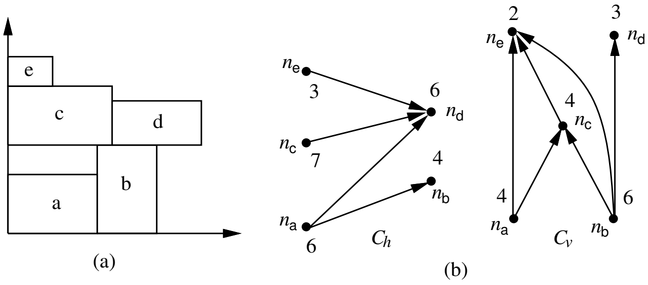

Figure 2: (a) A placement in a chip. (b) TCG.

3.2 From a TCG to its placement

We have introduced how to derive a TCG from its placement in the previous section. We now present the packing method for a TCG.

Given a TCG, its corresponding placement can be obtained in $O(m^2)$ time by performing a well-known *longest path algorithm* [5] on the TCG, where m is the number of modules. To facilitate the implementation of the longest path algorithm, we augment the given two closure graphs as follows. (Note that the TCG augmentation is performed only for packing. It will be clear later that such augmentation is not needed for other operations such as solution perturbation.) We introduce two special nodes with zero weights for each closure graph, the source n_s and the sink n_t, and construct an edge from n_s to each node with in-degree equal to zero, and also from each node with out-degree equal to zero to n_t. Figure 3 shows the augmented TCG for the TCG shown in Figure 2(b).

Let $L_h(n_i)$ ($L_v(n_i)$) be the length 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 ($L_h(n_i)$, $L_v(n_i)$). 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 $L_h(n_t)L_v(n_t)$.

3.3 Properties of TCG

The TCG has the following properties:

Property 1 *(TCG Feasibility Conditions)*

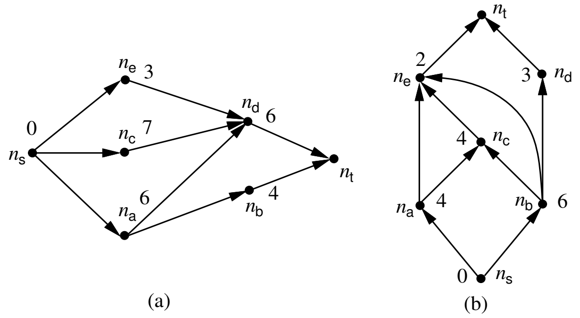

Figure 3: Augmented TCG. (a) Augmented C_h. (b) Augmented C_v.

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.*

Property 1 ensures that a module b_i cannot be both left and right to (below and above) another module b_j in a placement. Property 2 guarantees that no two modules overlap since each pair of modules have exactly one of the horizontal or vertical relation. Property 3 is used to eliminate redundant solutions. It guarantees that if there exists a path from n_i to n_j in one closure graph, the edge (n_i, n_j) must also appear in the same closure graph. For example, there exist two edges (n_i, n_j) and (n_j, n_k) in C_h, which means that $b_i \vdash b_j$ and $b_j \vdash b_k$, and thus $b_i \vdash b_k$. If the edge (n_i, n_k) appears in C_v instead of in C_h, b_k is not only left to b_i but also above b_i. The resulting area of the corresponding placement must be larger or equal to that when the edge (n_i, n_k) appears in C_h. Figure 4 illustrates this phenomenon. In Figure 4(a), there exists a path from n_a to n_e in c_v, which consists of (n_a, n_c) and (n_c, n_e). Thus, the edge (n_a, n_e) must also belong to C_v. If the edge (n_a, n_e) appears in C_h instead of in C_v as shown in Figure 4(c), the resulting area of the placement must be larger or equal to the configuration of Figures 4(a) and (b). Property 3 eliminates such a redundancy.

Based on the properties of TCG, we have the following theorems.

