SIAM J. DISC. MATH.
Vol. 8, No. 2, pp. 258–280, May 1995

SLICEABLE FLOORPLANNING BY GRAPH DUALIZATION *

GARY K. H. YEAP† AND MAJID SARRAFZADEH‡

Abstract. Previous algorithms on rectangular dual graph floorplanning generate general floorplans which include the class of nonsliceable floorplans. We examine the framework of generating sliceable floorplans using the rectangular dual graph approach and present an algorithm that generates a sliceable floorplan if the input graph satisfies certain sufficient conditions. For general input, the algorithm is still able to generate sliceable floorplans by introducing pseudomodules where the areas occupied by the pseudomodules are used for wiring. For an n-vertex adjacency graph, the algorithm generates a sliceable floorplan in $O(n \log n + hn)$ time where h is the height of the sliceable floorplan tree.

Key words. very large scale integration (VLSI) floorplanning, sliceable floorplans, planar graphs, graph dualization

AMS subject classifications. 05C85, 68Q20, 68Q35, 68Q25, 68R10

1. Introduction.

The rectangular dual graph approach to very large scale integration (VLSI) floorplanning, introduced in [1], is based on the idea of preserving the planar adjacency of input modules. The adjacency requirements of a floorplan are specified by an adjacency graph. Each vertex represents a rectangular partition (module) of the chip and each edge represents an adjacency requirement between two modules. A floorplan and its adjacency graph exhibit a certain duality relation. To ensure the existence of a floorplan, the adjacency graph must be restricted to a special class of planar triangulated graphs. We call it a *rectangular admissible graph*. The characterization of the graph was introduced by Kozminski and Kinnen [1], [2]. They also introduced algorithms to generate floorplans based on this dual graph approach. Bhasker and Sahni [4] improved the time complexity by providing an algorithm which finds a floorplan in linear time if one exists. Sun and Sarrafzadeh [5] and Yeap and Sarrafzadeh [6] generalized the approach to incorporate modules with shapes more complicated than rectangles.

Previous approaches yielded general floorplans which include the class of nonsliceable floorplans. We focus our interest on generating the class of sliceable floorplans. A sliceable floorplan has several attractive features. Since it is inherently a tree structure (as opposed to a graph structure), it facilitates later phases of layout processing [9]. For example, a version of the floorplan sizing algorithm has been shown to be NP-complete for general floorplans but is optimally solvable for sliceable floorplans [10].

For some classes of rectangular admissible graphs, only nonsliceable floorplans exist. For example, the graph in Fig. 1 has a rectangular floorplan as shown. By inspection, one can verify that no sliceable floorplans exist for the graph. The exact characterization of this class of graphs is still unknown. Finding a sliceable floorplan for a rectangular admissible graph, if one exists, is an open problem. One major condition on the rectangular admissible graph is that it contains no complex cycle

* Received by the editors November 13, 1991; accepted for publication (in revised form) April 19, 1994. This work was supported in part by National Science Foundation grant MIP-8921540.

† Motorola, 2100 East Elliot Road, MD EL510, Tempe, Arizona 85284.

‡ Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, Illinois 60208.

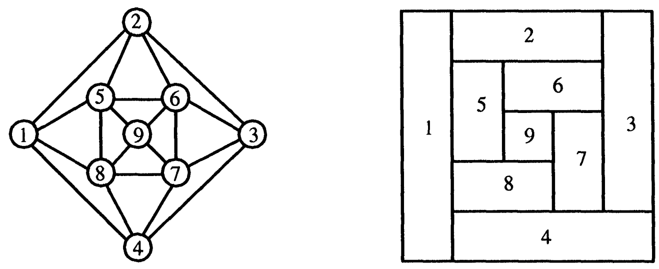


FIG. 1. *A nonsliceable rectangular admissible graph.*

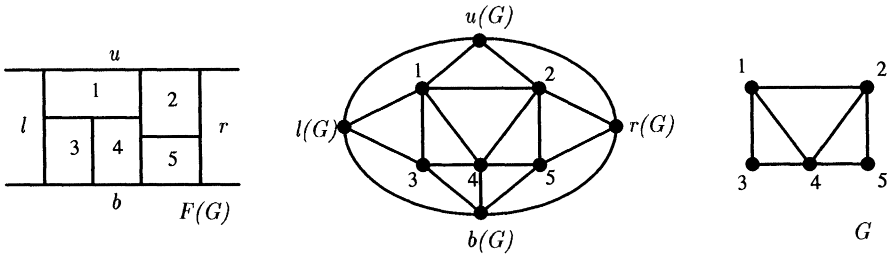


FIG. 2. *A floorplan $F(G)$, extended dual $EPTG(G)$, and dual G.*

of length 3. We will prove that with the additional constraint that the input graph contains no complex cycle of length 4, sliceable floorplans always exist.

When an adjacency graph does not admit a sliceable floorplan, one could introduce pseudovertices to the input graph. We add a pseudovertex on an edge when adjacency cannot be satisfied. Pseudovertices represent rectangular areas created purely to satisfy the adjacency requirements. The areas consist of only routing wires and contain no devices. The areas consumed by the pseudovertices depend on the density of the wires carried by the edges. If the wiring density of an edge is zero, adding pseudovertices will not waste any chip space. Zero edge density occurs when the original adjacency of modules is not a fully triangulated graph. In our algorithm, heuristic measures can be imposed to minimize the areas consumed by pseudovertices.

Section 2 of this paper defines the formal terminologies. Section 3 discusses a formal framework for sliceable rectangular floorplans. The sliceability requirements on the primal floorplan are translated to the dual graph, and the corresponding properties in the dual are examined. Section 4 presents an algorithm for generating sliceable floorplans based on the dual graph approach. Section 5 presents some application aspects of the floorplanning algorithm.

2. Preliminaries.

A *rectangular floorplan* (RFP) F is a plane graph where

(a) each edge is either a vertical or a horizontal line segment,

(b) each face is a rectangle,

(c) each vertex has degree 3,

(d) the boundary of F is a rectangle.

The restriction of a floorplan to degree 3 vertices does not lose generality. A floorplan with degree 4 vertices can be transformed into one with only degree 3 vertices by a minor modification [1]. We assume that F is bounded by four infinite faces r, u, l, b as shown in Fig. 2. Given F, the *extended dual* $EPTG(G)$ of F can be constructed as

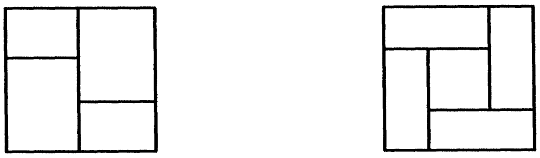


FIG. 3. *A sliceable floorplan (left) and a nonsliceable floorplan (right).*

follows:

(a) Each face of F corresponds to a vertex of G.

(b) There is an edge between two vertices of G if the two corresponding faces in F are adjacent.

If the vertices $r(G), u(G), l(G), b(G)$ and their incident edges are deleted from $EPTG(G)$, the resulting graph G is called the *dual* of F. The rectangular dual approach to floorplanning consists of obtaining a floorplan F from a given dual graph G.

Two different F's may have an identical dual G. A vertex in G which is adjacent to more than one vertex of $\{r(G), u(G), l(G), b(G)\}$ is called a *corner vertex*. The four corner vertices in G correspond to the four corner faces of F. The corner vertices in G need not be distinct if the corner faces of F are not distinct. An F with dual G is denoted by $F(G)$. If G is the dual of some floorplan F, we say that G *admits* a floorplan F.

