TCG-S: Orthogonal Coupling of P^*-Admissible Representations for General Floorplans

Jai-Ming Lin and Yao-Wen Chang

***Abstract*—In this paper, we extend the concept of the P-admissible floorplan representation to that of the P^*-admissible one. A P^*-admissible representation can model the most general floorplans. Each of the currently existing P^*-admissible representations, sequence pair (SP), bounded-slicing grid, and transitive closure graph (TCG), has its strengths as well as weaknesses. We show the equivalence of the two most promising P^*-admissible representations, TCG and SP, and integrate TCG with a packing sequence (part of SP) into a representation, called *TCG-S*. TCG-S combines the advantages of SP and TCG and at the same time eliminates their disadvantages. With the property of SP, a fast packing scheme is possible. Inherited nice properties from TCG, the geometric relations among modules are transparent to TCG-S (implying faster convergence to a desired solution), placement with position constraints becomes much easier, and incremental update for cost evaluation can be realized. These nice properties make TCG-S a superior representation which exhibits an elegant solution structure to facilitate the search for a desired floorplan/placement. Extensive experiments show that TCG-S results in the best area utilization, wirelength optimization, convergence speed, and stability among existing works and is very flexible in handling placement with special constraints.**

***Index Terms*—Floorplanning, layout, physical_design, transitive_closure_graph.**

I. Introduction

As technology advances, the circuit size in modern very large scale integration (VLSI) design increases dramatically. To handle the increasing design complexity, hierarchical designs and IP modules are widely used to optimize area and timing for design convergence. Further, the need to integrate heterogeneous systems or special modules imposes some placement constraints, e.g., the boundary-module constraint which requires some modules to be placed along the chip boundaries for shorter connections to pads, the preplaced-module constraint which preassigns modules to specific positions, etc. These trends make floorplanning/placement much more important than ever, and it is of particular significance to consider the floorplanning/placement with various constraints. To cope with these challenges, it is desired to develop an efficient and effective floorplan representation that can model the geometric relations among regular as well as constrained modules.

Many floorplan representations have been proposed in the literature, e.g., slicing tree [16], normalized Polish expression [19], sequence pair (SP) [13], bounded-slicing grid (BSG) [15], O-tree [3], B^*-tree [1], corner block list (CBL) [4], and transitive closure graph (TCG) [9]. Unlike the traditional classification of the slicing and nonslicing structures, we can alternatively classify them into two categories, *P^*-admissible* and *non-P^*-admissible* representations. A representation is said to be P-admissible if it satisfies the following four conditions [13]: 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. We extend in this paper the concept of the P-admissible representation to that of the P^*-admissible one by adding the fifth condition: 5) both horizontal and vertical [and thus two dimensional (2-D)] geometrical information between modules are defined in the representation. With this condition, any placement can be modeled. Therefore, a P^*-admissible representation can represent the most general floorplans and contains a complete structure for searching for an optimal floorplan/placement solution. It is thus desirable to develop an effective and flexible P^*-admissible representation.

Among the existing popular representations, SP, BSG, and TCG are P^*-admissible while slicing tree, normalized Polish expression (NPE), O-tree, B^*-tree, and CBL are not. The slicing tree and normalized Polish expression are intended for slicing floorplans only. Since an optimal placement could be a nonslicing structure, the two representations are not P^*-admissible [i.e., violation of P^*-admissible Condition (4)]. An O-tree defines only one-dimensional geometrical relation between *compacted* modules and thus can obtain the relation in the other dimension only after packing [i.e., violation of Condition (5)]. [Note that O-tree is undefined for some uncompacted placements which may correspond to the best solutions for wirelength optimization. Therefore, as far as wirelength optimization is concerned, O-tree is not even P-admissible since Condition (4) is violated.] A B^*-tree requires a placement to be left and/or bottom compacted. If the placement after a tree packing is not compacted, a sequence of compaction operations are applied to make all modules compacted to the left and/or bottom. However, the space intended for placing a module may be occupied by previously placed modules during packing, resulting in a mismatch between the original representation and its *compacted* placement. Therefore, it may not be feasible to find a *compacted* placement corresponding to the original B^*-tree, and thus it is not P^*-admissible [i.e., violation of Condition (2)]. CBL can represent only *mosaic* floorplans, in which each region in the floorplan contains exactly one module. CBL is not P^*-admissible because it cannot guarantee a feasible solution after a perturbation [i.e., violation of Condition (2)]. Non-P^*-admissible representations intrinsically have a smaller solution space and lower packing cost since their corresponding floorplanning/placement structures are more restricted [e.g., slicing structures (slicing tree, NPE), compacted placements (O-tree, B^*-tree), mosaic floorplans (CBL), etc.]. However, lack of the guarantee in the feasibility and/or the optimality of their representations would inevitably lead to longer running times and/or lower solution quality.

The existing P^*-admissible representations, SP, BSG, and TCG, have their own distinct properties as well as common ones. Nevertheless, researchers tend to favor SP over BSG because BSG incurs many redundancies and thus a much larger solution space, implying a longer running time to search for a good solution. Therefore, we shall focus on SP and TCG. Both SP and TCG are considered very flexible representations and construct constraint graphs to evaluate their packing cost. SP consists of two sequences of modules (Γ_+, Γ_-), where Γ_+ specifies the module ordering from top-left to bottom-right and Γ_- from bottom-left to top-right. Hence, Γ_- corresponds to the ordering for packing modules to the bottom-left direction and thus can be used to guide module packing. However, like most existing representations (e.g., NPE, BSG, O-tree, B^*-tree, CBL) the geometric relations between modules are *not transparent* to the operations of SP (i.e., the effect of an operation on the change of module relation is not clear before packing) and, thus, we need to construct constraint graphs from scratch after each perturbation to evaluate the packing cost; this deficiency makes SP harder to converge to a desired solution and to handle placement with constraints (e.g., boundary modules, preplaced modules, etc.).

Manuscript received October 21, 2002; revised February 2, 2003 and May 18, 2003. This work was supported in part by the National Science Council of Taiwan R.O.C. under Grant No. NSC-89-2215-E-009-117. This paper was recommended by Associate Editor M. D. F. Wong.

J.-M. Lin is with the Department of Computer and Information Science, National Chiao Tung University, Hsinchu 300, Taiwan (e-mail: gis87808@cis.nctu.edu.tw).

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

Digital Object Identifier 10.1109/TCAD.2004.828114

TCG consists of a horizontal transitive closure graph C_h to define the horizontal geometric relations between modules and a vertical one C_v for vertical geometric relations. Contrast to SP, the geometric relations between modules are transparent to TCG as well as its operations, facilitating the convergence to a desired solution. 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. Nevertheless, like SP, constraint graphs are also needed for TCG to evaluate its packing cost, and unlike SP, we need to perform extra operations to obtain the module packing sequence.

Therefore, an interesting question arises. Is it possible to develop a representation that can combine the advantages of SP and TCG and at the same time eliminate their disadvantages? We answer this question in affirmation by showing the equivalence of TCG and SP, and integrating them into $\mathrm{TCG-S} = (C_h, C_v, \Gamma_-)$. The orthogonal combination leads to a representation with at least the following advantages:

- With the property of SP, a fast $O(m \lg m)$-time packing scheme is possible for a P^*-admissible representation, where m is the number of modules. (Note that a linear-time packing scheme is possible for the tree-based representations, but they can represent only more restricted compacted floorplans.)
- It is clear later that the sequence Γ_- is the topological order of both C_h and C_v and is thus uniquely determined by C_h and C_v. Therefore, the combination will not incur any redundancy over the original TCG, and the solution space of TCG-S is still $(m!)^2$ (same as TCG and SP), where m is the number of modules.
- Inherited from TCG, the geometric relations among modules are transparent to TCG-S, implying faster convergence to a desired solution.
- Inherited from TCG, placement with position constraints becomes much easier.
- Inherited from TCG, TCG-S can support incremental update for cost evaluation.

These nice properties make TCG-S an effective, efficient, and flexible representation. Extensive experiments based on a set of commonly used Microelectronics Center of North Carolina (MCNC) benchmarks show that TCG-S results in the best area utilization, wirelength optimization, convergence speed, and solution stability among existing works. (Note that we also consider the convergence speed and stability to eliminate the possible unfairness due to the nondeterministic behavior of simulated annealing, which were neglected in most previous works.)

To show the flexibility of TCG-S, we also consider placement with preplaced and boundary modules. For placement with preplaced modules, Murata *et al.* [14] proposed an *adaptation algorithm* to transform an infeasible SP with preplaced modules into a feasible one. However, the process incurs expensive computations. For placement with the boundary-module constraint, Tang and Wong in [18] handled the constraint by adding dummy edges into the constraint graphs of SP. Ma *et al.* in [12] assigned a penalty to a misplaced boundary module and perturbed the CBL to reduce the penalty. However, both [18] and [12] cannot guarantee a feasible solution in each perturbation and their final placements. Lai *et al.* in [7] gave the feasibility conditions for SP with boundary modules and transformed an infeasible solution into a feasible one. However, the method is very complex, and many rules are needed to cope with the constraints.

In this paper, we also present the methods for handling placement with preplaced and boundary modules. Different from the previous works on boundary modules that cannot guarantee a feasible solution or need to transform an infeasible solution into a feasible one, TCG-S can easily maintain the feasibility during each perturbation. We compared our work with [7] on placement with boundary modules. (Note that there are no common benchmark circuits for this constraint.)

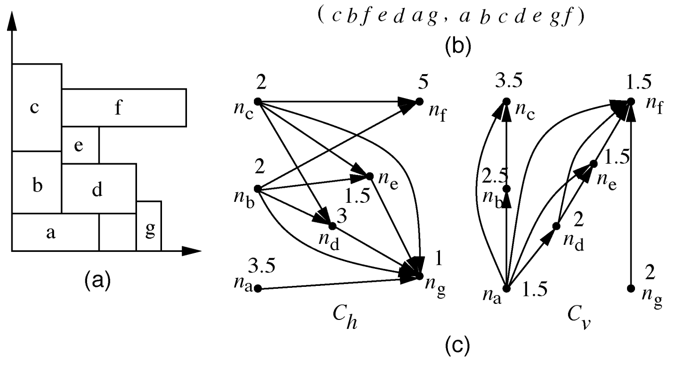


Fig. 1. (a) A placement. (b) The corresponding SP of (a). (c) The corresponding TCG of (b).

Experimental results show that TCG-S results in smaller areas than [7].

The remainder of this paper is organized as follows. Section II formulates the floorplan/placement design problem. Section III compares SP and TCG. Section IV presents the procedures to build the TCG-S from a placement and construct the placement from a TCG-S. Section V gives the operations to perturb a TCG-S. Section VI presents our methods to handle placement with boundary and preplaced modules. Experimental results are reported in Section VII. Finally, we give concluding remarks in Section VIII.

II. Problem Definition

Let $B = \{b_1, b_2, \ldots, b_m\}$ be a set of m rectangular modules whose width, height, and area are denoted by W_i, H_i, and A_i, $1 \leq i \leq m$. Let (x_i, y_i) $((x'_i, y'_i))$ denote coordinate of the bottom-left (top-right) corner of module 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 optimize the area (i.e., the minimum bounding rectangle of $\mathcal{P}$) and/or the wirelength (i.e., the summation of half bounding box of interconnections) induced by the assignment of b_i's on the chip.