Theorem 1 *There exists a unique placement corresponding to a TCG.*

Theorem 2 *The size of the solution space for TCG is $(m!)^2$, where m is the number of modules.*

Theorem 3 *TCG is P-admissible.*

4 Floorplanning Algorithm

We develop a simulated annealing based algorithm [4] using TCG for non-slicing floorplan design. Given an initial solution represented by a TCG, the algorithm perturbs the TCG to obtain a new

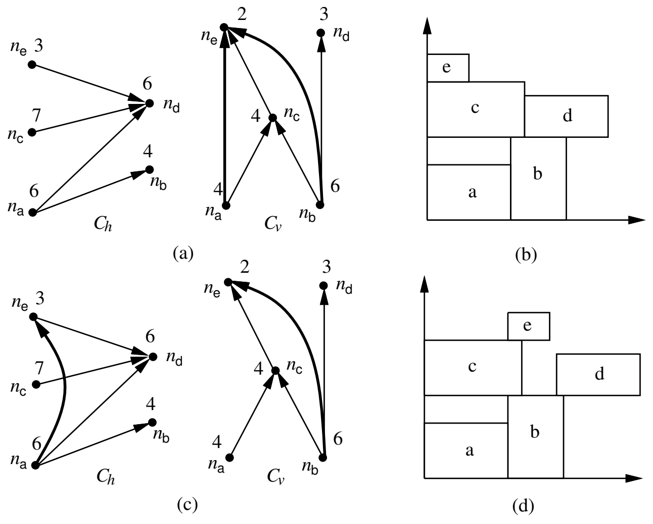

Figure 4: Examples of the TCG feasibility. (a) A feasible TCG. (b) The placement corresponding to the TCG shown in (a). (c) A non-TCG. (d) The placement corresponding to the non-TCG shown in (c).

TCG. To ensure the correctness of the new TCG, as described in the previous section, the new TCG must satisfy the aforementioned three feasibility properties. To identify feasible TCG for perturbation, we introduce the concept of *transitive reduction edges* of TCG in the following section.

4.1 Transitive Reduction Edges

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, in the C_v of Figure 4(a), the edges (n_a, n_c), (n_b, n_c), (n_c, n_e), and (n_b, n_d) are reduction edges while (n_a, n_e) and (n_b, n_e) are closure ones since there exist respective paths $< n_a, n_c, n_e >$ and $< n_b, n_c, n_e >$ from n_a to n_e and from n_b to n_e.

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 $n_j \in F_{out}(n_i)$. Here, we define the fan-in (fan-out) of a node n_i, denoted by $F_{in}(n_i)$ ($F_{out}(n_i)$), as the nodes n_j's with edges (n_j, n_i) ((n_i, n_j)). 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_v shown in Figure 4(a), $F_{out}(n_a) = \{n_c, n_e\}$. Since n_e belongs to $F_{out}(n_c)$, edge (n_a, n_e) is a closure edge while (n_a, n_c) is a reduction one.

Lemma 1 *Given an arbitrary node n_i in one transitive closure graph, for nodes $n_k, n_l \in F_{out}(n_i)$, the edge (n_i, n_k) cannot be a reduction edge if $n_k \in F_{out}(n_l)$.*

Theorem 4 *Given a node n_i in C_h or C_v, it takes $O(m^2)$ time to find a reduction edge (n_i, n_j), where m is the number of modules.*

4.2 Solution Perturbation

We apply the following four operations to perturb a TCG:

- *Rotation*: Rotate a module.
- *Swap*: Swap two nodes in both of 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.

Rotation and Swap do not change the topology of a TCG while Reverse and Move do. To maintain the properties of the TCG after performing the Reverse and Move operations, we may need to update the resulting graphs. We detail the four operations as follows.

4.2.1 Rotation

To rotate a module b_i, we only need to exchange the weights of the corresponding node n_i in C_h and C_v. Figure 5(b) shows the resulting C_h, C_v, and placement after rotating the module d shown in Figure 5(a). Notice that the weights associated with the node n_d in C_h and C_v have been exchanged.

4.2.2 Swap

To swap two nodes n_i and n_j, we only need to exchange two nodes in both C_h and C_v. Figure 5(c) shows the resulting C_h, C_v, and placement after swapping the nodes n_a and n_b shown in Figure 5(b). Notice that the nodes n_a and n_b in both C_h and C_v have been exchanged.