Given G, unless otherwise stated we assume that the four corner vertices of G have been fixed, thus $EPTG(G)$ is known. We denote the corner vertices as $\mathrm{NW}(G), \mathrm{NE}(G), \mathrm{SE}(G)$, and $\mathrm{SW}(G)$ (northwest, northeast, southeast, and southwest). The selection of corner vertices is not a trivial process. If the corners are not selected carefully, the extended dual $EPTG(G)$ may not be rectangular admissible. Basically, the corners should be selected so that the addition of vertices $r(G), u(G), l(G), b(G)$ does not result in complex triangles (to be defined). The external edges (edges incident to the infinite face) of G are divided into four mutually disjoint subsets (possibly empty), denoted by $\mathrm{TOP_e}(G), \mathrm{LEFT_e}(G), \mathrm{BOTTOM_e}(G)$, and $\mathrm{RIGHT_e}(G)$. The corresponding vertices (incident to the infinite face) are denoted by $\mathrm{TOP_v}(G), \mathrm{LEFT_v}(G)$, $\mathrm{BOTTOM_v}(G)$, and $\mathrm{RIGHT_v}(G)$. If G is a dual of F, all internal faces of $EPTG(G)$ are triangles because vertices of F have degree 3. This is also true for G since it is a subgraph of $EPTG(G)$. For example, in Fig. 2,

$$\begin{aligned}
&\mathrm{NW}(G) = 1, \mathrm{NE}(G) = 2, \mathrm{SE}(G) = 5, \mathrm{SW}(G) = 3;\\
&\mathrm{TOP_e}(G) = \{(1,2)\}, \mathrm{LEFT_e}(G) = \{(1,3)\},\\
&\mathrm{BOTTOM_e}(G) = \{(3,4),(4,5)\}, \mathrm{RIGHT_e}(G) = \{(2,5)\},\\
&\mathrm{TOP_v}(G) = \{1,2\}, \mathrm{LEFT_v}(G) = \{1,3\},\\
&\mathrm{BOTTOM_v}(G) = \{3,4,5\}, \mathrm{RIGHT_v}(G) = \{2,5\}.
\end{aligned}$$

An RFP F is called *sliceable* if it is a rectangle or can be decomposed into two nonempty RFPs, F_1, F_2 by a vertical or horizontal line such that F_1 and F_2 are both sliceable. Examples of sliceable and nonsliceable floorplans are shown in Fig. 3. A sliceable RFP can be represented by a tree.

A *path* is an ordered set of adjacent vertices. A path can also be equivalently denoted by the set of edges in the path. If the two end vertices of a path are identical, it is called a *cycle*. A *complex cycle* (of length n) is a cycle C_n of n edges (of a plane graph) where there is at least one vertex in the finite region bounded by C_n. A complex cycle with vertices $\{v_1, \ldots, v_n\}$ is denoted by $C_n(v_1, \ldots, v_n)$. In particular, we are interested in C_3 and C_4.

A *chord free path* (CFP) in G is a path $P = (v_1, \ldots, v_n)$, where for all $i \neq j$,

(a) $v_i \neq v_j$, and

(b) if $(v_i, v_j) \in G$ then $|i - j| = 1$.

If a path $Q = (q_1, \ldots, q_n)$ is not a CFP in G, then either $q_i = q_j$ for some $i \neq j$ or there exist some edges $(q_i, q_j) \in G$, where $|i - j| \geq 2$. In the latter case, the edges (q_i, q_j) are called *chords* of path Q.

A *vertical slice* is an ordered set of edges $E_s = (e_1, \ldots, e_n)$ in G, where

(a) $e_1 \in \text{TOP}_\text{e}(G), e_n \in \text{BOTTOM}_\text{e}(G)$,

(b) for $1 < i < n, e_i \notin \text{TOP}_\text{e}(G) \cup \text{BOTTOM}_\text{e}(G) \cup \text{LEFT}_\text{e}(G) \cup \text{RIGHT}_\text{e}(G)$;

(c) G is decomposed into exactly two nonempty components G_l (left subgraph) and G_r (right subgraph) when E_s is removed;

(d) for any $e \in E_s$, adding e causes G_l and G_r to be connected.

A *horizontal slice* is defined similarly where $e_1 \in \text{LEFT}_\text{e}(G)$ and $e_n \in \text{RIGHT}_\text{e}(G)$, and G is decomposed into G_u (upper subgraph) and G_b (lower subgraph). A *slice* is either a vertical or horizontal slice. Let $e_1 = (v_1, w_1) \in \text{TOP}_\text{e}(G)$ and $e_n = (v_n, w_n) \in \text{BOTTOM}_\text{e}(G)$ be edges of a vertical slice with $v_1, v_n \in G_l$ and $w_1, w_n \in G_r$. When G is decomposed into G_l and G_r by the vertical slice, the four corner vertices of G_l and G_r are well defined:

$$\begin{aligned} &\text{NW}(G_l) = \text{NW}(G), \quad \text{NE}(G_l) = v_1, \quad \text{SE}(G_l) = v_n, \quad \text{SW}(G_l) = \text{SW}(G); \\ &\text{NW}(G_r) = w_1, \quad \text{NE}(G_r) = \text{NE}(G), \quad \text{SE}(G_r) = \text{SE}(G), \quad \text{SW}(G_r) = w_n. \end{aligned}$$

EPTG(G_l) can be constructed by adding $r(G_l)$ and edges $\{(r(G_l), v_1), \ldots, (r(G_l), v_n)\}$. *EPTG*$(G_r)$ can be constructed similarly by adding $l(G_r)$. Note that we may have $v_1 = v_n$ and/or $w_1 = w_n$. Figure 4 shows an example of decomposition by a vertical slice.

A vertical slice E_s on *EPTG*(G) defines a *left boundary path* $P_l(E_s) = (u(G), v_1, \ldots, v_n, b(G)), v_i \in \text{RIGHT}_\text{v}(G_l)$, and a *right boundary path* $P_r(E_s) = (u(G), w_1, \ldots, w_m, b(G)), w_i \in \text{LEFT}_\text{v}(G_r)$ (see Fig. 4). Note that $(v_1, \ldots, v_n) \cap (w_1, \ldots, w_m) = \phi$ and $(v_1, \ldots, v_n) \cup (w_1, \ldots, w_m)$ is the set of vertices of E_s. Boundary paths of a horizontal slice are defined similarly. When a vertical slice is specified, the left and right boundary paths are well defined; conversely, when the left or right boundary paths are specified, vertical slice can be defined. Given a right boundary path $P = (v_1, \ldots, v_n)$, where $v_1 \in \text{TOP}_\text{v}(G)$ and $v_n \in \text{BOTTOM}_\text{v}(G)$, we can construct a slice E_s by taking all edges incident to vertices of P and on the left (but excluding edges) of P. If such E_s satisfies the condition of a slice, we call E_s the *left-induced slice* of path P. Similarly, we can define the *right-induced slice* of P. If an E_s so constructed is not a slice, the left- (right-) induced slice of P is undefined.

A *vertical slice* in F (not in G) is a set of edges $S = (e_1, \ldots, e_n)$ of a nonboundary path in F, where

(a) all e_i's are on a common vertical cut-line;

(b) e_1 is incident to the top boundary of $F(G)$; and

(c) e_n is incident to the bottom boundary of $F(G)$.

Horizontal slice and *slice* on F are defined similarly. In a sliceable RFP, at least one slice exists.

3. Properties of sliceable floorplans and their duals.

Many properties of a floorplan are reflected on its corresponding dual and vice versa. Some of the properties have been investigated in earlier literature [1]–[6]. Lemma 1 below was proved in [1].

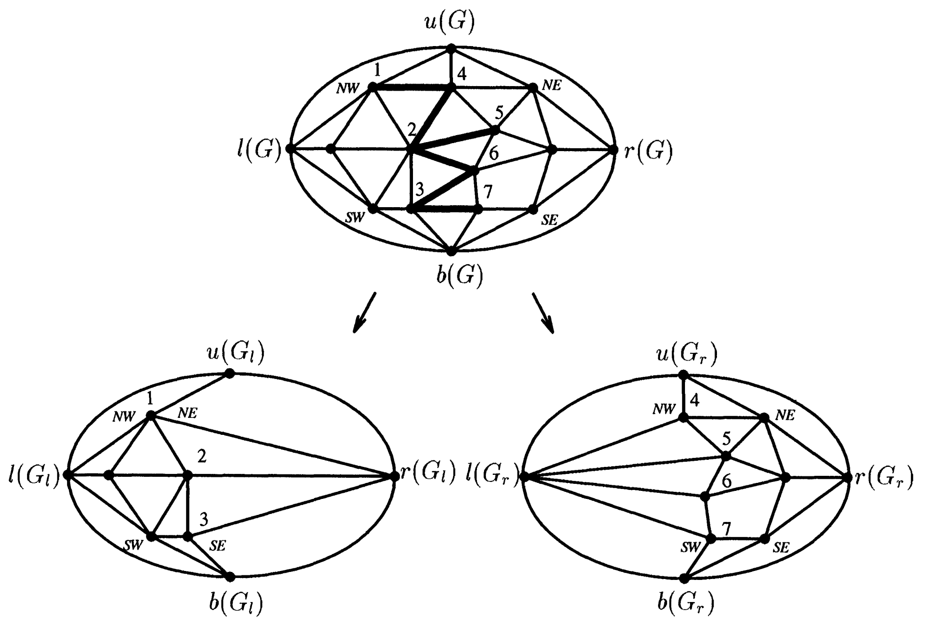


FIG. 4. *Decomposition by a vertical slice.* $E_s = \{(1,4),(2,4),(2,5),(2,6),(3,6),(3,7)\}, P_l(E_s) = (u(G),1,2,3,b(G)), P_r(E_s) = (u(G),4,5,6,7,b(G))$.