III. P^*-Admissible Representations

In this section, we first review the two P^*-admissible representations, TCG and SP, then show their equivalence, and compare their properties.

A. Review of TCG and SP

TCG describes the geometric relations between 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* [9]:

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]

Fig. 1(a) shows a placement with seven modules a, b, c, d, e, f, and g whose widths and heights are (3.5, 1.5), (2, 2.5), (2, 3.5), (3, 2), (1.5, 1.5), (5, 1.5), and (1, 2), respectively. Fig. 1(c) shows the $\mathrm{TCG} = (C_h, C_v)$ corresponding to the placement of Fig. 1(a). The

[1]The transitive closure of a directed acyclic graph $\boldsymbol{G}$ is defined as the graph $\boldsymbol{G}' = (\boldsymbol{V}, \boldsymbol{E}')$, where $\boldsymbol{E}' = \{(\boldsymbol{n}_i, \boldsymbol{n}_j)$: **there is a path from node** $\boldsymbol{n}_i$ **to node** $\boldsymbol{n}_j$ **in** $\boldsymbol{G}\}$.

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_g) in C_h, module b_a is left to b_g. Similarly, b_a is below b_b since there exists an edge (n_a, n_b) in C_v.

SP uses a pair of sequences (Γ_+, Γ_-) to represent a floorplan/placement, where Γ_+ and Γ_- give two permutations of module names. The geometric relation of modules can be derived from an SP as follows. Module b_a is left (right) to module b_b if a appears before (after) b in both Γ_+ and Γ_-. Module b_a is below (above) module b_b if b appears before (after) a in Γ_+ and a appears before (after) b in Γ_-. Fig. 1(b) shows the corresponding SP. Since a is before g in both Γ_+ and Γ_-, module b_a is left to module b_g. Similarly, b_a is below b_b since a is after b in Γ_+ and before b in Γ_-.

It is an important observation that the sequence Γ_- is the topological order of both C_h and C_v and is thus uniquely determined by C_h and C_v. For example, as shown in Fig. 1(b) and (c), $\Gamma_- = \langle abcdegf \rangle$ is the topological order of both C_h and C_v. The observation is the key to the nonredundant combination of TCG and SP, which is the theme of this paper.

B. Equivalence of SP and TCG

Like the relations between a skewed slicing tree [16] and an NPE [19] for slicing floorplans as well as O-tree and B*-tree for nonslicing floorplans, TCG and SP are equivalent.

We can transform between TCG and SP as follows: Let the *fanin* (*fanout*) of a node n_i, denoted by $F_{in}(n_i)$ $(F_{\text{out}}(n_i))$, be the nodes n_j's with edges (n_j, n_i) $((n_i, n_j))$. Given a TCG, we can obtain a sequence Γ_+ by repeatedly extracting a node n_i with $F_{in}(n_i) = \emptyset$ in C_h and $F_{\text{out}}(n_i) = \emptyset$ in C_v, and then deleting the edges (n_i, n_j)'s $((n_j, n_i)$'s) from C_h (C_v) until no node is left in C_h (C_v). Similarly, we can transform a TCG into another sequence Γ_- by repeatedly extracting the node n_i with $F_{in}(n_i) = \emptyset$ both in C_v and C_h, and then deleting the edges (n_i, n_j)'s from both C_h and C_v until no node is left in C_h and C_v. Given an $\text{SP} = (\Gamma_+, \Gamma_-)$, we can obtain a unique $\text{TCG} = (C_h, C_v)$ from the two constraint graphs of the SP by removing the source, sink, and associated edges. For example, the SP of Fig. 1(b) is equivalent to the TCG of Fig. 1(c). It is proved in [9] that there exists a one-to-one correspondence between TCG and SP.

C. Comparison Between TCG and SP

Although TCG and SP are equivalent, their properties and induced operations are significantly different. Both SP and TCG are considered very flexible representations and construct constraint graphs to evaluate their packing cost. Γ_- of an SP corresponds to the ordering for packing modules to the bottom-left direction and thus can be used for guiding module packing. However, like most existing representations, the geometric relations among modules are not transparent to the operations of SP (i.e., the effect of an operation on the change of module relation is not clear before packing), and thus we need to construct constraint graphs from scratch after each perturbation to evaluate the packing cost; this deficiency makes SP harder to converge to a desired solution and to handle placement with constraints (e.g., boundary modules, preplaced modules, etc).

Contrast to SP, the geometric relations among modules are transparent to TCG as well as its operations, facilitating the convergence to a desired solution. 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. Unlike SP, nevertheless, we need to perform extra operations to obtain the module packing sequence and an additional $O(m^2)$ time to find a special type of edges, called *reduction edges*, in C_h (C_v) for some operations. (We will define the edges later.)

For both SP and TCG, the packing scheme by applying the longest path algorithm is time-consuming since all edges in the constraint graphs are processed, even though they are not on the longest path. As shown in C_h of Fig. 1(c), if we add a source with zero weight and connect it to those nodes with zero in-degree, the x coordinate of each module can be obtained by applying the longest path algorithm on the resulting directed acyclic graph. Therefore, we have $x_g = \max\{x'_a, x'_b, x'_c, x'_d, x'_e\}$. To reduce the number of modules considered for placing a module, we introduce the concept of a horizontal (vertical) contour, denoted by R_h (R_v). R_h (R_v) is a list of modules b_i's for which there exists no module b_j with $y_j \geq y'_i$ $(x_j \geq x'_i)$ and $x'_j \geq x'_i$ $(y'_j \geq y'_i)$; that is, R_h (R_v) is a list of modules in the horizontal (vertical) contour. For the placement of Fig. 1(a), for example, $R_h = \langle b_c, b_f \rangle$ and $R_v = \langle b_g, b_d, b_e, b_f, b_c \rangle$. To place a new module, we only need to consider the bends (and thus the modules) in the contour, and thus the packing time can be improved.

Suppose we have packed the modules b_a, b_b, b_c, b_d, and b_e based on the sequence Γ_-. Then, the resulting horizontal contour $R_h = \langle b_c, b_e, b_d \rangle$. Keeping R_h, we only need to traverse the contour from b_e, the successor of b_e (in terms of in-order search tree traversal), to the last module b_d, which have a horizontal relation with b_g (since there is an edge (n_d, n_g) in C_h). Thus, we have $x_g = x'_d$. Packing modules in this way, we only need to consider x_e and x_d, and can get rid of the computation for a maximum value, leading to a faster packing scheme. We will show later how to apply a balanced binary tree to implement the contour operation to get a loglinear-time packing scheme.

IV. TCG-S Representation

Combining $\text{TCG} = (C_h, C_v)$ and $\text{SP} = (\Gamma_+, \Gamma_-)$, we develop a representation, called $\text{TCG} - \text{S} = (C_h, C_v, \Gamma_-)$, which uses horizontal and vertical transitive closure graphs as well as the packing sequence Γ_- to represent a placement. With the characteristics of TCG and SP, TCG-S has the following four *feasibility properties*.

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.
4) The packing sequence Γ_- is the topological order of both C_h and C_v.

In this section, we first introduce how to construct Γ_-, C_h, and C_v from a placement. Then, we propose an $O(m \lg m)$-time packing scheme for TCG-S, where m is the number of modules.

A. From a Placement to TCG-S

In this section, we first introduce the procedure to extract Γ_- from a placement, and then construct C_h and C_v according to Γ_-.

For two nonoverlapped 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 left to (below) b_j and their projections on the y (x) axis overlap. For two nonoverlapped modules b_i and b_j, b_i is said to be *diagonally related* to b_j if b_i is left to b_j and their projections on the x and the y axes do not overlap. To simplify the operations on geometric relations, a diagonal relation for modules b_i and b_j is treated as a horizontal one unless there exists a chain of vertical relations from b_i (b_j), followed by the modules overlapped with the rectangle defined by the two closest corners of b_i and b_j, and finally to b_j (b_i), for which it is considered as $b_i \perp b_j$ $(b_j \perp b_i)$.

Given a placement, Γ_- can be extracted as follows. We first extract the module on the bottom-left corner. At each iteration, we extract the left-most unvisited module b with all the modules below b having been

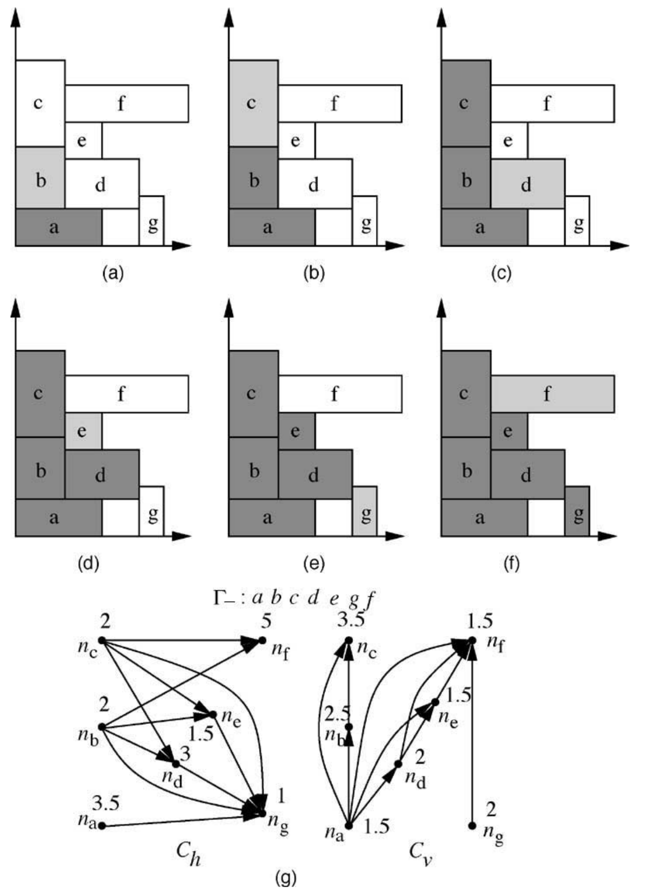


Fig. 2. (a)–(f) Process to extract a $\mathbf{\Gamma_-}$ from the placement of Fig. 1(a). (g) Resulting TCG-S.

extracted. The process repeats until no module left. Fig. 2(a)–(f) illustrate the procedure to extract a Γ_- from the placement of Fig. 1(a). We first extract the module b_a on the bottom-left corner [see Fig. 2(a)], and then b_b since it is the left-module with all the modules below b_b having been extract [see Fig. 2(b)]. This process continues until no module is left, resulting in $\Gamma_- = \langle abcdegf \rangle$.

After extracting Γ_-, we can construct C_h and C_v based on Γ_-. For each module b_i in Γ_-, we introduce a node n_i with the weight being b_i's width (height) in C_h (C_v). Also, for each module b_i before b_j in Γ_-, we introduce an edge (n_i, n_j) in C_h (C_v) if $b_i \vdash b_j$ ($b_i \perp b_j$). As shown in Fig. 2(b) and (g), for the first two modules b_a, b_b in Γ_-, we introduce the nodes n_a and n_b in C_h (C_v) and assign the weights as their widths (heights). Also, we construct a directed edge (n_a, n_b) in C_v since module b_a is before b_b and $b_a \perp b_b$. The process repeats for all modules in Γ_-, resulting in the TCG-S shown in Fig. 2(g). Each transitive closure graph has seven nodes and 21 edges in total (eleven in C_h and ten in C_v). We have the following theorem.