4.2.3 Reverse

The Reverse operation reverses the direction of a *reduction* edge (n_i, n_j) in a transitive closure graph, which corresponds to changing the geometric relation of the two modules b_i and b_j. 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 $b_j \vdash b_i$ ($b_j \perp b_i$). Therefore, the geometric relation among modules is transparent not only to the TCG representation but also to the Reverse *operation* (i.e., the effect of such an operation on the change of the geometric relation is known *before* packing); this property can facilitate the convergence to a desired solution.

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 edges (n_k, n_l) (or (n_l, n_k)) in the other transitive closure graph, if any, to maintain the properties of the TCG.

Figure 5(d) shows the resulting C_h, C_v, and placement after reversing the *reduction edge* (n_c, n_e) of the C_v shown in Figure 5(c). In the C_v of Figure 5(d), $F_{in}(n_e) \cup \{n_e\} = \{n_a, n_b, n_e\}$ and $F_{out}(n_c) \cup \{n_c\} = \{n_c\}$. For each node $n_k \in F_{in}(n_e) \cup \{n_e\}$ and $n_l \in F_{out}(n_c) \cup \{n_c\}$, we check whether the edge (n_k, n_l) exists. Since the edge (n_e, n_c) was just added to C_v and the edges (n_b, n_c) and (n_a, n_c) already exist in C_v of Figure 5(c), we do not need to add the three edges to C_v. Neither do we need to update the C_h of Figure 5(c). Notice that by reversing the reduction edge (n_c, n_e) in the C_v, we transform the relation $b_c \perp b_e$ into $b_e \perp b_c$ in the resulting placement of Figure 5(d).

To maintain the properties of a TCG, we can only reverse a reduction edge. For example, if we reverse a closure edge (n_i, n_k) associated with the two reduction edges (n_i, n_j) and (n_j, n_k), a cycle $< n_i, n_j, n_k, n_i >$ is formed, and thus the resulting graphs are no longer a TCG. Further, for each edge introduced in a transitive closure graph, we remove its corresponding edge from the other graph. Therefore, there is always exactly one relation between each pair of modules.

4.2.4 Move

The Move operation moves a *reduction* edge (n_i, n_j) in a transitive closure graph to the other, which corresponds to switching 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 $b_i \perp b_j$ ($b_i \vdash b_j$). Therefore, the geometric relation among modules is also transparent to the Move operation.

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 the edge from G and add it 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.