LEMMA 1. *Let G be a planar triangulated graph with corner vertices fixed. G admits an RFP if and only if $EPTG(G)$ contains no C_3 (complex cycle of length 3).*

LEMMA 2. *Let G be a planar triangulated graph which admits an RFP. Let E_s be a vertical slice on G, and G_l, G_r be two subgraphs decomposed by E_s. If both boundary paths $P_l(E_s), P_r(E_s)$ are CFPs, then both G_l and G_r admit RFPs.*

Proof. Let $P_l(E_s) = (p_1, \ldots, p_n)$. Since G admits an RFP, by Lemma 1 $EPTG(G)$ contains no C_3. Consider the $EPTG(G_l)$. Suppose for the sake of contradiction that G_l does not admit an RFP. By Lemma 1, there is a $C_3(a,b,c)$ in $EPTG(G_l)$. Since $EPTG(G)$ contains no C_3, at least one of the edges of $C_3(a,b,c)$ is not in $EPTG(G)$. The only edges that exist in $EPTG(G_l)$ but not in $EPTG(G)$ are incident to $r(G_l)$ (refer to Fig. 4). Since $P_l(E_s)$ is a CFP, any edge incident to $r(G_l)$ cannot form a C_3. Thus, $EPTG(G_l)$ contains no C_3, a contradiction. By Lemma 1, $EPTG(C_l)$ admits an RFP. Similarly, $EPTG(G_r)$ also admits an RFP. A similar lemma applies to horizontal slices. □

A slice E_s that satisfies the conditions that both boundary paths $P_1(E_s)$ and $P_2(E_s)$ are CFPs is called a *proper slice.* When a left floorplan F_l and a right floorplan F_r are placed side by side to form a floorplan, we denote it as $F_l|E_s|F_r$, where E_s is the corresponding slice of the floorplan.

LEMMA 3. *Let E_s be a proper vertical slice on G which admits an RFP. Let G_l and G_r be the left and right subgraphs decomposed by E_s. $F(G_l)|E_s|F(G_r)$ has dual G.*

Proof. The dual of $F(G_l)|E_s|F(G_r)$ can be constructed by embedding G_l on the left and G_r on the right with E_s connecting them. The resulting graph is exactly G. □

LEMMA 4. *Let $F(G)$ be a sliceable RFP. Let $S = (e_1, \ldots, e_n)$ be a slice of $F(G)$. The corresponding set of dual edges $E_s = (e'_1, \ldots, e'_n)$ in G is a proper slice.*

Proof. Since S consists of only vertical (horizontal) line segments, it is a union of

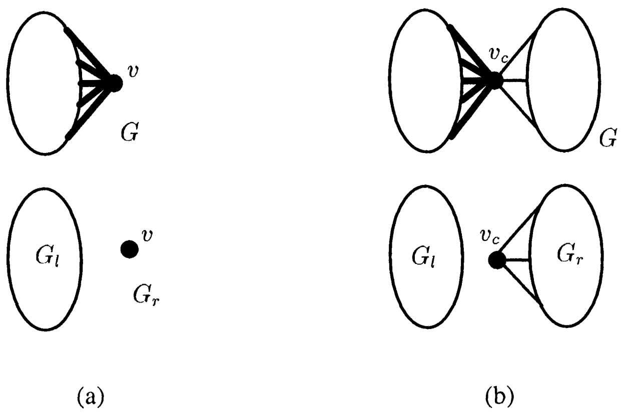


FIG. 5. *Slicing G for special cases.* (a) *nondistinct corner vertex v.* (b) *G contains a cut vertex v_c.*

two floorplans F_l and F_r with S as the common boundary. Let G_l and G_r be the dual of F_l and F_r. By Lemma 1, $EPTG(G_l)$ and $EPTG(G_r)$ contain no C_3. Therefore, there are no chords on the left and right boundary paths of E_s. □

THEOREM 1. *Let G be a planar triangulated graph which admits an RFP. If G contains no C_4 (complex cycle of length 4), then it admits a sliceable RFP.*

Proof. We prove the theorem by showing that, given an arbitrary planar triangulated graph G, where $EPTG(G)$ contains no C_3 and G contains no C_4, there exists at least one proper slice E_s.

If G contains only one vertex, the proof is trivial. Since $EPTG(G)$ is given, the corner vertices are fixed. If there is a vertex $v \in G$ which occupies exactly two corners (no vertex can occupy exactly 3 corners), simply let E_s be the set of edges incident to v. Since G contains no C_3, E_s is a proper slice. If G contains a cut vertex v_c, let E_s be the set of edges incident to the left (or right) of v_c. The graphs in these cases are shown in Fig. 5. Thick lines show proper slices E_s.

We are only left with the cases where the four corners of G are distinct and G contains no cut vertices. We label the vertices of $EPTG(G)$ using the following procedure:

(a) Delete $u(G), l(G), b(G)$ and their incident edges.

(b) Perform a breath first search (BFS) traversal on the remaining graph with $r(G)$ as the root. Label a vertex v at level i in a BFS tree with a superscript i, i.e., v^i.

We "redraw" the graph such that vertices on the same BFS level are aligned vertically. Figure 6 shows such a BFS drawing. Solid lines indicate edges that must exist in $EPTG(G)$. Dashed lines indicate that there may be zero or more edges/vertices. For the purpose of discussion, $u(G)$ and $b(G)$ are special vertices which can be at levels 1 and 2. In fact, we need not redraw the whole graph because we are only interested in the vertices and edges up to level 2. We will show that we can construct a proper vertical slice by considering the BFS drawing of the $EPTG(G)$.

Let $E(i,j)$ be the set of edges between levels i and j and let the vertices at level i be labeled $v_1^i, \ldots, v_{n_i}^i$ from the top (see Fig. 6). Consider the vertical slice

$$S_0 = E(1,2) - \{(u(G),x)|x \in V\} - \{(b(G),y)|y \in V\} \tag{1}$$

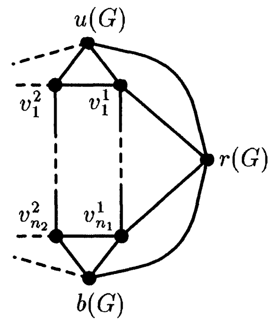


FIG. 6. *BFS drawing of an extended graph $EPTG(G)$.*

and its corresponding boundary paths

$$P_r(S_0) = (v_0^1 = u(G), v_1^1, \ldots, v_{n_1}^1, v_{n_1+1}^1 = b(G)),$$
$$P_l(S_0) = (v_0^2 = u(G), v_1^2, \ldots, v_{n_2}^2, v_{n_2+1}^2 = b(G)).$$

Since the corner vertices are distinct, $n_1, n_2 \geq 2$. An example of $EPTG(G)$ and S_0 is given in Appendix I.

If S_0 is a proper slice, we are done. If S_0 is not a proper slice, the right boundary path $P_r(S_0)$ is still a CFP because any edge $e = (v_i^1, v_j^1) \in E(1,1)$ is not a chord of $P_r(S_0)$; otherwise there exists a complex triangle $C_3(v_i^1, v_j^1, r(G))$. Thus the chords must be in $P_l(S_0)$. Because of the way S_0 is selected, the chords must appear on the left of $P_l(S_0)$.

A chord $(v_i^2, v_j^2), i < j$, where there are no other chords (v_k^2, v_l^2), where $k \leq i < j \leq l$, is called a *maximal chord.* Let $E_c = \{e_1, \ldots, e_c\}$ be the set of maximal chords of $P_l(S_0)$. For any chord $(v_i^2, v_j^2) \in E_c$, there is no chord (v_k^2, v_l^2), where $k \in [i+1, j-1]$ and $l \notin [i,j]$ (i.e., chords are pairwise "noncrossing"). This observation allows us to perform local modifications in the neighborhood of a maximal chord (v_i^2, v_j^2) to construct a proper slice.

The neighborhood of a maximal chord $e_p = (v_i^2, v_j^2) \in E_c, i < j$ can be depicted by the general configuration in Fig. 7(a). v_m is the vertex to the right of e_p which contributes to the triangular face (v_i^2, v_j^2, v_m). v_m must be located at level 2 or above. Figures 7(b) and (c) show some special cases of the general configuration when v_m is at level 2. Again, dashed lines indicate zero or more edges/vertices.