Theorem 1: There exists a unique TCG-S corresponding to a placement.

Proof: TCG-S is composed of three tuples C_h, C_v, and Γ_-. To get Γ_- from a placement, we repeatedly extract the left-most unvisited module b with all modules below b having been visited. The process repeats until all modules have been visited. Since such a module is unique in each iteration, the resulting Γ_- is unique. For each pair of modules b_i and b_j in Γ_-, say $\langle \ldots b_i \ldots b_j \ldots \rangle$, there exists a unique relation ($b_i \perp b_j$ or $b_i \vdash b_j$) between the two modules in the placement based on the definition of TCG. Therefore, there exist unique C_h and C_v corresponding to the placement if we construct a node n_i (or an edge (n_i, n_j)) for each module b_i (or for each pair of modules b_i and b_j based on their geometric relation) in Γ_-.

To show that the 3-tuple C_h, C_v, Γ_- is a TCG-S, we prove that it satisfies the four TCG-S feasibility properties. Properties 1–3 are inherited from TCG. See the proof of Property 1 in [11]. For Γ_-, we repeatedly extract the left- and bottom-most unvisited module b until all modules have been processed. The left-most (bottom-most) module corresponds to the node in C_h (C_v) with in-degree equal to zero. Therefore, the packing sequence Γ_- is the topological order of both C_h and C_v.[2] ■

B. From TCG-S to a Placement

In this section, we propose an $O(m \lg m)$-time packing scheme based on Γ_- as well as a horizontal and a vertical contours R_h and R_v, where m is the number of modules. The basic idea is to process the modules based on the sequence defined in Γ_-, and then pack the current module to a corner formed by two previously placed modules in R_h (R_v) according to the geometric relation defined in C_h (C_v).

We detail the packing scheme as follows. Recall that the horizontal contour R_h (the vertical contour R_v) is a list of modules b_i's for which there exists no module b_j with $y_j \geq y'_i$ ($x_j \geq x'_i$) and $x'_j \geq x'_i$ ($y'_j \geq y'_i$). We can keep the modules b_i's in R_h (R_v) in a balanced binary search tree (e.g., the Red-Black tree [2]) T_h (T_v); the search-tree (in-order traversal) order in T_h (T_v) corresponds to a nondecreasing order of the *right* (*top*) boundaries of the modules in the contour R_h (R_v); i.e., the coordinates of the right (top) boundaries are sorted and kept as the search-tree order in the binary search tree T_h (T_v). By ordering, in the following discussions, we refer to the search-tree order. (It is clear later that this important property leads to an efficient packing algorithm to be introduced shortly.) For easier presentation, we add a *dummy module* b_s (b_t) to R_h (R_v) to denote the left (bottom) boundary module of a placement. We have $b_s \vdash b_i$ and $b_t \perp b_i, \forall b_i$. Let $(x'_s, y'_s) = (0, \infty)$ and $(x'_t, y'_t) = (\infty, 0)$. R_h (R_v) consists of b_s (b_t) initially, as does the corresponding T_h (T_v). To pack a module b_j in Γ_-, we search for the last module b_p with $b_p \vdash b_j$ ($b_p \perp b_j$) to compute the x coordinate (y coordinate) of b_j. We traverse the modules b_k's in T_h (T_v) from its root, and go to the right child of b_k if $b_k \vdash b_j$ ($b_k \perp b_j$). The reason to proceed to the right child is that if $b_k \vdash b_j$ ($b_k \perp b_j$), then $x'_j \geq x'_k$ ($y'_j \geq y'_k$) [the right (top) boundary of the module b_j is larger than that of the module b_k] and thus b_j must be located in b_k's right subtree according to the definition of a binary search tree. In contrast, we proceed to the left child if $b_k \perp b_j$ ($b_k \vdash b_j$). The process continues until a leaf position is encountered, and we make b_j the leaf node; therefore, $x_j = x'_p$ ($y_j = y'_p$), where b_p is the last module with $b_p \vdash b_j$ ($b_p \perp b_j$) in the path. (Each module is placed at a bend in the current contour, according to its geometric relationship to the placed modules defined in the constraint graphs.) As an example shown in Fig. 3(a) and (b), we make b_a the right child of b_s in T_h since $b_s \vdash b_a$ and then b_b the left child of b_a in T_h since $b_a \perp b_b$, respectively.

After b_j is inserted into T_h (T_v), every node b_l after b_j (in terms of in-order search tree order) with $x'_l \leq x'_j$ ($y'_l \leq y'_j$) in T_h (T_v) is deleted, since b_l is no longer in the contour. For the example shown in Fig. 3(d), we delete b_a from T_h after inserting b_d, since $x'_a \leq x'_d$ (module b_a is no longer in the horizontal contour T_h). For another example shown in Fig. 3(g), we delete b_d, b_e, b_g (which are after b_f) from T_h after inserting b_f, since $x'_i \leq x'_f$, $i = d, e$, or g (module b_d, b_e, b_g are no longer in the horizontal contour T_h after b_f is inserted). It is not difficult to see that those modules [b_a in Fig. 3(d) and b_d, b_e, b_g in

[2]Note that, by this definition, the $\mathbf{\Gamma_-}$ may not always be the same as the second sequence defined in the corresponding SP. Nevertheless, both of our $\mathbf{\Gamma_-}$ and the second sequence of an SP are good for the efficient packing scheme introduced in this paper.

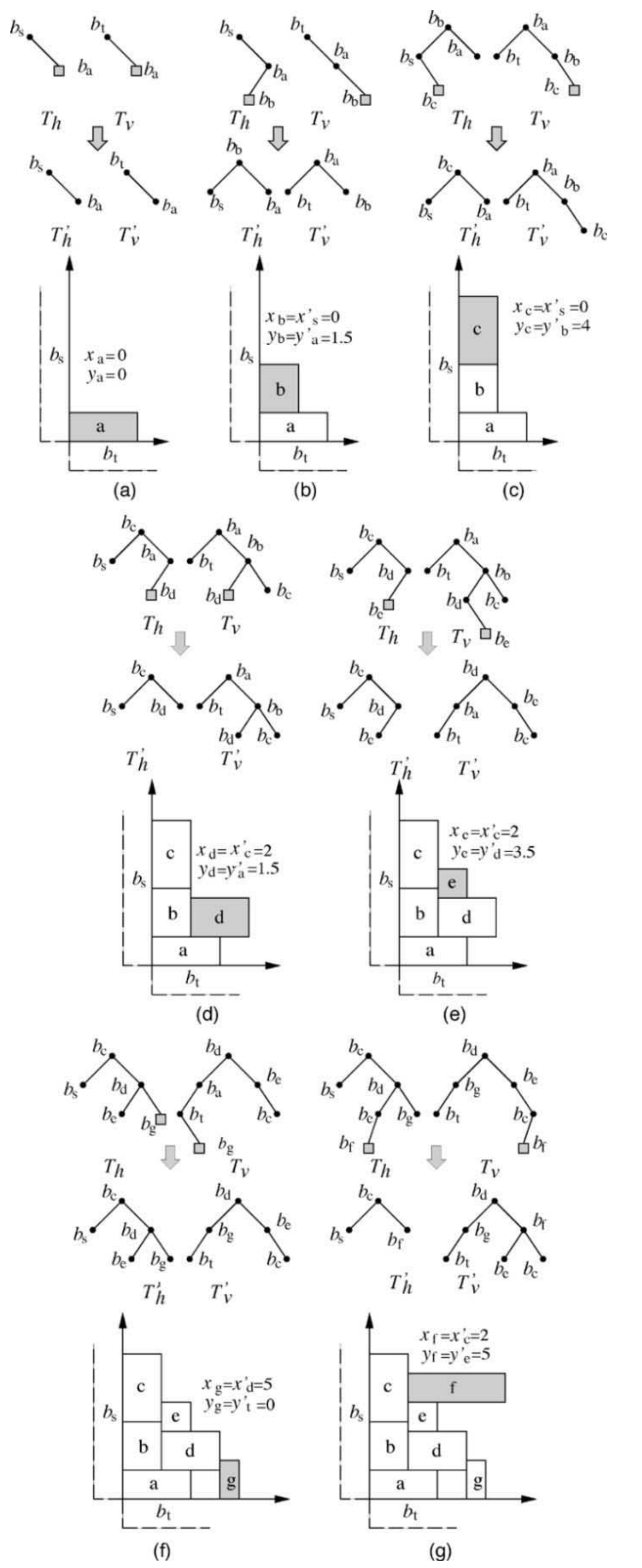


Fig. 3. Packing scheme for the TCG-S of Fig. 2(b). In each step, the red-black trees $\boldsymbol{T}_h$ and $\boldsymbol{T}_v$ corresponding to the $\boldsymbol{R}_h$ and $\boldsymbol{R}_v$ right after the module insertion, are shown. $\boldsymbol{T}_h'$ $(\boldsymbol{T}_v')$ gives the resulting red-black tree after removing the modules no longer in $\boldsymbol{R}_h$ $(\boldsymbol{R}_v)$ and performing rotation to balance the tree. Note that, as a fundamental property of the binary search tree, the search-tree (in-order traversal) order is still maintained after the tree rotation.

Fig. 3(g)] are all the modules that need to be removed—the nodes b_p's before the newly inserted node b_j in T_h all have $x_p' \leq x_j$ since b_j must be placed at a bend formed by two modules according to our scheme, and the remaining nodes b_q's after b_j in T_h all have $x_q' > x_j'$, and thus b_p's and b_q's are still in the horizontal contour T_h. The situation for the vertical contour T_v can be defined analogously. In Fig. 3(d) [Fig. 3(e)], for example, b_c (b_d) remains in T_h after b_d (b_e) is inserted into T_h since $x_c' \leq x_d$ $(x_d' > x_e')$. This process repeats for all modules in Γ_-. We have $W = x_v'$ $(H = y_v')$ if b_v is the module in the resulting T_h (T_v) with the largest value, where H (W) denotes the width (height) of the placement.

Fig. 3 shows the packing scheme for the TCG-S of Fig. 2(g). T_h (T_v) consists of b_s (b_t) initially. To pack the first module b_a in Γ_-, we traverse T_h (T_v) from the root b_s (b_t) and insert it to the right child of b_s (b_t) since $b_s \vdash b_a$ $(b_t \perp b_a)$. As shown in Fig. 3(a), the first module b_a in Γ_- is placed at the bottom-left corner (i.e., $(x_a, y_a) = (0, 0)$) since b_s (b_t) is the last module that is horizontally (vertically) related to b_a and $x_s' = 0$ $(y_t' = 0)$. [Note that T_h' (T_v') in Fig. 3(a) denotes a balanced binary search tree after b_a is inserted into T_h (T_v).] Similarly, to pack the second module b_b in Γ_-, we traverse T_h from the root b_s and then its right child since $b_s \vdash b_b$. Then, b_b is inserted to the left child of b_a since $b_a \perp b_b$. Because b_s is the last module with $b_s \vdash b_b$ in the path, $x_b = x_s' = 0$. Similarly, we traverse T_v from the root b_t and then its right child b_a since $b_t \perp b_a$. Then, b_b is inserted to the right child of b_a in T_v since $b_a \perp b_b$. Therefore, $y_b = y_a' = 1.5$ because b_a is the last module with $b_a \perp b_b$ in the path. The resulting balanced binary search trees after performing tree rotations T_h', T_v' (see [2] for the rotation operations for keeping a tree balanced), and the corresponding placement are shown in Fig. 3(b). Note that, as a fundamental property of the search tree, the tree rotation still maintain the ordering of the nodes in the search tree [2]. [See the resulting T_h' and T_v' in Fig. 3(b).] As shown in Fig. 3(c), after b_c is inserted, b_b in T_h is deleted since b_b is after b_c (in terms of in-order search tree traversal) and $x_b' \leq x_c'$ (i.e., b_b is no longer in the horizontal contour). The resulting T_h', T_v', and placement are shown in Fig. 3(c). The process is repeated for all modules in Γ_-, and the final T_h', T_v', and placement are shown in Fig. 3(g). Then, we have $W = x_f'$ since b_f is the module with the largest x value in the final T_h', and $H = y_c'$ since b_c is the module with the largest y value in the final T_v'.