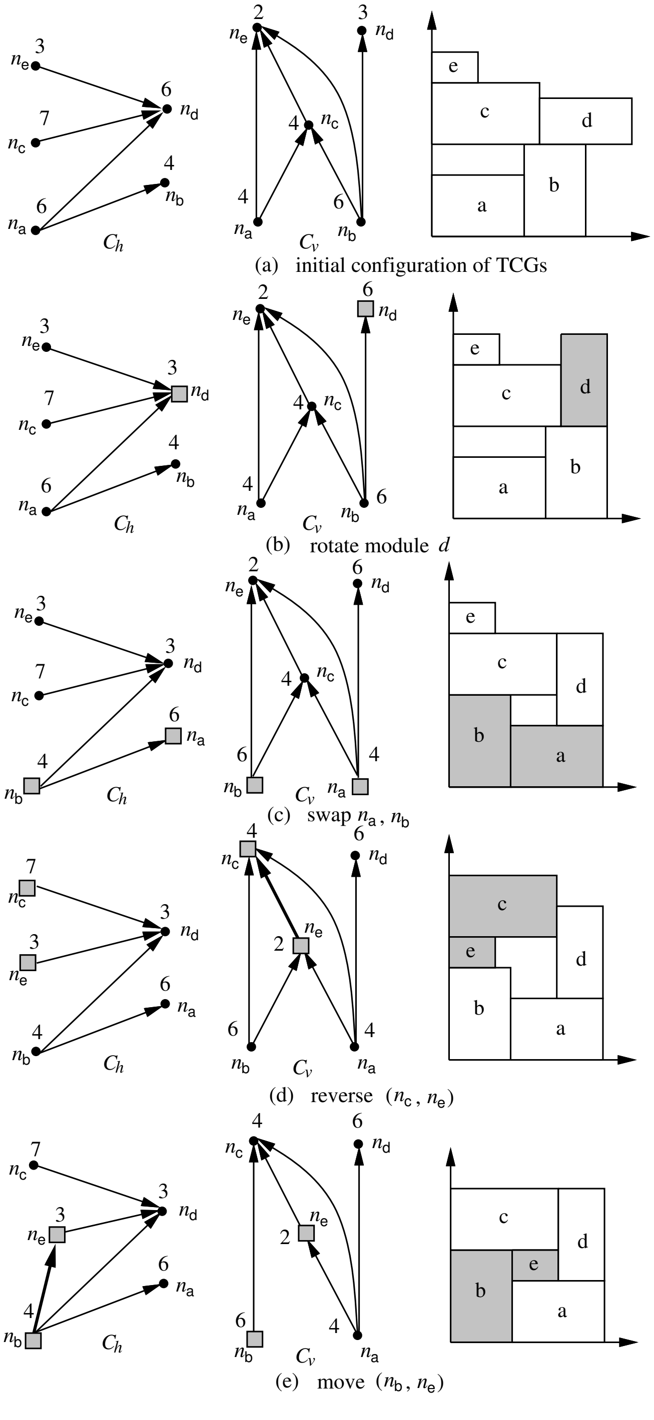

Figure 5: Four types of perturbation. (a) The initial TCG (C_h and C_v) and placement. (b) The resulting TCG and placement after rotating the module d shown in (a). (c) The resulting TCG and placement after swapping the nodes n_a and n_b shown in (b). (d) The resulting TCG and placement after reversing the reduction edge (n_c, n_e) shown in (c). (e) The resulting TCG and placement after moving the reduction edge (n_b, n_e) from the C_v of (d) to C_h.

Circuit	# of modules	# of I/O pads	# of nets	# of pins
apte	9	73	97	214
xerox	10	107	203	696
hp	11	43	83	264
ami33	33	42	123	480
ami49	49	24	408	931

Table 2: The five MCNC benchmark circuits.

Circuit	O-tree		enhanced O-tree		TCG	
	Wire (mm)	Time (sec)	Wire (mm)	Time (sec)	Wire (mm)	Time (sec)
apte	317	47	317	15	363	2
xerox	368	160	372	39	366	15
hp	153	90	150	19	143	10
ami33	52	2251	52	177	44	52
ami49	636	14112	629	688	604	767
Comparison	+3.56%	-	+3.18%	-	0.00%	-

Table 4: Wirelength and runtime comparisons among O-tree (on SUN Ultra60), enhanced O-tree (on Sun Ultra60), and TCG (on Sun Ultra60) for wirelength optimization.

Figure 5(e) shows the resulting C_h, C_v, and placement after moving the *reduction edge* (n_b, n_e) in the C_v of Figure 5(d) to C_h. In the C_h shown in Figure 5(e), $F_{in}(n_b) \cup \{n_b\} = \{n_b\}$ and $F_{out}(n_e) \cup \{n_e\} = \{n_d, n_e\}$. For each node $n_k \in F_{in}(n_b) \cup \{n_b\}$ and $n_l \in F_{out}(n_c) \cup \{n_c\}$, we check whether the edge (n_k, n_l) exists in G_h. Since the edge (n_b, n_e) was just added to C_h and the edge (n_b, n_d) already exists in C_h of Figure 5(d), we need to do nothing (except moving the reduction edge (n_b, n_e) from C_v to C_h). Neither do we need to update C_v (except removing the edge (n_b, n_e)). Notice that by moving the reduction edge (n_b, n_e) from C_v to C_h, we transform the relation $b_b \perp b_e$ into $b_b \vdash b_e$ in the resulting placement shown in Figure 5(e).

To maintain the properties of a TCG, we can only move a reduction edge. If we move a closure edge (n_i, n_k) associated with the two reduction edges (n_i, n_j) and (n_j, n_k) in one transitive closure graph to the other, then there exist a path from n_i to n_k in the two graphs, implying that $b_i \vdash b_k$ and $b_i \perp b_k$, which gives a redundant solution. Further, for each edge introduced in a transitive closure graph, we remove its corresponding edge from the other graph. Therefore, there is always exactly one relation between each pair of modules.