For each maximal chord $e_p = (v_i^2, v_j^2) \in E_c, i < j$, let

$E_{tz}(a,b,c,d) \subseteq E(1,2)$ be the set of edges bounded by the trapezoid region $(v_a^2, v_b^2, v_c^1, v_d^1)$, including the edges $(v_a^2, v_d^1), (v_b^2, v_c^1)$;

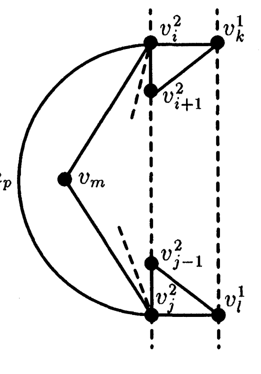
$E_p(i) \subset E(2,2)$ be the set of edges from (including) (v_i^2, v_{i+1}^2) to $e_p = (v_i^2, v_j^2)$ (not including) if we scan the edges incident to vertex v_i^2 clockwise; if $i = 0$, i.e., edge e_p is incident to $u(G), E_p(i = 0) = \phi$;

$E_p(j) \subset E(2,2)$ be the set of edges from (not including) $e_p = (v_j^2, v_i^2)$ to (v_j^2, v_{j-1}^2) (including) if we scan the edges incident to vertex v_j^2 clockwise; if $j = n_2 + 1$, i.e., edge e_p is incident to $b(G), E_p(j = n_2 + 1) = \phi$.

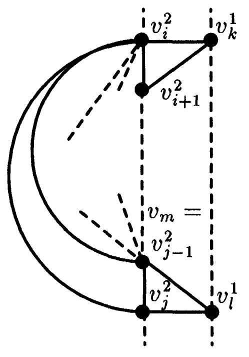
The regions of the edge sets $E_{tz}(i,j,l,k), E_p(i), E_p(j)$ of a maximal chord $e_p = (v_i^2, v_j^2)$ are depicted by the shaded regions in Fig. 8.

For each maximal chord $e_p = (v_i^2, v_j^2) \in E_c, p = 1, 2, \ldots, c$, we identify the two vertices v_k^1 and v_l^1 at level 1 (see Fig. 7) and obtain a new set S_p by

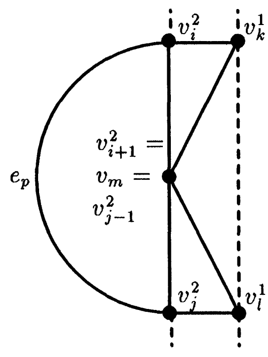
$$S_p = S_{p-1} - E_{tz}(i+1, j-1, l, k) + E_p(i) + E_p(j), \qquad p = 1, 2, \ldots, c,$$



(a)



(b)



(c)

FIG. 7. *Configurations of a maximal chord:* (a) *general configuration;* (b) *configuration* $v_m = v_{j-1}^2$; (c) *configuration when* $v_{i+1}^2 = v_m = v_{j-1}^2$.

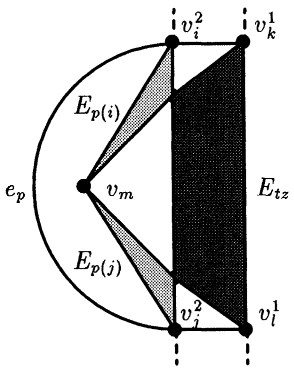


FIG. 8. *The regions of the sets* $E_{tz}(i,j,l,k), E_p(i)$, *and* $E_p(j)$ *of a maximal chord* e_p.

where S_0 is defined by equation (1). The above operation is called *bypassing a maximal chord* corresponding to e_p. An example of bypassing operation is shown in Fig. 9. The original slice (in thick lines) is shown on the left and the bypassed slice is shown on the right with maximal chord $e_p = (5, 8)$. After the bypassing operation is performed

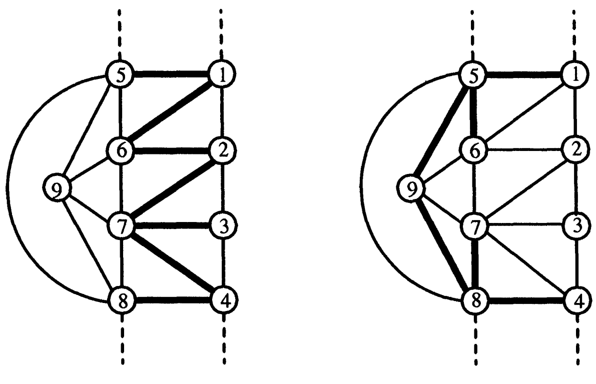


FIG. 9. *Bypassing a maximal chord c_p: original slice (left) and bypassed slice (right).*

on all maximal chords of E_c, we claim that the set $E_s = S_c$ is a proper slice ($c = |E_c|$). This implies that there exists at least one proper slice in G. Since the subgraphs G_l and G_r also contain no C_3 or C_4, the theorem applies recursively and a sliceable RFP exits.

To prove the claim, we observe that because of the bypassing operation for constructing E_s, the left boundary path $P_l(E_s)$ of $EPTG(G)$ is a CFP. It remains to show that $P_r(E_s)$ is also a CFP. There are two types of vertices in $P_r(E_s)$. Type 1 vertices are vertices at level 1 which are also found in the original $P_r(S_0)$ before bypassing. The other vertices are called type 2 vertices. They are added when a chord is bypassed, and by definition all type 2 vertices are not located at level 1 (see Fig. 8). If a type 2 vertex is added when a maximal chord $e_p = (v_i^2, v_j^2)$ is bypassed, we call it a type 2.p vertex. Each type 2.p vertex is adjacent to either v_i^2 or v_j^2. Suppose for the sake of contradiction that $P_r(E_s)$ contains a chord (x, y). The chord must be located on the right of $P_r(E_s)$. Consider the following cases.

Case 1 (x, y are both type 1 vertices). Since x and y are level 1 vertices, $x, y \in P_r(S_0)$. However, (x, y) cannot be a chord of $P_r(S_0)$ since $P_r(S_0)$ is a CFP. Therefore, (x, y) must be an edge of $P_r(S_0)$. It became a chord when we bypassed $e_p = (v_i^2, v_j^2)$ for some p. Referring to Fig. 7, we can see that this condition implies the existence of $C_4(x, y, v_i^2, v_j^2)$.

Case 2 (x, y are both type 2 vertices). From the construction of E_s, both vertices must belong to identical subtype 2.p. Suppose both vertices x, y are adjacent to v_i^2. This implies the existence of $C_3(x, y, v_i^2)$ (the chord (x, y) is on the right of $P_r(E_s)$). The same argument applies if both vertices are adjacent to v_j^2. If x is adjacent to v_i^2 and y is adjacent to v_j^2, it implies the existence of $C_4(x, y, v_j^2, v_i^2)$.

Case 3 (x is type 1 and y is type 2.p). Without loss of generality, let v_i^2 be the vertex adjacent to y. From Fig. 7, a chord (x, y) exists only when x is v_k^1 or v_l^1. If $x = v_k^1$, there exists $C_3(x = v_k^1, y, v_i^2)$. If $x = v_l^1$, there exists $C_4(x = v_l^1, y, v_i^2, v_j^2)$.

All possible cases above lead to contradictions by implying the existence of C_3 or C_4. Thus $P_r(E_s)$ is a CFP. □

An example illustrating an application of Theorem 1 is given in Appendix I. The converse of Theorem 1 is certainly not true, that is, there exist sliceable floorplans whose duals contain some C_4, for example, a C_4 with one vertex inside.

To help the development of a slicing algorithm, we observe the following lemma.