According to this packing scheme, if the coordinate of a module b_i in Γ_- is changed, we only need to recompute the coordinates of modules after b_i in Γ_-, since the coordinates of modules before b_i do not change.

The above packing scheme leads to the following theorem and lemmas.

Theorem 2: There exists a unique placement corresponding to a TCG-S.

Proof: As shown in Section III-B, TCG-S is equivalent to TCG and SP. Since there exists a unique placement corresponding to a TCG [9] (or an SP [13]), there exists a unique placement corresponding to a TCG-S. (The above packing scheme provides a faster approach to determining the coordinate for each module.) ■

Theorem 3: The size of the solution space for $\mathrm{TCG-S} = (C_h, C_v, \Gamma_-)$ is $(m!)^2$, where m is the number of modules.

Proof: The sequence Γ_- is the topological order of both C_h and C_v and is thus uniquely determined by C_h and C_v. Therefore, the solution space of TCG-S is the same as that of TCG, which is $(m!)^2$ [9], where m is the number of modules. ■

Lemma 1: For each module b_j in Γ_-, b_j must be placed adjacent to the right (top) boundary of some module b_i in R_h (R_v) during the packing.

Proof: For a placed module b_k, if b_k is not in R_h, there must exist a placed module b_l above b_k whose right boundary is larger than that of b_k (i.e., $x_l' > x_k'$ and $y_l \geq y_k'$). Since b_l is placed before b_j, Γ_- is

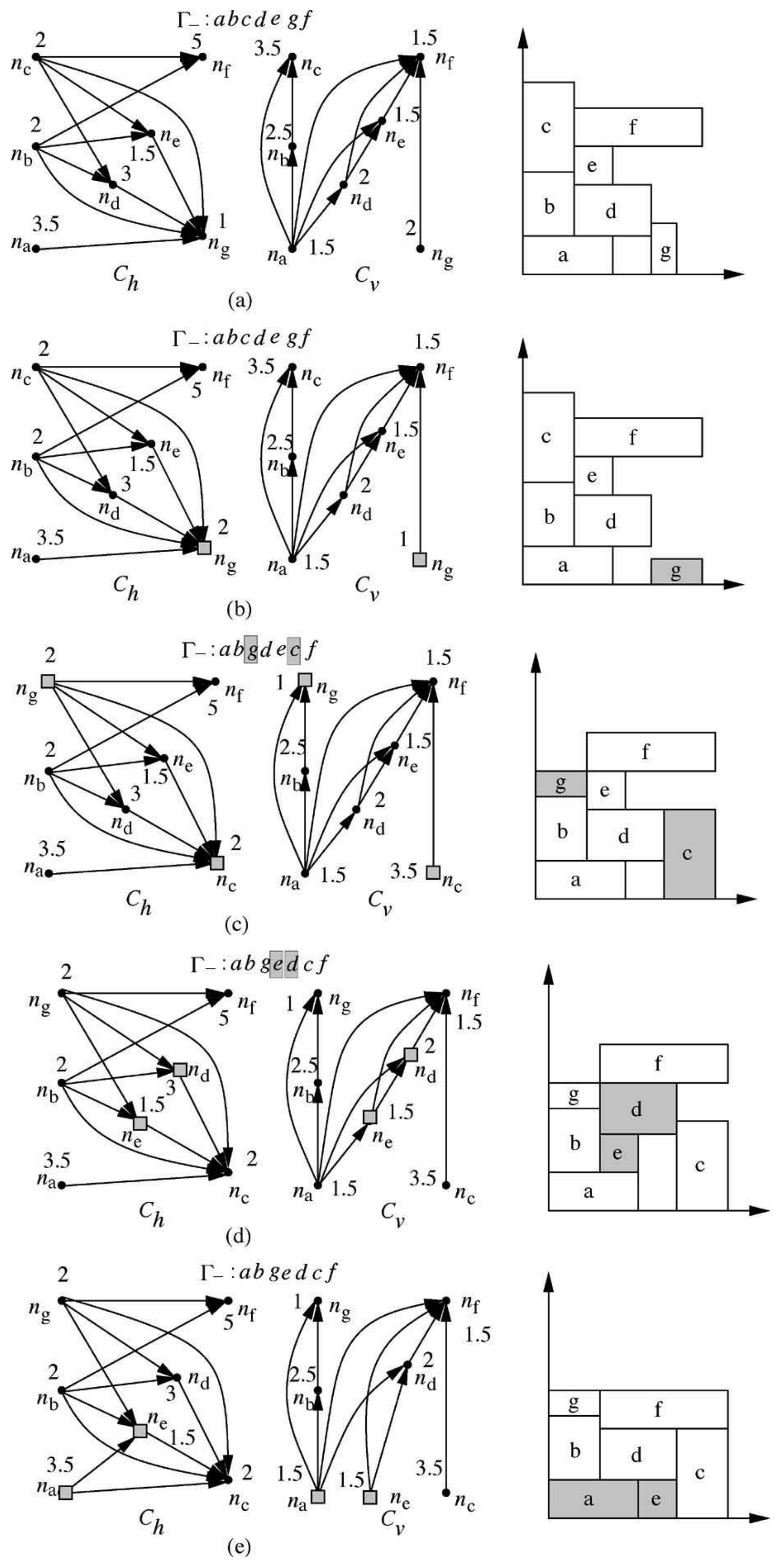


Fig. 4. Four types of perturbation. (a) The initial TCG-S and placement. (b) The resulting TCG-S and placement after rotating the module $\boldsymbol{b}_g$ shown in (a). (c) The resulting TCG-S and placement after swapping the nodes $\boldsymbol{n}_c$ and $\boldsymbol{n}_g$ shown in (b). (d) The resulting TCG-S and placement after reversing the reduction edge $(\boldsymbol{n}_d, \boldsymbol{n}_e)$ shown in (c). (e) The resulting TCG-S and placement after moving the reduction edge $(\boldsymbol{n}_a, \boldsymbol{n}_e)$ from the $\boldsymbol{C}_v$ of (d) to $\boldsymbol{C}_h$.

of the sequence $\langle \ldots l \ldots i \ldots \rangle$. If b_j is adjacent to the right boundary of b_k, b_j should also be below b_l. This implies that there exists an edge (b_j, b_l) in C_v, and thus Γ_- is of the sequence $\langle \ldots j \ldots l \ldots \rangle$, instead of $\langle \ldots l \ldots j \ldots \rangle$ (a contradiction). Therefore, b_j must be adjacent to some module b_i in current R_h. Similarly, b_j must be placed above some module in R_v during packing. ■

Lemma 2: Given a module b_j in Γ_- to be placed, if $b_i \in R_h(R_v)$ and $b_i \perp b_j$ ($b_i \vdash b_j$), any module $b_k \in R_h(R_v)$ with $x'_k > x'_i$ ($y'_k > y'_i$) cannot bear the relation $b_k \vdash b_j$ ($b_k \perp b_j$).

Proof: Suppose there exists a module b_k in R_h so that $x'_k > x'_i$ and $b_k \vdash b_j$, when we determine the x coordinate of a module b_j. The relation between b_i and b_k can be either $b_i \vdash b_k$ (since $x'_i < x'_k$) or $b_k \perp b_i$ (since b_i would have been removed from R_h if $b_i \perp b_k$). If $b_i \vdash b_k$, then $b_i \vdash b_j$ since $b_k \vdash b_j$ (based on Property 3 of TCG-S), contradicting Property 2 of TCG-S because there are two geometric relations $b_i \vdash b_j$ and $b_i \perp b_j$ between b_i and b_j. Similarly, if $b_k \perp b_i$, then $b_k \perp b_j$ since $b_i \perp b_j$, contradicting Property 2 of TCG-S because $b_k \vdash b_j$ and $b_k \perp b_j$. Similar claims hold for R_v. ■

By this lemma, during the packing for the module b_j, if $b_i \in R_h$ and $b_i \perp b_j$, then any module $b_k \in R_h$ after b_i (in the search-tree order) cannot have the relation $b_k \vdash b_j$. It is clear later that this lemma is useful for searching for the last module b_p with $b_p \vdash b_j$ for computing the coordinate of b_j—if $b_i \in R_h$ and $b_i \perp b_j$, every module $b_k \in R_h$ after b_i can be discarded during the search.

For each module b_j, its x coordinate x_j equals the maximum x'_k (the largest right boundary) among those modules b_k's with $b_k \vdash b_j$ in TCG-S. Based on Lemma 1, b_k must be in R_h. Therefore, we only need to traverse the modules in R_h to find such a module. Since modules in R_h are stored in a balanced binary search tree T_h, the traversal from its root to a leaf takes only $O(\lg m)$ time, where m is the number of nodes (modules). For each module b_p encountered, we go to its right child if $b_p \vdash b_j$ since b_p's right child corresponds to some module b_u with $x'_u \geq x'_p$ based on the definition of our search-tree order. Note that our goal is to find the module b_k with the largest right boundary such that $b_k \vdash b_j$; if such a module is found, then $x_j = x'_k$ and we have obtained the x coordinate of the module x_j. However, if $b_p \perp b_j$, we traverse the left child of b_p. By Lemma 2, the modules b_q's after b_p (in the search-tree order) cannot bear the relation $b_q \vdash b_j$ if $b_p \perp b_j$; therefore, we do not need to consider the right subtree of b_p for this case. As the example shown in Fig. 3(e) for the computation of the x coordinate of the module e, we traverse the right child of the root b_c since $b_c \vdash b_e$, and then make b_e the left child of b_d since $b_d \perp b_e$. Therefore, we are done with the traversal, and $x_e = x'_c$ since b_c has the maximum right boundary among the modules horizontally related to b_e. For the example shown in Fig. 3(f) for the computation of the x coordinate of the module g, we traverse the right child b_d of the root b_c since $b_c \vdash b_g$, and then make b_g the right child of b_d since $b_d \vdash b_g$. We therefore make $x_g = x'_d$ since b_d is the module with the largest right boundary such that $b_d \vdash b_g$. Note that we do not need to consider b_e in this case, since it is before b_d in the search tree, implying that b_e's right boundary is not larger than that of b_d. By repeating this procedure, x_j equals x'_r if b_r is the last module in the search path with $b_r \vdash b_j$. A similar process can be applied to compute the y coordinate of module b_i. We have the following theorem.