Theorem 5 *TCG is closed under the rotation, swap, reverse, and move operations, and each of the operations takes respective $O(1), O(1), O(m^2)$, and $O(m^2)$ time, where m is the number of modules in the placement.*

5 Experimental Results

Based on a simulated annealing method [4], we implemented the TCG representation in the C++ programming language on a 433 MHz SUN Sparc Ultra-60 workstation with 1 GB memory. We compared TCG with O-tree [2], B*-tree [1], enhanced O-tree [12], and CBL [3] based on the five MCNC benchmark circuits listed in Table 2. Columns 2, 3, 4, and 5 of Table 2 list the respective numbers of modules, I/O pads, nets, and pins of the five circuits.

The experiments consist of three parts: area optimization, wirelength optimization, and simultaneous area and wirelength optimization. The area of a placement is measured by that of the minimum bounding box enclosing the placement. The area and runtime comparisons among O-tree [2], B*-tree [1], enhanced O-tree [12], CBL [3], and TCG are listed in Table 3. As shown in Table 3, TCG achieves average improvements of 2.22%, 1.18%, 2.04%, and 3.54% in area utilization compared to O-tree, B*-tree, enhanced O-tree, and CBL, respectively. The runtimes are significantly smaller than O-tree and B*-tree, and comparable to the enhanced O-tree [12]. Figure 6

Circuit	O-tree		B*-tree		enhanced O-tree		CBL		TCG	
	Area (mm^2)	Time (sec)	Area (mm^2)	Time (sec)	Area (mm^2)	Time (sec)	Area (mm^2)	Time (sec)	Area (mm^2)	Time (sec)
apte	47.1	38	46.92	7	46.92	11	NA	NA	46.92	1
xerox	20.1	118	19.83	25	20.21	38	20.96	30	19.83	18
hp	9.21	57	8.947	55	9.16	19	-	-	8.947	20
ami33	1.25	1430	1.27	3417	1.24	118	1.20	36	1.20	306
ami49	37.6	7428	36.80	4752	37.73	406	38.58	65	36.77	434
Comparison	+2.22%	-	+1.18%	-	+2.04%	-	+3.54%	-	0.00%	-

Table 3: Area and runtime comparisons among O-tree (on SUN Ultra60), B*-tree (on Sun Ultra-I), enhanced O-tree (on Sun Ultra60), CBL (on Sun Sparc 20), and TCG (on Sun Ultra60) for area optimization.

Circuit	O-tree			enhanced O-tree			CBL			TCG		
	Area (mm^2)	Wire (mm)	Time (sec)	Area (mm^2)	Wire (mm)	Time (sec)	Area (mm^2)	Wire (mm)	Time (sec)	Area (mm^2)	Wire (mm)	Time (sec)
apte	51.92	320.7	47	51.95	320.7	14	-	-	NA	48.48	378.0	49
xerox	20.42	380.6	142	20.42	380.6	41	20.233	403.47	NA	20.42	385.0	114
hp	9.490	152.6	84	9.384	151.9	21	NA	NA	NA	9.490	151.8	59
ami33	1.283	51.31	2349	1.299	52.13	205	1.226	51.67	NA	1.237	50.29	939
ami49	39.55	688.7	15318	39.92	702.8	700	38.378	732.84	NA	38.20	663.1	3613
Comparison	+2.87%	-2.01%	-	+3.33%	-1.34%	-	-0.45%	+5.98%	-	0.00%	0.00%	-

Table 5: Area, wirelength, and runtime comparisons among O-tree (on SUN Ultra60), enhanced O-tree (on Sun Ultra60), CBL (on Sun Sparc 20), and TCG (on Sun Ultra60) for simultaneous area and wirelength optimization.