LEMMA 5. *The intersection of a slice E_s and any C_4 is either empty or contains exactly two edges.*

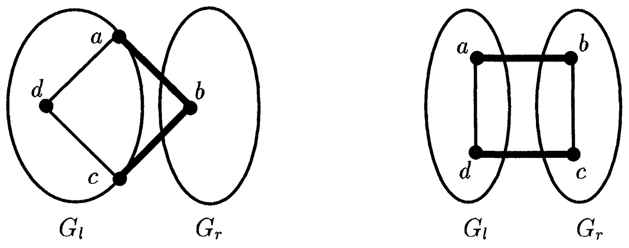


FIG. 10. *The two cases of slicing a C_4: corner-sliced (left) and center-sliced (right).*

Proof. The proof of the lemma follows immediately from the definition of E_s. Let I be the set of edges from the intersection of E_s and C_4. $|I|$ must be no greater than 4 since $|C_4| = 4$. Let G_l and G_r be the left and right subgraphs sliced by E_s. Suppose $|I| = 1$. Since the vertices of C_4 are still connected, they must belong to the same connected component. Thus we can add the edge in I without increasing the number of connected components in $G - E_s$. This contradicts the fact that E_s is a slice. If $|I| = 3$ and $G - E_s$ contain exactly two components, there must be at least one edge in I that can be added to $G - E_s$ without reducing the number of connected components, a contradiction. If $|I| = 4$, there are at least three connected components in $G - E_s$, i.e., G_l, G_r, and the vertices in the area bounded by the C_4; this is another contradiction. □

4. An algorithm for generating sliceable floorplans.

In this section we discuss some issues of slicing a planar triangulated graph (PTG) to generate a sliceable floorplan. The slicing operation is complicated by the presence of C_4 in a PTG. We first examine the properties of slicing a C_4. We then present an algorithm that generates a sliceable floorplan from a PTG. If the PTG contains no C_4, the algorithm always generates a sliceable floorplan. Otherwise, pseudovertices may be added to maintain sliceability in the floorplan. Special considerations are made when slicing a PTG containing C_4.

4.1. Slicing a C_4.

Consider a $C_4(a, b, c, d)$ and a slice E_s. Let E_i be the set of edges in the intersection of $C_4(a, b, c, d)$ and E_s. If E_i is not empty, then by Lemma 5, E_i contains exactly two edges. Without loss of generality, we can classify the two edges of E_i as follows:

(a) *corner-slice*: $E_i = \{(a, b), (b, c)\}$;

(b) *center-slice*: $E_i = \{(a, b), (c, d)\}$.

The two cases are shown in Fig. 10. When a C_4 is corner-sliced, $b \in G_r$ (without loss of generality) and $a, c, d \in G_l$, where a, c are vertices of the slice E_s. Consider the vertices of E_s which are adjacent to vertex b. Let $P_s = \{a = v_1, \ldots, v_n = c\}$ be the vertices. The existence of a chord (v_i, v_j) in P_s would imply a complex triangle $C_3(v_i, v_j, b)$. Thus no chords exist in P_s. Furthermore, edges (c, d) and (d, a) can never be chords of $P_l(E_s)$ because $d \notin P_l(E_s)$.

For a center-sliced $C_4, a, d \in G_l, b, c \in G_r$, and $a, b, c, d \in E_s$. Since the C_4 contains some vertices, either (a, d) is a chord of $P_l(E_s)$ or (b, c) is a chord of $P_r(E_s)$ (or both). Bypassing operation on the chords does not help because the condition holds regardless of the slice E_s. Thus, when a C_4 is center-sliced, we have to modify the input graph G to produce a sliceable floorplan.

From the above observations, we conclude that we need to corner-slice as many C_4's as possible to avoid chords on boundary paths of a slice. The number of C_4's in a PTG may exceed $O(n)$ where n is the number of vertices of the PTG. If we have to identify all C_4's, the time complexity will be dominated by the search for all C_4's. However, we could process C_4 hierarchically: a complex cycle C may contain other complex cycles in the region bounded by the edges of C. We define a *maximal* C_4 (denoted by MC_4) to be a C_4 which is not contained in any other C_4 of the PTG. The number of MC_4's in a PTG is $O(n)$. By definition, an MC_4 cannot contain another MC_4. When a PTG contains no C_3, two MC_4's do not intersect. We only consider an MC_4 of a PTG when searching for a proper slice. We try to find a corner-slice for each MC_4. Due to rotation symmetry, there are four different corner-slicings of an MC_4.

4.2. Slicing a planar triangulated graph. The proposed algorithm for generating a sliceable floorplan from an $EPTG(G)$ is a divide-and-conquer algorithm. The fundamental task of the algorithm is to construct a proper slice E_s from the $EPTG(G)$. From now on we will assume that the corners of $EPTG(G)$ are distinct and G contains no cut vertices. If G has nondistinct corners and/or contains cut vertices, generating a proper slice is trivial (see Fig. 5).

In general, an $EPTG(G)$ may have an exponential number of proper slices, while for others no proper slice may exist. The proof of Theorem 1 provides a method to construct a proper slice of an $EPTG(G)$. When G contains no C_4 a proper slice always exists. In the proof, the initial slice S_0 is chosen to be the edge incident to the vertices on the right boundary of G. The choice of such S_0 ensures that the right boundary path $P_r(S_0)$ is a CFP since $EPTG(G)$ contains no C_3. We could have chosen any slice S as long as $P_r(S)$ is a CFP. Different choices of S will yield different floorplans. Choosing an S where $P_r(S)$ is a CFP is not a difficult task. We start with an arbitrary path P, where the end vertices of P are vertices $u(G)$ and $b(G)$. If P is a CFP we construct S from the edges incident to P and on the left of P (the left-induced slice of P). If P is not a CFP we choose a maximal subset P' of vertices of P by traversing the chords so that P' is a CFP. We then choose the edges on the left of P' as S.

To search for a vertical CFP P, we define a search graph G_s, which is a directed subgraph of $EPTG(G)$, using the following procedure:

P1. For each $MC_4 = C_4(a,b,c,d)$ in G:

- **P1.1.** Delete all vertices (and incident edges) contained in $MC_4(a,b,c,d)$. Do not delete vertices a,b,c,d.
- **P1.2.** Add four directed edges $(a,b),(b,c),(c,d),(d,a)$ where the vertices a,b,c,d are in counterclockwise order. If adding such edges results in two directed edges (u,v) and (v,u), delete both edges.
- **P1.3.** Add two undirected edges (a,c) and (b,d).

P2. Delete all edges incident to $\mathrm{LEFT_v}(G)$.

P3. Delete all edges incident to $r(G)$.

P4. Delete all edges in $\mathrm{TOP_e}(G) \cup \mathrm{BOTTOM_e}(G)$.

P5. Change all undirected edges incident to u into directed edges away from u.

P6. Change all undirected edges incident to b into directed edges toward b.

P7. Change all undirected edges on $\mathrm{RIGHT_e}(G)$ into directed edges pointing downwards.

An example of $EPTG(G)$ and its search graph G_s is shown in Fig. 11. The undirected edges in G_s can be traversed in both directions. We search for a directed CFP P_s from $u(G)$ to $b(G)$ in G_s. If a diagonal edge (a,c) in an $MC_4(a,b,c,d)$ (in counterclockwise order) is traversed in G_s, the corresponding path in G should traverse

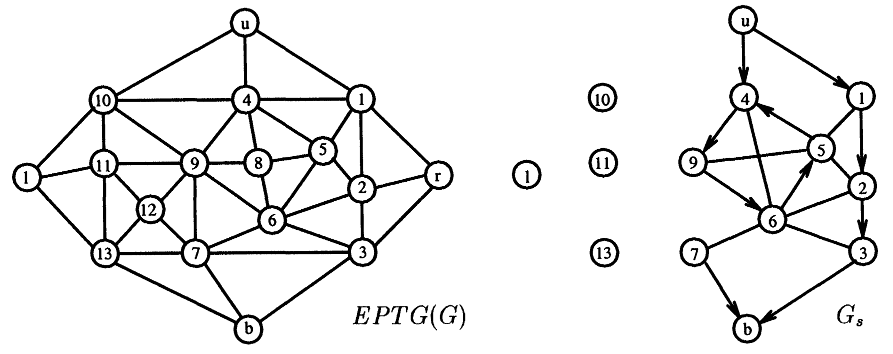


FIG. 11. *An EPTG(G) (left) and its search graph G_s (right).*

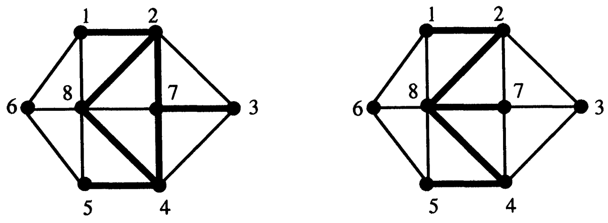


FIG. 12. *Modifying initial slice to satisfy the first condition of CFP.*

all vertices adjacent to vertex b in $MC_4(a, b, c, d)$. For example, if we traverse the edge (4, 6) of G_s in Fig. 11, we should traverse all vertices adjacent to vertex 5, i.e., (4, 8, 6) in G. From P_s in G_s, we find the corresponding path $P_r(S_0)$ in G. The left-induced slice S_0 of $P_r(S_0)$ serves as an initial slice. For example, in Fig. 11,

$$P_s = (u, 4, 6, 7, b) \in G_s,$$
$$P_r(S_0) = (u, 4, 8, 6, 7, b) \in EPTG(G),$$
$$S_0 = \{(4,10), (4,9), (8,9), (6,9), (7,9), (7,12), (7,13)\}.$$

Procedure **P1** guarantees that an MC_4 is corner-sliced and not center-sliced. **P1.2** is needed to avoid center-slicing an MC_4 on the right of path P_s. **P1.3** allows an MC_4 to be corner-sliced. **P2** is needed since G_l must contain all vertices of $\mathrm{LEFT_v}(G)$. (We have assumed that G contains no cut vertices.) **P3** eliminates paths through $r(G)$. The rest of the procedure simplifies the analysis: if $P_r(S_0)$ traverses $\mathrm{TOP_e}(G)$ or $\mathrm{BOTTOM_e}(G)$, then it must contain a chord; **P4** excludes this possibility. Since the end vertices of $P_r(S_0)$ are $u(G)$ and $b(G)$, **P5** and **P6** are included. **P7** is included since $\mathrm{RIGHT_v}(G)$ must be on G_r.