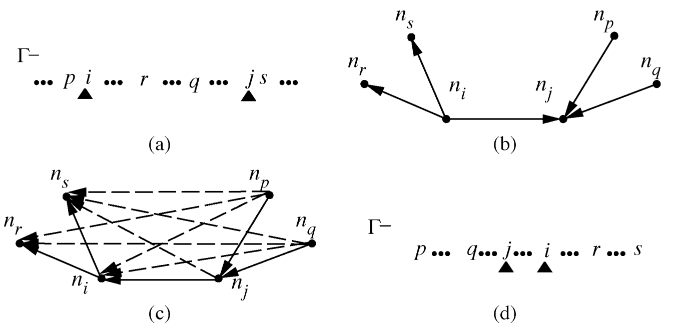
Theorem 4: The proposed scheme correctly packs all modules in $O(m \lg m)$ time, where m is the number of modules.

Proof: For each module b_j, its x coordinate x_j equals the maximum x'_k among those modules b_k's with $b_k \vdash b_j$ in TCG-S. Based on Lemma 1, b_k must be in R_h. Therefore, we only need to traverse the modules in R_h to find such a module. Since the modules in R_h are stored in a balanced tree T_h with the search-tree order defined on the right boundaries of the modules, the traversal can be done from its root to a leaf. For each module b_p encountered, we go to its right child if $b_p \vdash b_j$ since b_p's right child corresponds to some module b_u with $x'_u \geq x'_p$ based on the definition of our search-tree order. Note that our goal is to find the module b_k with the largest right boundary such that $b_k \vdash b_j$; if such a module is found, then $x_j = x'_k$ and we have obtained the x coordinate of the module x_j. However, if $b_p \perp b_j$, we traverse the left child of b_p. By Lemma 2, the modules b_q's after b_p (in the search-tree order) cannot bear the relation $b_q \vdash b_j$ if $b_p \perp b_j$; therefore, we do not need to consider the right subtree of b_p. By repeating this procedure, x_j equals x'_r if b_r is the last module in the search path with $b_r \vdash b_j$. Similar claim holds for the process to get the y coordinate of module b_i in R_v.

For any module $b_p \in R_h$, either $b_p \vdash b_j$ or $b_p \perp b_j$, since we place b_j at the bend formed by two modules in the horizontal contour. Therefore, we only need to search for a path from the root to a leaf to pack a module. For a balanced binary tree with n nodes, the path length is given by the height of the tree, which is $O(\lg n)$. The time complexity of inserting and deleting a node in the tree is $O(\lg n)$. Therefore, the time complexity of our packing scheme is given by

$$O\left(\sum_{i=1}^{m} \lg i + \sum_{i=1}^{m} \lg i\right) = O(m \lg m)$$

where the first and the second terms correspond to the time complexity of inserting and deleting nodes into the tree, respectively. ■



Γ−
... p i ... r ... q ... j s ...
(a)
n_s
n_p
n_r
n_q
n_i
n_j
(b)
n_s
n_p
n_r
n_q
n_i
n_j
(c)
Γ−
p ... q ... j ... i ... r ... s
(d)

Fig. 5. Example of updating $\boldsymbol{\Gamma_-}$ by reversing an edge $(\boldsymbol{n_i, n_j})$. (a) $\boldsymbol{\Gamma_-}$ before the operation. (b) A constraint graph before the operation. (c) The new constraint graph after the operation. (d) The new $\boldsymbol{\Gamma_-}$ after the operation.

V. Floorplanning Algorithm

We develop a simulated annealing-based algorithm [6] by using TCG-S for nonslicing floorplan design. Given an initial TCG-S, the algorithm perturbs the TCG-S into a new TCG-S to find a desired solution. To ensure the correctness of the new C_h and C_v, they must satisfy the three feasibility conditions given in Section III-A. To identify feasible C_h and C_v for perturbation, we describe the concept of *reduction edges* in the following subsection.

A. Reduction Edge

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*. In Fig. 2(g), for example, edges (n_a, n_g), (n_d, n_g), and (n_e, n_g) are reduction edges while (n_b, n_g) and (n_c, n_g) are closure ones. With Γ_-, we can find a *set* of reduction edges in $O(m)$ time (where m is the number of modules), a significant improvement from $O(m^2)$ time using TCG alone [9].

Given an arbitrary node n_i in a transitive closure graph C_h (C_v), we can find all the nodes n_j's that form reduction edges (n_i, n_j)'s using a *Linear Scan* method as follows. First, we extract from Γ_- those nodes n_j's in $F_{\text{out}}(n_i)$ of C_h (C_v) and keep their original ordering in Γ_-. Let the resulting sequence be S. The first node n_k in S and n_i must form a reduction edge (n_i, n_k). Then, we continue to traverse S until a node n_l with (n_k, n_l) *not* in C_h (C_v) is encountered. (n_i, n_l) must also be

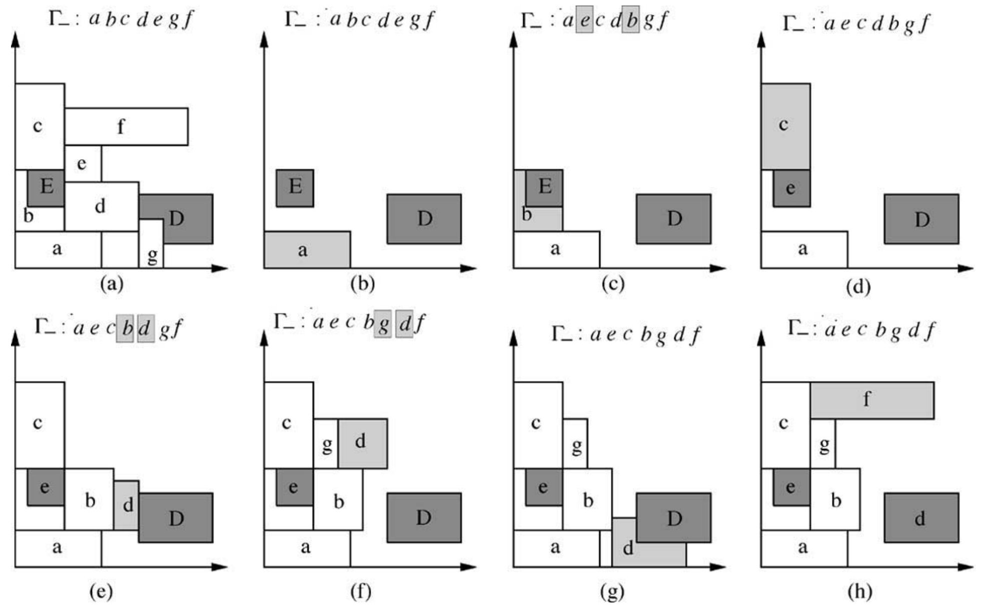


Fig. 6. Examples for preplaced modules. (a) The resulting placement of TCG-S shown in Fig. 2(g). Here, the heavily shaded rectangles $\boldsymbol{D}$ and $\boldsymbol{E}$ denote the preplaced locations for $\boldsymbol{b}_d$ and $\boldsymbol{b}_e$. (b)–(h) The process to transform an infeasible TCG-S into a feasible one.

a reduction edge. Starting from n_l, we continue the same process until no node is left in S.

As an example shown in C_h of Fig. 2(g), we are to extract all reduction edges emanating from n_c. We first find $S = \langle n_d, n_e, n_g, n_f \rangle$ by extracting nodes in $F_{\text{out}}(n_c)$ based on the sequence in Γ_-. n_c and the first node n_d in S form a reduction edge (n_c, n_d). Traversing S, we have another reduction edge (n_c, n_e) since edge (n_d, n_e) is not in C_h. Starting from n_e, we search the next node n with (n_e, n) not in C_h. We find node n_f, implying (n_c, n_f) is also a reduction edge. Therefore, we have found all reduction edges emanating from n_c: (n_c, n_d), (n_c, n_e), and (n_c, n_f). (Note that (n_c, n_g) is not a reduction edge, because we found (n_e, n_g) in C_h during the processing.)

Theorem 5: Given a node n_i in C_h (C_v), the Linear Scan method finds all reduction edges emanating from n_i in $O(m)$ time, where m is the number of modules.

Proof: Given an arbitrary node n_i in a transitive closure graph C_h (C_v), we first extract those nodes n_j's in $F_{\text{out}}(n_i)$ by scanning modules in Γ_-. Let the resulting sequence be S. n_i and the first node n_k in S must form a reduction edge (n_i, n_k). If (n_i, n_k) is a closure edge, there must exist a node n_p in the path $\langle n_i, \ldots, n_p, \ldots, n_k \rangle$ from n_i to n_k in C_h (C_v), implying that n_p must be extracted before n_k in the scanning (contradicting the fact that n_k is the first node encountered in Γ_-). Therefore, (n_i, n_k) is a reduction edge. Then, we continue traversing nodes in S until another node n_l associated with edge (n_k, n_l) not in C_h (C_v) is met. For those nodes, n_q's between n_k and n_l, (n_i, n_q) must be a closure edge since there exists a path $\langle n_i, n_k, \ldots, n_q \rangle$ from n_i to n_q in C_h (C_v). However, nodes n_i and n_l form a reduction edge. If there exists a path from n_i to n_l, the path must pass through at least one node n_q between n_k and n_l in S. However, n_q is connected by n_k, implying the existence of the path $\langle n_k, n_q, n_l \rangle$. According to Property 3 of TCG, the edge (n_k, n_l) also exists (a contradiction). Therefore, n_i and n_l form a reduction edge (n_i, n_l). The same process continues for node n_l.

For a successor node n_r of n_l, if edge (n_l, n_r) exists in C_h (C_v), edge (n_i, n_r) must be the closure edge according to the above reasoning. If the edge (n_l, n_r) does not exist in C_h (C_v), we show that n_r cannot connect to a node n_v, where n_v is the node before n_l in S. Suppose that the edge (n_l, n_r) does not exist in C_h (C_v), but n_r is connected to node n_v, where n_v is the node before n_l in S and the edge (n_i, n_v) is a reduction edge. [If (n_i, n_v) is a closure edge, we can always find another node n_u that is a fanin of n_v and (n_i, n_u) is a reduction edge.] If (n_i, n_v) and (n_i, n_l) are reduction edges in C_h (C_v), there must exist an edge (n_v, n_l) in C_v (C_h) since there exists a unique edge between each pair of nodes in TCG. Similarly, because the edge (n_l, n_r) does not exist in C_h (C_v), the edge (n_l, n_r) must exist in C_v (C_h). The path $\langle n_v, n_l, n_r \rangle$ exists in C_v (C_h), implying that the edge (n_v, n_r) also exists in C_v (C_h). Therefore, the edge (n_v, n_r) exists in both C_h and C_v, contradicting Property 2 of TCG.

It takes $O(m)$ time to scan modules in Γ_- to find nodes n_j's that belong to $F_{\text{out}}(n_i)$. Therefore, the Linear Scan method finds all reduction edges emanating from n_i in $O(m)$ time, where m is the number of modules. ■

TABLE I
FIVE MCNC BENCHMARK CIRCUITS

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

B. Solution Perturbation

We extend the four operations *rotation*, *swap*, *reverse*, and *move* presented in [9] to perturb C_h and C_v. During each perturbation, we must maintain the three feasibility properties for C_h and C_v. Unlike the rotation operation, swap, reverse, and move may change the configurations of C_h and C_v and thus their properties. Further, we also need to maintain Γ_- to conform to the topological ordering for new C_h and C_v.