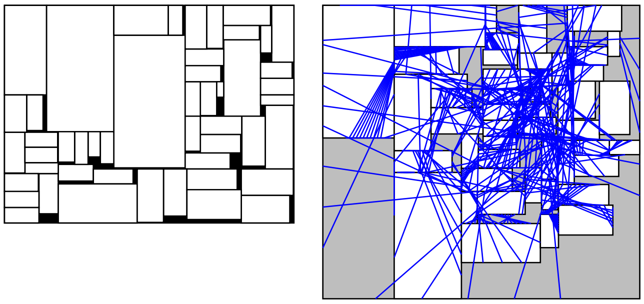

Figure 6: Resulting placements of ami49 for (1) left: optimizing area alone (area $= 36.77mm^2$); (2) right: optimizing wirelngth alone (wire $= 604mm$).

(left) shows the resulting placement for ami49 with area optimization.

For wirelength optimization, we estimated the wirelength of a net by half the perimeter of the minimum bounding box enclosing the net. The wirelength of a placement is given by the summation of the wirelengths of all nets. The comparisons with the previous works are listed in Table 4. (Note that B*-tree and CBL did not report the results on optimizing wirelength alone.) As shown in Table 4, TCG achieves average reductions of 3.56% and 3.18% in wirelength, compared to the O-tree and the enhanced O-tree, respectively. Figure 6 (right) shows the resulting placement for ami49 with wirelength optimization. For simultaneous area and wirelength optimization, we assigned the same weight for area and wirelength in the cost function. The results are listed in Table 5, which shows that ours are slightly better than previous works.[1]

6 Concluding Remarks

We have presented the TCG representation for non-slicing floorplans and shown its superior properties. Experimental results have shown that TCG is very effective in area and wirelength optimization. As revealed in the representation, TCG keeps the information of boundary modules as well as the shapes and the relative positions of modules. These properties make TCG a promising choice for dealing with the general floorplan/placement problems with various constraints, such as boundary constraint. Research along this direction is ongoing.

[1] We excluded the CBL results for hp in Table 3 and for apte in Table 5 in the comparisons since the CBL test cases may not be the same as others. For example, CBL reports an area of 66.14 mm^2 for hp, which is about seven times larger than others.

References

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

[2] P.-N. Guo, C.-K. Cheng, and T. Yoshimura, "An O-Tree representation of non-slicing floorplan and its applications,"*Proc. DAC*, pp. 268–273, June 1999.

[3] 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 non-slicing floorplan," *Proc. ICCAD*, pp. 8–12, Nov. 2000.

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

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

[6] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, "Rectangle-packing based module placement," *Proc. ICCAD*, pp. 472–479, Nov. 1995.

[7] S. Nakatake, K. Fujiyoshi, H. Murata, and Y. Kajitani, "Module placement on BSG-structure and IC layout applications," *Proc. ICCAD*, pp. 484–491, Nov. 1996.

[8] T. Ohtsuki, N. Suzigama, and H. Hawanishi, "An optimization technique for integrated circuit layout design," *Proc. ICCST*, *Kyoto*, pp. 67–68, 1970.

[9] H. Onodera, Y. Taniquchi, and K. Tamaru, "Branch-and-bound placement for building block layout," *Proc. DAC*, pp. 433–439, 1991.

[10] R.H.J.M. Otten, "Automatic floorplan design,"*Proc. DAC*, pp.261–267, June 1982.

[11] P. Pan and C.-L. Liu, "Area minimization for floorplans," *IEEE TCAD*, pp. 123–132, Jan. 1995.

[12] Y.-Pang, C.-K. Cheng, and T. Yoshimura, "An enhanced perturbing algorithm for floorplan design using the O-tree representation," *Proc. ISPD*, pp. 168-173, April 2000.

[13] S. M. Sait and H. Youssef, *VLSI Physical Design Automation,* IEEE Press, 1995.

[14] X. Tang and D. F. Wong, " FAST-SP: A fast algorithm for block placement based on sequence pair," *Proc. ASP-DAC*, pp. 521–526, Jan. 2001.

[15] T.-C. Wang, and D. F. Wong, "An optimal algorithm for floorplan and area optimization,"*Proc. DAC*, pp.180–186, June 1990.

[16] D. F. Wong, and C.-L. Liu, "A new algorithm for floorplan design," *Proc. DAC*, pp. 101–107, June 1986.