In general, there is more than one path in G_s. Heuristic measures can be employed to choose a good path. The search heuristics should try to corner-slice as many MC_4's as possible. Corner-slicing is equivalent to traversing a diagonal edge of an MC_4 or traversing exactly one vertex of an MC_4 when searching for P_s in G_s. If corner-slicing an MC_4 is not possible, we should try to avoid traversing any vertices of the MC_4. The unsliced MC_4 will appear in either G_l or G_r (exclusively). The MC_4 will be sliced when its subgraph is decomposed.

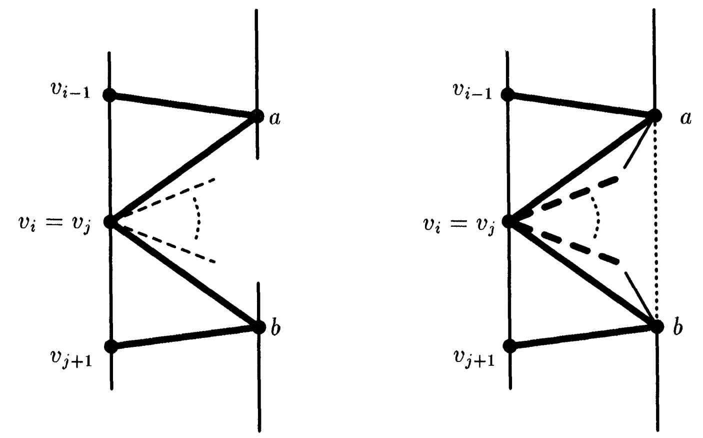


FIG. 13. *Initial slice (left) and modified initial slice (right).*

If no path exists in G_s, then there is no proper slice. In this case we have to accept an improper slice by center-slicing some MC_4. Again, we could impose heuristics to select an improper slice. When the presence of a chord is inevitable, we have to *fix the chord.* This will be discussed later in this section.

When the path P_s on G_s has been chosen, $P_r(S_0)$ in $EPTG(G)$ is determined. We obtain the left-induced slice S_0 of $P_r(S_0)$. By definition, $P_r(S_0)$ is also the right boundary path of the slice S_0. $P_r(S_0)$ is a CFP since we only accept CFPs when searching for P_s. There are two cases where the left boundary path $P_l(S_0) = (v_1, \ldots, v_n)$ fails to be a CFP:

(a) For some $i \neq j, v_i = v_j$.

(b) There are some chords in $P_l(S_0)$.

The two cases are derived from the definition of a CFP. The first case is easy to solve. We construct a path P_l' by excluding vertices $v_k, i < k < j$ from $P_l(S_0)$. The process is repeated until we obtain a path P_l^* where the first condition is not violated. The right-induced slice of P_l^* must exist and serves as the modified initial slice S_0'. An example of the procedure is shown in Fig. 12. The original slice is shown on the left with $P_r(S_0) = (2,3,4)$ and $P_l(S_0) = (1,8,7,8,5)$. On the right of the figure, $P_r(S_0) = (2,7,4)$ and $P_l(S_0) = (1,8,5)$.

We claim that the right boundary path $P_r(S_0')$ of modified slice S_0' is still a CFP. Let $v_i = v_j, i < j$ in the original slice S_0. The neighborhood of the graph for the modified slice S_0' is shown in Fig. 13. Thick edges represent edges in the slices S_0 and S_0'. Dotted edges may or may not exist. Solid edges and vertices must exist as shown in the configuration. From the figure, we can see that all vertices between vertices a and b in $P_r(S_0')$ are adjacent to $v_i = v_j$. Thus there are no chords (x, y) among the vertices or we would have $C_3(x, y, v_i = v_j)$. Also, the vertices cannot form any chords with any vertices in the original $P_r(S_0)$ due to the subpath of $P_r(S_0)$ between vertices a and b (shown on the right with a dotted vertical line).

From the above construction, we can thus assume that the first condition of an initial slice S_0 is always satisfied. The only conditions that need to be examined are the chords of $P_l(S_0)$. If G does not contain any C_4, we can apply the bypassing operation described in Theorem 1 to obtain a proper slice. In the presence of C_4, the operation is still successful if the maximal chord is not part of any C_4.

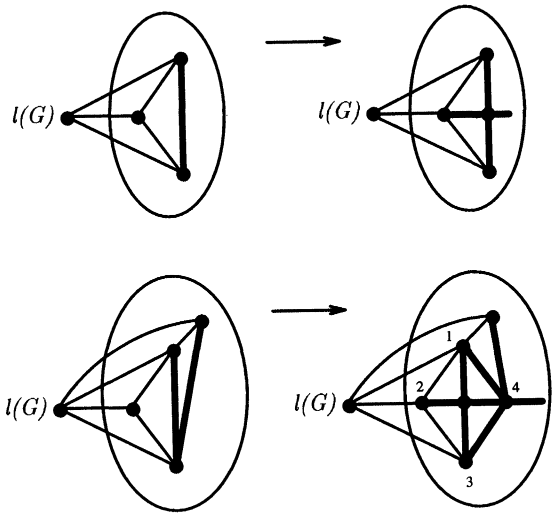


FIG. 14. *Fixing the chords of an improper slice.*

When all of the above methods fail and a chord cannot be avoided, the algorithm has to accept an improper slice. When this occurs, each chord will contribute a C_3 in one of the subgraphs, say $EPTG(G_r)$. We delete the chord, add a vertex in the middle of the chord and four new edges. The operation is called *fixing a chord*, as shown in Fig. 14. Fixing a chord eliminates the C_3 caused by the chord and maintains triangulation of graph $EPTG(G_r)$. Note that when we fix a chord, a new C_4 is created (e.g., $C_4(1, 2, 3, 4)$ of Fig. 14). However, three vertices of the C_4 fall in the boundary of G. This ensures that the C_4 will be corner-sliced in the future. C_3's of $EPTG(G_l)$ are handled similarly.

A summary of the algorithm is described as follows.

```
ALGORITHM SLICE($EPTG(G)$)

INPUT: An extended dual $EPTG(G)$.
OUTPUT: A planar triangulated graph $G'$ and a sliceable floorplan $F(G')$. $G'$ is $G$
        with some pseudovertices added. If $EPTG(G)$ contains no $C_3$ and $G$ contains
        no $C_4$, then $G' = G$.
BEGIN
    1. Check if:
        1.1. the corners of $EPTG(G)$ are not distinct.
        1.2. $G$ contains a cut vertex.
        if so, a slice is trivially generated.
    2. Find all $MC_4$'s of $G$.
    3. Determine the orientation of the slice: Horizontal or Vertical.
    4. Construct the search graph $G_s$.
    5. Search for a path $P_s$ in $G_s$ which satisfies the following criteria:
        Priority 1: Maximize the number of $MC_4$'s which are corner-sliced by $P$.
        Priority 2: Minimize the number of $MC_4$'s which are center-sliced by $P$.
    6. Find the left-induced slice $S_0$ in $G$ from $P_s$.
```

```
  7. Modify the slice $S_0$ to eliminate vertices of $P_l(S_0)$ so that the first condition
     of CFP is not violated.
  8. Find the maximal chords of $P_l(S_0)$.
  9. For each maximal chord $e_p$, perform a bypass operation on $e_p$ and obtain a
     final slice $E_s$. /* see Theorem 1 */
 10. Decompose $G$ into $G_l$ and $G_r$ and obtain $EPTG(G_l)$ and $EPTG(G_r)$.
 11. If $EPTG(G_r)$ or $EPTG(G_l)$ contains a $C_3$ due to chords, fix the chords by
     adding a pseudovertex and corresponding edges. Update $G'$ if pseudovertices
     are added. /* see Fig. 14 */
 12. Recursively call SLICE($EPTG(G_l)$) and SLICE($EPTG(G_r)$).
 13. Construct floorplan $F(G')$ by merging the floorplans of $F(G'_l)$ and $F(G'_r)$.
END.
```

The MC_4 of a graph can be found in $O(n \log n)$ time, where n is the number of vertices of G. The operation need not be repeated during the recursive steps since the MC_4's of G_l and G_r are also those of G. An $O(n \log n)$ algorithm for finding all MC_4's is described in Appendix II.