1) Rotation: To rotate a module b_i, we exchange the weights of the corresponding node n_i in C_h and C_v. Since the configurations of C_h and C_v do not change, so does Γ_-. Fig. 4(b) shows the resulting TCG-S and placement after rotating the module g shown in Fig. 4(a). Notice that the new Γ_- is the same as that in Fig. 4(a).

Theorem 6: TCG-S is closed under the rotation operation, and such an operation does not change the topology of the TCG and Γ_-.

TABLE II
AREA AND RUNTIME COMPARISONS AMONG SP (ON SUN SPARC ULTRA60), O-TREE (ON SUN SPARC ULTRA 60), $\mathbf{B}^*$-TREE (ON SUN SPARC ULTRA-I), ENHANCED O-TREE (ON SUN SPARC ULTRA60), CBL (ON SUN SPARC 20), TCG (ON SUN SPARC ULTRA60) AND TCG-S (ON SUN SPARC ULTRA60) FOR AREA OPTIMIZATION

Circuit	SP		O-tree		B*-tree		Enhanced O-tree		CBL		TCG		TCG-S	
	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)	Area (mm^2)	Time (sec)	Area (mm^2)	Time (sec)
apte	48.12	13	47.1	38	46.92	7	46.92	11	NA	NA	46.92	1	46.92	1
xerox	20.69	15	20.1	118	19.83	25	20.21	38	20.96	30	19.83	18	19.796	5
hp	9.93	5	9.21	57	8.947	55	9.16	19	(66.14)	(32)	8.947	20	8.947	7
ami33	1.22	676	1.25	1430	1.27	3417	1.24	118	1.20	36	1.20	306	1.185	84
ami49	38.84	1580	37.6	7428	36.80	4752	37.73	406	38.58	65	36.77	434	36.398	369

Proof: The configuration of TCG is not changed after a rotation operation, neither is Γ_-. ■

2) Swap: Swapping n_i and n_j does not change the topologies of C_h and C_v. Therefore, we only need to exchange b_i and b_j in Γ_-. Fig. 4(c) shows the resulting TCG-S and placement after swapping the nodes n_c and n_g shown in Fig. 4(b). Notice that the modules b_c and b_g in Γ_- in Fig. 4(c) are exchanged.

Theorem 7: TCG-S is closed under the Swap operation, and such an operation takes $O(1)$ time.

Proof: The Swap operation only exchanges two nodes n_i and n_j in both C_h and C_v without changing their topologies. Therefore, we only need to exchange two nodes in Γ_-, which takes $O(1)$ time. ■

3) Reverse: To reverse a reduction edge (n_i, n_j) in one transitive closure graph, we first delete the edge (n_i, n_j) from the graph, and then add the edge (n_j, n_i) to the graph. To keep C_h and C_v feasible, for each node $n_k \in F_{in}(n_j) \cup \{n_j\}$ and $n_l \in F_{\text{out}}(n_i) \cup \{n_i\}$ in the new graph, we have to keep the edge (n_k, n_l) in the new graph. If the edge does not exist in the graph, we add the edge to the graph and delete the corresponding edge (n_k, n_l) [or (n_l, n_k)] in the other graph. To make Γ_- conform to the topological ordering of new C_h and C_v, we delete b_i from Γ_- and insert b_i after b_j. For each module b_k between b_i and b_j in Γ_-, we shall check whether the edge (n_i, n_k) exists in the same graph. We do nothing if the edge (n_i, n_k) does not exist in the same graph; otherwise, we delete b_k from Γ_- and insert it after the most recently inserted module.

Fig. 4(d) shows the resulting TCG-S and placement after reversing the *reduction edge* (n_d, n_e) of the C_v in Fig. 4(c). Since there exists no module between b_d and b_e in Γ_-, we only need to delete b_d from Γ_- and insert it after b_e, and the resulting Γ_- is shown in Fig. 4(d).

Theorem 8: TCG-S is closed under the Reverse operation, and such an operation takes $O(m)$ time, where m is the number of modules.

Proof: If an edge (n_i, n_j) in C_h (C_v) is reversed, the Reverse operation will add some edges (n_k, n_l)'s to C_h (C_v) and delete the corresponding edges in C_v (C_h), where $n_k \in F_{in}(n_j) \cup \{n_j\}$ and $n_l \in F_{\text{out}}(n_i) \cup \{n_i\}$. Γ_- is changed because the topological ordering of nodes is changed by those newly added edges. To show the effects of the operation, without loss of generality, we consider Γ_- of the form shown in Fig. 5(a), where: 1) p is a module before module i and n_p is a fanin of n_j (since Reverse only adds edges that start from n_k's, where $n_k \in F_{in}(n_j) \cup \{n_j\}$, we do not need to consider the case where n_p is a fanin of n_i); 2) q and r are modules between modules i and j, and n_q is a fanin of n_j, while n_r is a fanout of n_i [note that if n_q or n_r is a fanout of n_i or a fanin of n_j, edge (n_i, n_j) must be a closure edge (a contradiction)]; and 3) s is a module after module j, and n_s is a fanout of n_i (since Reverse only adds edges that end in n_l's, where $n_l \in F_{\text{out}}(n_i) \cup \{n_i\}$, we do not need to consider the case where n_s is a fanout of n_j). See Fig. 5(b) for the corresponding C_h (C_v). The resulting C_h (C_v) after the operation is shown in Fig. 5(c), in which the dotted lines denote newly added edges. For edges (n_p, n_r), (n_p, n_i), and (n_p, n_s) [(n_j, n_r), (n_j, n_i), and (n_j, n_s)] starting from n_p (n_j), the topological ordering of nodes in the edges is not changed [i.e., n_p (n_j) is in front of n_r, n_i, or n_s]. However, for some edges (n_q, n_r) and (n_q, n_i) starting from n_q, the topological ordering of nodes in the edges is reversed (i.e., n_q is in front of n_i (n_r) instead of behind them). Therefore, we only need to move module i and those modules between modules i and j that belong to the fanout of n_i after module j in Γ_-. Fig. 5(d) shows the new Γ_- after the operation. The number of modules between modules i and j is at most $m-2$; therefore, it takes $O(m)$ time to move these modules after b_j. ■

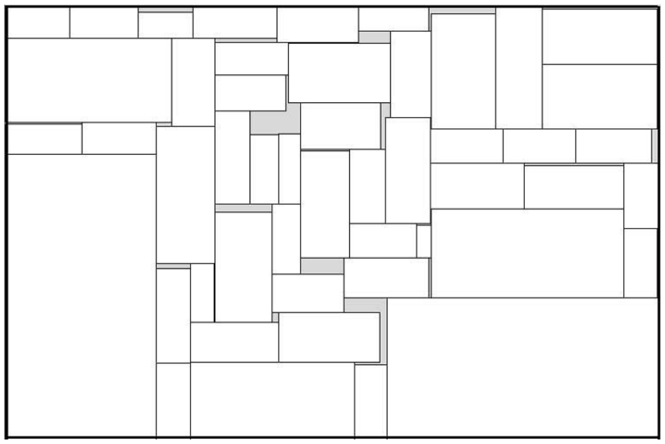


Fig. 7. Resulting placement of ami49 with area optimization (area $=$ $\mathbf{36.398}$ mm^2).

4) Move: To move a reduction edge (n_i, n_j) from a transitive closure graph G to the other G', we delete the edge from G and then add it to G'. Similar to Reverse, for each node $n_k \in F_{in}(n_i) \cup \{n_i\}$ and $n_l \in F_{\text{out}}(n_j) \cup \{n_j\}$ in G', we must move the edge (n_k, n_l) to G' if the corresponding edge (n_k, n_l) (or (n_l, n_k)) is in G. Since the operation changes only the edges in C_h or C_v but not the topological ordering among nodes, Γ_- remains unchanged.

Fig. 4(e) shows the resulting TCG-S and placement after moving the *reduction edge* (n_a, n_e) from C_v to C_h in Fig. 4(d). Notice that the resulting Γ_- is the same as that in Fig. 4(d). For the above operations, we have the following theorem.

Theorem 9: TCG-S is closed under the move operation, and such an operation takes $O(m)$ time, where m is the number of modules. In particular, Γ_- remains the same after the operation.

As shown in [5], in particular, the aforementioned operations form a complete set of operations for perturbing a TCG, i.e., every feasible TCG is reachable from any other feasible TCG. Since Γ_- is induced from TCG, the set of perturbation operations is also sufficient for TCG-S.

VI. PLACEMENT WITH CONSTRAINTS

In this section, we demonstrate the flexibility of TCG-S by extending it to handle placement with boundary and preplaced modules.

TABLE III
WIRELENGTH AND RUNTIME COMPARISONS AMONG O-TREE, ENHANCED O-TREE, TCG, AND TCG-S FOR WIRELENGTH OPTIMIZATION. ALL RAN ON SUN SPARC ULTRA60

Circuit	O-tree		enhanced O-tree		TCG		TCG-S	
	Wire (mm)	Time (sec)	Wire (mm)	Time (sec)	Wire (mm)	Time (sec)	Wire (mm)	Time (sec)
apte	317	47	317	15	363	2	363	2
xerox	368	160	372	39	366	15	366	6
hp	153	90	150	19	143	10	143	4
ami33	52	2251	52	177	44	52	43	89
ami49	636	14112	629	688	604	767	579	570

A. TCG-S With Boundary Modules

The placement with boundary constraints is to place a set of prespecified modules along the designated boundaries of a chip, which can be formulated as follows:

Definition 1: Boundary Constraint: Given a boundary module b, it must be placed in one of the four sides: on the left, on the right, at the bottom or at the top in a chip in the final packing.

TCG-S keeps the following properties that make placement with boundary constraints much easier than other representations.

Theorem 10: If a module b_i is placed along the left (right) boundary, the in-degree (out-degree) of the node n_i in C_h equals zero. If a module b_i is placed along the bottom (top) boundary, the in-degree (out-degree) of node n_i in C_v equals zero.

Proof: If the in-degree of a node n_i in C_h is zero, there exists no module on the left side of b_i in the corresponding placement. Therefore, b_i is always placed along the left boundary after packing. If the out-degree of a node n_i in C_h is zero, there exists no module on the right side of b_i in the corresponding placement. Therefore, we can push the module to the right boundary by assigning its x coordinate as $W - W_i$, where W is the width of the placement. Similar claims hold for the bottom and top boundary modules. ■

For each perturbation, we can guarantee a feasible placement by checking whether the conditions of boundary modules are satisfied. We discuss the modifications for the four perturbation operations as follows.

1) Rotation: Since Rotation does not change module location, the operation remains the same as before.

2) Swap: We can swap two nodes n_a and n_b if

1) b_a and b_b are not boundary modules,
2) b_a and b_b are boundary modules of the same type, or
3) b_a is a boundary module and b_b is not, and n_b satisfies the boundary constraint of b_a.

3) Reverse: If b_a is a left boundary module or b_b is a right boundary module, then the reduction edge (n_a, n_b) in C_h cannot be reversed. Similarly, we cannot reverse the reduction edge (n_a, n_b) in C_v if b_a is a bottom boundary module or b_b is a top boundary module.

4) Move: If b_a is a top boundary module or b_b is a bottom boundary module, we cannot move the reduction edge (n_a, n_b) from C_h to C_v. Similarly, we cannot move the reduction edge (n_a, n_b) from C_v to C_h if b_a is a right boundary module or b_b is a left boundary module.

We have the following theorem.