The search graph G_s can be constructed in linear time. The path P can also be found in linear time using simple heuristics, e.g., using a depth-first search which incorporates some cost measure to achieve balanced slice. The bypassing operation also has linear time complexity since the number of edges in a planar graph is $O(n)$. Standard planar graph data structures like a doubly connected edge list [11] are sufficient for the algorithm. All other steps in the algorithm are standard operations on planar graphs which can be achieved in $O(n)$ time.

The time complexity of the recursive part of the algorithm is

$$T(n) = T(n_l) + T(n_r) + O(n),$$

where n_l and n_r are the number of vertices of G_l and G_r, respectively, with $n_l + n_r = n$. The operation of fixing a chord adds at most c vertices where c is the number of chords. Since $c < n$, the time complexity remains unchanged. At each level of the recursion, we spend $O(n)$ time. If we consider the output floorplan tree, we spend $O(n)$ time in total to construct the tree nodes at each level. Thus we have

$$T(n) = O(hn),$$

where h is the height of the floorplan tree.

In general, n_l and n_r are not balanced. In fact, there exist instances of input $EPTG(G)$ where the trees are extremely unbalanced. The graph shown in Fig. 15 is an example. The graph belongs to a class of $EPTG(G)$ called *4-contractable*. The floorplan of a 4-contractable graph is unique [8] as given by the figure; thus $h = O(n)$. h is $\theta(\log n)$ when a balanced partition can be achieved for the nonterminal nodes of the floorplan tree. Since finding MC_4 requires $O(n \log n)$, the time complexity of the algorithm is $O(n \log n + hn)$.

5. Algorithm improvements and extensions.

5.1. Minimizing wasted area. When a proper slice does not exist, we perform a chord fixing operation. For each chord, a pseudovertex is added to the original G. These pseudovertices contain no circuit elements in VLSI layout. They are added merely to satisfy the adjacency requirements specified by G. In the final chip layout, the rectangles corresponding to the pseudovertices contain only routing wires with

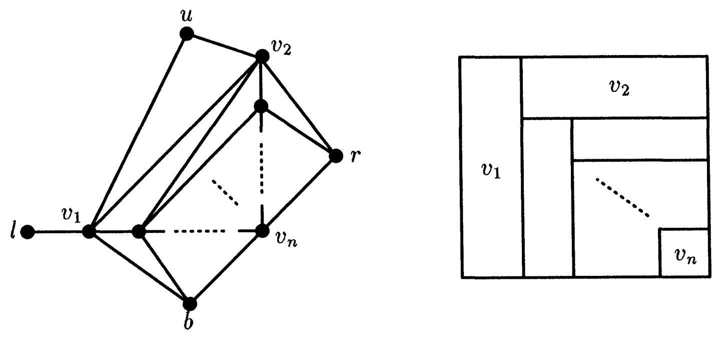


FIG. 15. *An inherently unbalanced floorplan.*

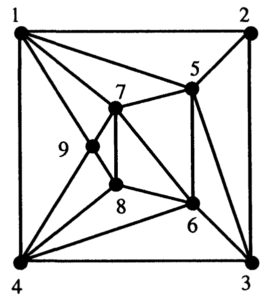


FIG. 16. *Choosing a slice to minimize wasted area.*

no devices. Minimizing the area consumed by these pseudovertices becomes a major concern. The routing area depends on the number of routing wires of the edge being fixed. Therefore, the slicing algorithm should prefer chords with less routing wires.

Proper choice of a slice also helps in reducing the wasted area. Consider $C_4(1, 2, 3, 4)$ in Fig. 16. Suppose corner-slicing of the C_4 is not possible. We have to slice the C_4 with edges (1, 2), (3, 4). If we choose the slice $\{(1,2),(1,5),(1,7),(1,9),(4,9),(4,8),(4,6),(4,3)\}$, we have three chords (2, 3), (5, 6), and (7, 8). However, if we choose $\{(1,2),(2,5),(3,5),(3,6),(3,4)\}$, we have only one chord (1, 4). The proper choice of slicing depends on the wiring density of the edges and the configuration of the C_4. The objective is to find a slice with less wasted area after pseudovertices are added to fix the chords.

5.2. Generating families of sliceable floorplans and sizing. In general, there is a family of floorplans which have identical dual G. An algorithm for generating all floorplans in the family is an immediate extension of the basic algorithm. In [2], algorithms for generating all floorplans for an input planar triangulated graph were discussed. In [7], an algorithm that generates all floorplans with linear time complexity per floorplan was reported. It should be noted that the number of floorplans in a family may be exponential in terms of the number of vertices.

Our algorithm is immediately extended to generate all sliceable floorplans of the input graph. Two sliceable floorplans are distinct if and only if their tree representations are distinct. The problem of enumerating all floorplans is reduced to the task of enumerating all proper slices on the search graph G_s. Any algorithm for enumerating paths can be employed.

5.3. Pseudovertices. Figure 17 demonstrates the outputs of the algorithm

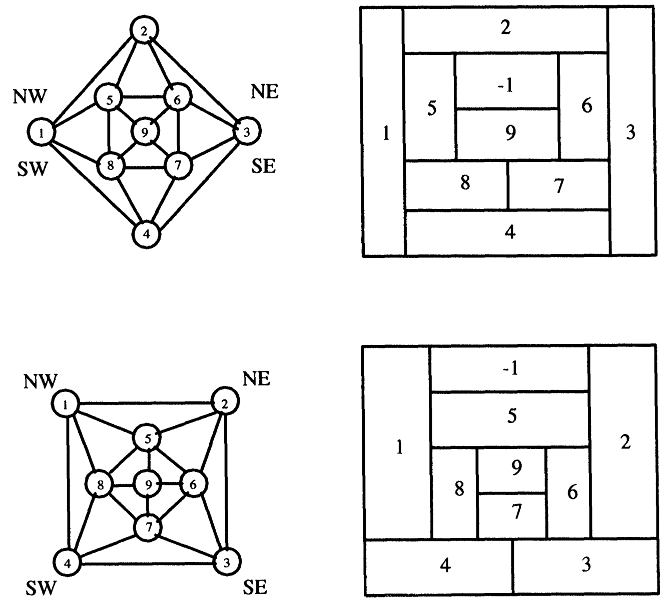


FIG. 17. *Adding pseudovertices to maintain sliceability.*

where pseudovertices are added to satisfy the sliceability requirement. The inputs are taken from graph of Fig. 1, which does not admit any sliceable floorplan. The two extended graphs show identical adjacency graphs with different corner assignments. The outputs of the program are also shown in the figure. The negative numbered rectangles in the floorplans represent pseudomodules.

Experiments using randomly generated planar triangulated graphs have shown that the number of pseudovertices added to the input graph is approximately 22% of the number of input vertices. On the pseudovertices added, approximately 70% (16% of the number of input vertices) are generated by C_3 of the original input. This represents the inherent price we have to pay for using the rectangular dual graph approach. But with only an additional 6% more vertices, we are able to generate sliceable floorplans. Since sliceable floorplans have many desirable characteristics, it is justifiable to pay some extra cost by applying our algorithm.

6. Conclusions and future work.

An algorithm for generating sliceable floorplans based on a rectangular approach has been presented. If the input graph contains no C_4 (complex cycle of length 4), then the algorithm always generates a sliceable floorplan. In the presence of C_4, a sliceable floorplan is also possible though not guaranteed. With minor modifications to the adjacency requirements of an input graph, the algorithm always generates a sliceable floorplan even when the input graph is known to be inherently nonsliceable. Typically, the sliceability requirement only introduces a small number of pseudovertices to the original input graph, which makes the algorithm very suitable for practical applications.