Theorem 11: TCG-S is closed under the rotation, swap, reverse, and move operations with boundary constraints.

Proof: To show TCG-S is closed under the rotation, swap, reverse, and move operations with boundary constraints, we only need to prove that the feasibility conditions of boundary modules defined in Theorem 10 are not violated under these operations.

- Rotation: Since the rotation operation does not change the configuration of TCG-S, the feasibility conditions of boundary modules are not violated.

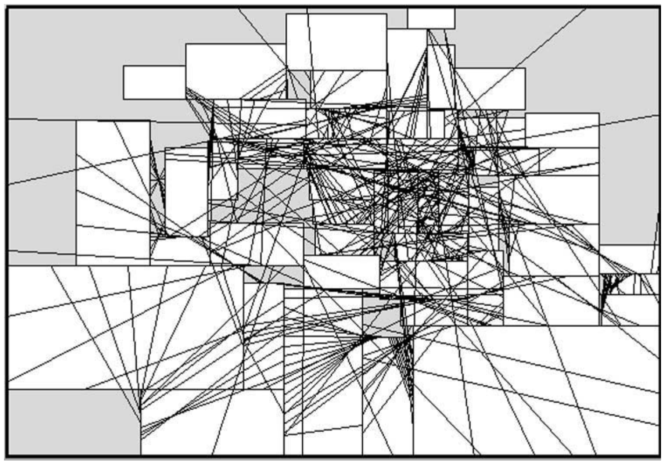


Fig. 8. Resulting placement of ami49 with wirelength optimization $(\texttt{wire} = \texttt{579}$ mm$)$.

- Swap: If a boundary module b_i and another module b_j are exchanged in the Swap operation, the feasible condition of the boundary module b_i is checked before the operation. Therefore, Theorem 10 is not violated under the swap operation.
- Reverse: If an edge (n_i, n_j) in C_h (C_v) is reversed, the reverse operation will add some edges (n_k, n_l)'s to C_h (C_v) and delete the corresponding edges in C_v (C_h), where $n_k \in F_{in}(n_j) \cup \{n_j\}$ and $n_l \in F_{\text{out}}(n_i) \cup \{n_i\}$. We first consider the feasibility condition of the top (bottom, left, or right) module if a reduction edge in C_h is reversed as follows: Since the reverse operation does not add any edge to C_v if an edge (n_i, n_j) in C_h is reversed, it will not generate any new edge associated with the fanin (fanout) of a bottom (top) module in C_v. Therefore, the feasibility condition of the bottom (top) module is not violated. For a left boundary module b_q, suppose there exists an edge (n_p, n_q) in C_h after the operation. n_q is either n_i or a fanout of n_i. However, we do not reverse the edge (n_i, n_j) in C_h if n_i is a left boundary module. Besides, n_q cannot be a fanout of others in the original graph C_h if b_q is a left boundary. Similarly, for a right boundary module b_p, suppose there exists an edge (n_p, n_q) in C_h after the operation. n_p is either n_j or a fanin of n_j. However, we do not reverse an edge (n_i, n_j) in C_h if n_j is a right boundary module. Besides, n_p cannot be fanin of others in the original graph C_h if b_p is a right boundary module. Similar claims hold when an edge (n_i, n_j) in C_v is reversed.
- Move: If an edge (n_i, n_j) in C_h (C_v) is moved to C_v (C_h), the Move operation will add some edges (n_k, n_l)'s to C_v (C_h) and delete the corresponding edges in C_h (C_v), where $n_k \in F_{in}(n_i) \cup \{n_i\}$ and $n_l \in F_{\text{out}}(n_j) \cup \{n_j\}$. We first consider the feasibility condition of the top (bottom, left, or right) module if a reduction edge in C_h is moved to C_v as follows: Since the Move

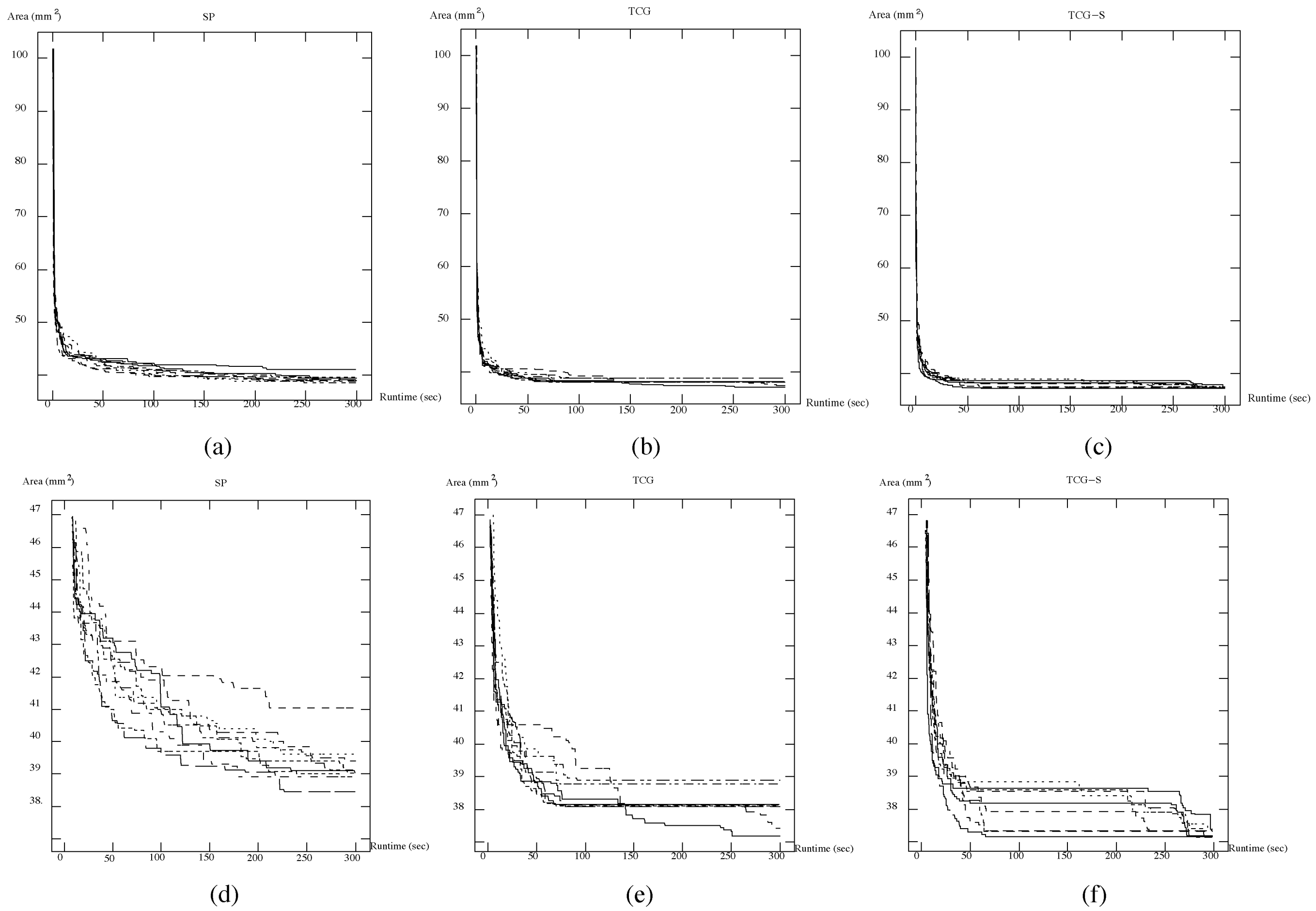


Fig. 9. Comparison for stability and convergence between SP, TCG, and TCG-S for ami49.

operation does not add any edge to C_v after an edge (n_i, n_j) in C_v is moved to C_h, it will not generate any new edge in the fanin (fanout) of a bottom (top) module in C_v. Therefore, the feasibility condition of the top (bottom) module is not violated. For a left boundary module b_q, suppose there exists an edge (n_p, n_q) in C_h after the operation. n_q is either n_j or a fanout of n_j. However, we do not move the edge (n_i, n_j) from C_v to C_h if n_j is a left boundary module. Besides, n_q cannot be a fanout of others in the original graph C_h if b_q is a left boundary module. Similarly, for the right boundary module b_p, suppose there exists an edge (n_p, n_q) in C_h after the edge (n_i, n_j) is moved to C_h. n_p is either n_i or a fanin of n_i. However, we do not move an edge (n_i, n_j) from C_v to C_h if n_i is a right boundary module. Besides, n_p cannot be a fanin of others in the original graph C_h if b_q is a right boundary module. Similar claims hold when an edge (n_i, n_j) is moved from C_v to C_h. ■

B. TCG-S With Preplaced Modules

The placement with preplaced modules is to place a set of prespecified modules at the designated locations of a chip, which can be formulated as follows:

Definition 2: Preplaced Constraint: Given a module b_i with a fixed coordinate (x_i, y_i) and an orientation, b_i must be placed at the designated location with the same orientation in the final packing.

Whether a preplaced module is packed at a correct location is not known until packing. Also, changing the coordinate of a module b_i may affect the packing for other modules after b_i in Γ_-. Therefore, we may need to modify a TCG-S to guarantee a feasible placement with the preplaced constraint after each perturbation.

Given a TCG-S, modules are packed one by one based on the sequence of Γ_-. A module b_i *interacts with* another module b_j if: 1) b_i overlaps b_j; 2) $b_j \vdash b_i$ and their projections on the y axis overlap; or 3) $b_j \perp b_i$ and their projections on the x axis overlap. If b_i interacts with a preplaced module b_j and b_i was placed, n_i and n_j are swapped in the TCG-S to make b_j placed at the designated location. If a preplaced module b_i was placed and the resulting placement of b_i does not interact with itself at the designated location, we swap b_i with the node b_j right after b_i in Γ_-; otherwise, b_i is placed at the designated location if there exists no module behind b_i in Γ_-.

Fig. 6(a) shows the resulting placement of the TCG-S of Fig. 2(g) without considering preplaced modules. Here, the heavily shaded rectangles, labeled D and E, give the correct locations for the preplaced modules b_d and b_e, respectively. As shown in the figure, the correct location for b_e (b_d) is occupied by b_b (b_g). Fig. 6(b)–(h) illustrate the process to transform an infeasible TCG-S into a feasible one with preplaced constraints. Fig. 6(b) and (c) show the placements after b_a and b_b are placed, respectively. Since b_b overlaps the region for the preplaced module b_e [see Fig. 6(c)], we swap b_b and b_e in TCG-S, and then place b_e at the designated location. (Note that b_b and b_e are exchanged in Γ_-.) Fig. 6(d) and (e) show the respective placements after b_c and the preplaced module b_d are placed. Since the preplaced module b_d does not intersect with itself at the designated coordinate [see Fig. 6(e)], we swap b_d and its successor b_b in Γ_-. (Note that b_d and b_b in Γ_- are exchanged.) Similarly, we swap b_d and b_g in TCG-S, resulting in the

TABLE IV
AREA AND RUNTIME COMPARISONS BETWEEN [7] (ON PENTIUM-II 350) AND TCG-S (ON SUN SPARC ULTRA60) FOR PLACEMENT WITH BOUNDARY MODULES

Circuit	$\#\|T\|, \#\|B\|, \#\|L\|, \#\|R\|$	Lai et al. [7]		TCG-S	
		Area (mm^2)	Time (sec)	Area (mm^2)	Time (sec)
apte	1, 1, 1, 1	46.92	15	46.92	3
xerox	1, 1, 1, 1	20.4	19	19.977	33
hp	1, 1, 1, 1	9.24	23	9.158	26
ami33	2, 2, 2, 2	1.21	290	1.190	238
ami49	3, 3, 2, 3	36.84	601	36.765	584

placement shown in Fig. 6(f). Then, b_d is placed at the designated location in Fig. 6(g). Finally, the resulting placement after packing b_f is shown in Fig. 6(h).

VII. EXPERIMENTAL RESULTS

Based on a simulated annealing method [6], we implemented the TCG-S representation in the C++ programming language on a 433-MHz SUN Sparc Ultra-60 workstation with 1 GB of memory. The code is available at http://cc.ee.ntu.edu.tw/~ywchang/research.html. Based on the five commonly used MCNC benchmark circuits listed in Table I, we conducted four experiments: 1) area optimization; 2) wirelength optimization; 3) solution convergence speed and stability; and 4) placement with boundary constraints. In Table I, Columns 2–5 list the respective numbers of modules, I/O pads, nets, and pins of the five circuits.

For Experiment 1, the area and runtime comparisons among SP, O-tree, B*-tree, enhanced O-tree, CBL, and TCG are listed in Table II. As shown in Table II, TCG-S achieves best area utilization for the benchmark circuits in very efficient running times. Fig. 7 shows the resulting placement for ami49 with area optimization.

For Experiment 2, 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. As shown in the Table III, TCG-S achieves the best average results in wirelength than O-tree, enhanced O-tree, and TCG using smaller running times. (Note that we did not compare with B*-tree and CBL here since they did not report the results on optimizing wirelength alone.) Fig. 8 shows the resulting placement for ami49 with wirelength optimization.

In addition to the area and timing optimization, in Experiment 3, we also compared the solution convergence speed and stability among SP, TCG, and TCG-S to eliminate the possible unfairness due to the nondeterministic behavior of simulated annealing, which were neglected in most previous works. (Note that other tools are not available to us for the comparative study.) We randomly ran SP, TCG, and TCG-S on ami49 ten times based on the same initial placement whose area is 102 mm^2. The resulting areas are plotted as functions of the running times (sec). Figs. 9(a)-(c) show the resulting curves of SP, TCG, and TCG-S, respectively. To see the detailed convergence rates, we show in Figs. 9 only the potions whose areas are smaller than 47 mm^2. As illustrated in Fig. 9(c), TCG-S converges very fast to desired solutions, and the results are very stable (≤ 37.5 mm^2 for all runs). In contrast, the convergence speed of SP is much slower than TCG-S and TCG, and the resulting areas are often larger than 39 mm^2. Further, there is a large variance in its final solutions. Based on the experimental results, we rank the convergence speed from the fastest to the slowest and the solution stability from the most stable to the least stable as follows: $\mathrm{TCG-S} \succ \mathrm{TCG} \succ \mathrm{SP}$. We note that the stability and convergence speed should be very important metrics to evaluate the quality

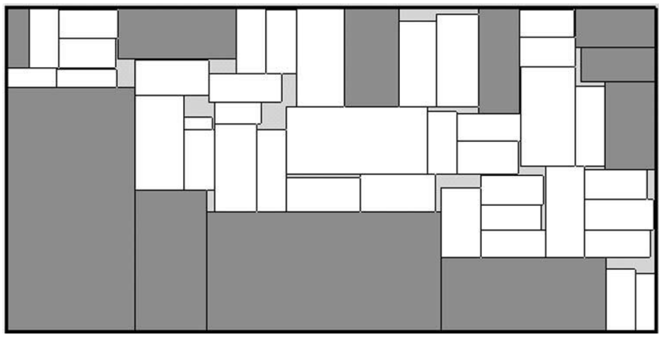


Fig. 10. Resulting placement of ami49 with boundary modules being heavily shaded $((\#|T|, \#|B|, \#|L|, \#|R|) = (3, 3, 2, 3)$, area $= 36.765$ mm^2). Note that the lightly shaded regions denote dead spaces.

of a floorplan representation because they reveal the corresponding solution structure for optimization. However, they were often ignored in previous works. (Most previous works focus on the comparison of solution space and packing complexity. Nevertheless, we find that the solution structure induced by a representation plays an even more important role in floorplan optimization.)

For the experiments with boundary modules, we compared TCG-S with the SP-based method in [7] using the same data generated by [7]. The second column of Table IV shows the numbers of the top, bottom, left, and right modules, denoted by $\#|T|$, $\#|B|$, $\#|L|$, and $\#|R|$, respectively. As shown in Table IV, TCG-S obtains smaller silicon areas than [7]. Fig. 10 shows the resulting placement for ami49 with boundary modules.

VIII. CONCLUDING REMARKS

We have presented the TCG-S representation for general floorplans by combining the advantages of two most promising P*-admissible representations TCG and SP and, at the same time, eliminating their disadvantages. We also have proposed a loglinear-time packing scheme for a P*-admissible representation. We note that this scheme can also be applied to most existing representations, such as SP and BSG. Based on our theoretical study and extensive experiments, we also have shown the superior capability, efficiency, stability, and flexibility of TCG-S for floorplan design.

ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their very constructive comments.

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. Design Automation Conf.*, 2000, pp. 458–463.
[2] T. Cormen, C. Leiserson, R. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd ed. New York: MIT Press/McGraw-Hill, 2001.
[3] P.-N. Guo, C.-K. Cheng, and T. Yoshimura, "An O-tree representation of nonslicing floorplan and its applications," in *Proc. Design Automation Conf.*, 1999, pp. 268–273.
[4] X. Hong, G. Huang, T. 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. Int. Conf. Computer-Aided Design*, 2000, pp. 8–12.
[5] Y. Hu, J. Lai, and T.-C. Wang, "A complete set of operations for perturbing transitive closure graphs," , 2003, submitted for publication.
[6] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, May 1983.

[7] J.-B. Lai, M.-S. Lin, T.-C. Wang, and L.-C. Wang, "Module placement with boundary constraints using the sequence-pair," in *Proc. Asia South Pacific-Design Automation Conf.*, 2001, pp. 515–520.

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

[9] J.-M. Lin and Y.-W. Chang, "TCG: A transitive closure graph-based representation for nonslicing floorplans," in *Proc. Design Automation Conf.*, 2001, pp. 764–769.

[10] ——, "TCG-S: Orthogonal coupling of $\mathbf{P}^*$-admissible representations for general floorplans," in *Proc. Design Automation Conf.*, 2002, pp. 842–847.

[11] ——, "TCG: A transitive closure graph-based representation for general floorplans," *IEEE Trans. VLSI Syst.*, 2003.

[12] Y. Ma, S. Dong, X. Hong, Y. Cai, C. K. Cheng, and J. Gu, "VLSI floorplanning with boundary constraints based on corner block list," in *Proc. Asia South Pacific-Design Automation Conf.*, 2001, pp. 509–514.

[13] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani, "Rectangle-packing based module placement," in *Proc. Int. Conf. Computer-Aided Design*, 1995, pp. 472–479.

[14] H. Murata, K. Fujiyoshi, and M. Kaneko, "VLSI/PCB placement with obstacles based on sequence pair," in *Proc. Int. Symp. Phys. Design*, 1997, pp. 26–31.

[15] S. Nakatake, K. Fujiyoshi, H. Murata, and Y. Kajitani, "Module placement on BSG-structure and IC layout applications," in *Proc. Int. Conf. Computer-Aided Design*, 1996, pp. 484–491.

[16] R. H. J. M. Otten, "Automatic floorplan design," in *Proc. Design Automation Conf.*, 1982, pp. 261–267.

[17] Y. Pang, C. K. Cheng, and T. Yoshimura, "An enhanced perturbing algorithm for floorplan design using the O-tree representation," in *Proc. Int. Symp. Phys. Design*, 2000, pp. 168–173.

[18] X. Tang and D. F. Wong, "FAST-SP: A fast algorithm for block placement based on sequence pair," in *Proc. Asia South Pacific-Design Automation Conf.*, 2001, pp. 521–526.

[19] D. F. Wong and C. L. Liu, "A new algorithm for floorplan design," in *Proc. Design Automation Conf.*, 1986, pp. 101–107.

A Unified Framework for Generating All Propagation Functions for Logic Errors and Events

Maria K. Michael, Themistoklis Haniotakis, and Spyros Tragoudas

***Abstract*—We present a generic framework that supports efficient generation of the traditional Boolean difference function of some output with respect to any line in a combinational circuit, which is important when testing for logic defects. The framework also allows for the generation of generalized Boolean difference functions, which reflect sensitivity on event propagation from a given line to some circuit output. This generalized function could apply in timing verification, analysis, and test. We implemented the proposed framework using various function representation environments, including binary decision diagrams, Boolean expression diagrams, and Boolean networks, and report experimental results on the ISCAS'85 and ISCAS'89 benchmarks.**

***Index Terms*—Automatic test pattern generation (ATPG), delay testing, testing, timing analysis, verification.**

I. Introduction

The problem of generating the propagation functions of all lines in a combinational (or fully scanned sequential) circuit is examined. The propagation function of a circuit line is a Boolean function that represents all the possible primary input assignments that allow for a change on the line to be observed (propagated) at some primary output.

Let a be any circuit line and z a primary output. For simplicity, we use the above symbols to also denote the functionality of the respective lines. The propagation function indicating whether line z is sensitive to logic changes on line a is also known as the *Boolean difference of function z with respect to function a*, and is defined as $(\partial z/\partial a)$.

Function-based automatic test pattern generation (ATPG) methods typically use the Boolean Difference function as their basis ([4], [12]) because it indicates whether an activated *logic error* propagates from line a to output z.[1] For this reason, the Boolean difference function is also called the *logic-error propagation function of z with respect to a*. Although function-based ATPG methods are slower than structural-based ATPG methods, they increasingly gain popularity due to advances in data structures that store and manipulate functions. These include binary decision diagrams (BDDs) [2], that are canonical forms, and noncanonical forms, such as Boolean networks (BNETs) [1], Boolean expression diagrams (BEDs) [7], conjunctive normal forms, among others. In addition, function-based methods inherently allow for such implicit representations of many test patterns per fault (in the case where BDDs are used, all test patterns per fault are represented) that can be easily manipulated using basic Boolean concepts to help in other test-related problems such as test set compaction ([6], [8]) and on-chip test-set embedding. Moreover, function-based methods can speed up the detection of those faults that are hard to detect by the structural methods. It is noted that function-based ATPG approaches as in [3], [5], [9], [14], and [15], among others, do not create distinct functions for fault activation and fault propagation in

Manuscript received October 9, 2002; revised June 18, 2003 and August 19, 2003. This paper was recommended by Associate Editor S. M. Reddy.

M. K. Michael is with the Electrical and Computer Engineering Department, University of Cyprus, CY-1678 Nicosia, Cyprus (e-mail: mmichael@ucy.ac.cy).

T. Haniotakis and S. Tragoudas are with the Electrical and Computer Engineering Department, Southern Illinois University, Carbondale, IL 62901 USA (e-mail: haniotak@siu.edu; spyros@engr.siu.edu).

Digital Object Identifier 10.1109/TCAD.2004.828112

[1]Logic error is understood as a fault (in the ATPG context) or a design error (in the verification context).