Many problems remain unsolved in the rectangular dualization approach to floorplanning. An important issue is the planarization and triangulation of the given adjacency graph to obtain a "good" planar triangulated graph. For example, the qualitative effect of planarization to the final floorplan is still unknown. Practical issues such as incorporating weights to vertices and edges of the graph are not well formulated. In particular, the effects of weight measures on the choice of slices are not

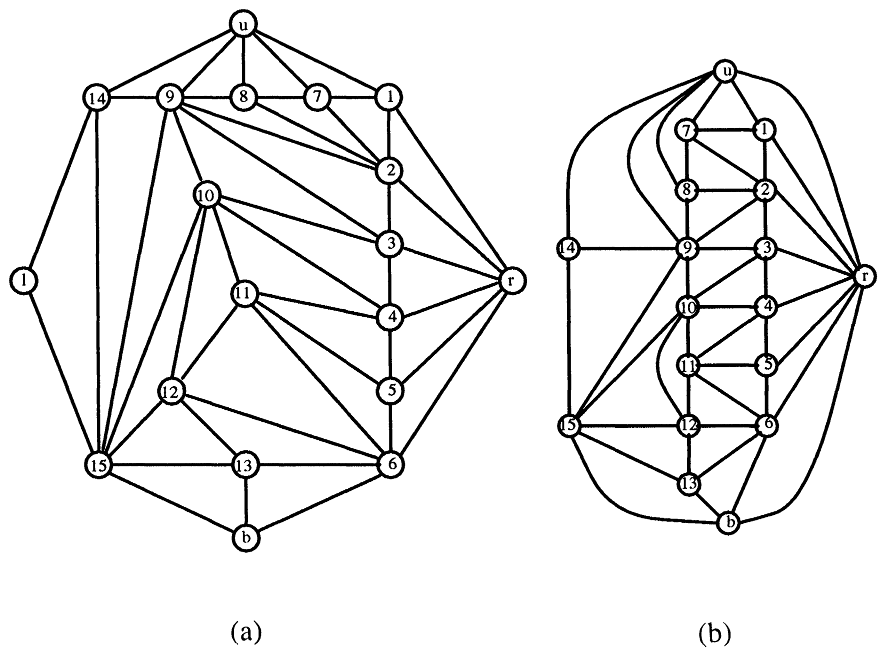


FIG. 18. *Construction of a proper slice when four corner vertices are distinct.* (a) *an EPTG(G) with no C_3 and C_4.* (b) *breadth first search drawing of EPTG(G).*

well understood. The problem of incorporating sizing in the floorplanning system is currently under investigation. Another interesting problem is the enumeration of all sliceable floorplans, currently being investigated by the authors.

Appendix I. An example for the proof of Theorem 1.

Figure 18(a) shows an $EPTG(G)$ with no C_3 and G with no C_4. The BFS drawing of the graph is depicted in Fig. 18(b). The construction of a proper slice E_s is as follows:

$$\begin{aligned}
&S_0 = \{(1,7),(2,7),(2,8),(2,9),(3,9),(3,10),(4,10),\\
&\qquad (4,11),(5,11),(6,11),(6,12),(6,13)\},\\
&P_l(S_0) = \{u,1,2,3,4,5,6,b\},\\
&P_r(S_0) = \{u,7,8,9,10,11,12,13,b\},\\
&\text{chords of } P_l(S_0) = \{(u,8),(u,9),(10,12)\},\\
&\text{maximal chords } E_c(S_0) = \{(u,9),(10,12)\},\\
&v_0^1 = u, v_1^1 = 1, v_2^1 = 2, v_3^1 = 3, v_4^1 = 4, v_5^1 = 5, v_6^1 = 6, v_7^1 = b,\\
&v_0^2 = u, v_1^2 = 7, v_2^2 = 8, v_3^2 = 9, v_4^2 = 10, v_5^2 = 11, v_6^2 = 12, v_7^2 = 13, v_8^2 = b.
\end{aligned}$$

For maximal chord $e_1 = (v_i^2, v_j^2) = (v_0^2, v_3^2) = (u, 9)$,

$$\begin{aligned}
&i = 0, j = 3, k = 1, l = 2, v_m = 8,\\
&E_{tz}(i+1, j-1, l, k) = E_{tz}(1,2,2,1) = \{(1,7),(2,7),(2,8)\},\\
&E_1(i) = E_1(0) = \phi, \qquad E_1(j) = E_1(3) = \{(8,9)\}.
\end{aligned}$$

For maximal chord $e_2 = (v_i^2, v_j^2) = (v_4^2, v_6^2) = (10, 12)$,

$$\begin{aligned}&i = 4, j = 6, k = 4, l = 6, v_m = 11,\\ &E_{tz}(i+1, j-1, l, k) = E_{tz}(5,5,6,4) = \{(4,11),(5,11),(6,11)\},\\ &E_2(i) = E_2(4) = \{(10,11)\}, \qquad E_2(j) = E_2(6) = \{(11,12)\}.\end{aligned}$$

To construct E_s,

$$\begin{aligned}S_1 &= S_0 - E_{tz}(1,2,2,1) + E_1(i=0) + E_1(j=3)\\ &= \{(8,9),(2,9),(3,9),(3,10),(4,10),(4,11),(5,11),(6,11),(6,12),(6,13)\},\\ E_s &= S_2 = S_1 - E_{tz}(5,5,6,4) + E_2(i=4) + E_2(j=6)\\ &= \{(8,9),(2,9),(3,9),(3,10),(4,10),(10,11),(11,12),(6,12),(6,13)\},\\ P_l(E_s) &= (u, 9, 10, 12, 13, b),\\ P_r(E_s) &= (u, 8, 2, 3, 4, 11, 6, b).\end{aligned}$$

It can be easily verified that E_s is a slice and $P_r(E_s), P_l(E_s)$ are CFPs.

Appendix II. Finding maximal C_4 of a planar triangulated graph. The algorithm requires a subalgorithm to detect MC_4. We present a divide-and-conquer algorithm to find all MC_4 of a PTG.

We partition the input graph G into vertex-balanced left and right subgraphs G_l, G_r with a slice $C = (e_1, \ldots, e_c)$. The slice need not strictly satisfy the condition that $e_1 \in \mathrm{TOP_e}(G)$ and $e_c \in \mathrm{BOTTOM_e}(G)$. We only require that e_1 and e_c be external edges while $e_i, 1 < i < c$, are not external edges. We call C a *cutset.* A vertex-balanced cutset can be easily generated by some tree search techniques (e.g., breadth first search) until half of the vertices are visited.

Let $P_l(C) = (v_0^l, \ldots, v_L^l)$ and $P_r(C) = (v_0^r, \ldots, v_R^r)$ be the left and right boundary paths of C. By Lemma 5, the intersection of an $MC_4(a,b,c,d)$ consists of either zero or two edges. At each recursion step, the algorithm finds all MC_4's that intersect the slice C. Referring to the discussions of §4.1, an $MC_4(a,b,c,d)$ is either corner-sliced or center-sliced. We consider both cases separately.

Case 1 (center-sliced: $(a,b),(c,d) \in C$). Without loss of generality, we assume that $a, d \in G_l$ and $b, c \in G_r$ (see Fig. 10). Since vertices $a, b, c, d \in P_l(C) \cup P_r(C)$, we can delete all vertices not in $P_l(C) \cup P_r(C)$. Now, the edges of $G_l(G_r)$ are only incident vertices of $P_l(C)(P_r(C))$. Let the remaining graph be G_d.

For the purpose of discussion, we assume that the vertices of $P_l(C)$ and $P_r(C)$ are distinct. The modifications needed when their vertices are not distinct will be discussed later. For each vertex v of $P_l(C)$ and $P_r(C)$, we find the minimum and maximum edge indices of C whose edges are incident to v. Let $v_{\min}$ and $v_{\max}$ be the indices. For example, if v is incident to edges $e_3, e_4, e_5, v_{\min} = 3$ and $v_{\max} = 5$. Let u, v be two vertices of $P_l(C)$ where u precedes v when $P_l(C)$ is traversed. There is an important monotonic property:

$$u_{\min} \le u_{\max} < v_{\min} \le v_{\max}.$$

The property also holds for $P_r(C)$.

Let the first edge of $C, e_1 = (a,b)$ with $a \in G_l$ and $b \in G_r$. Let $(a,d) \in G_l$ and $(b,c) \in G_r$ be the two external edges of G_d incident to vertices a and d, respectively. The situation is depicted in Fig. 19(a). We search for the existence of $MC_4(a,b,c,d)$.

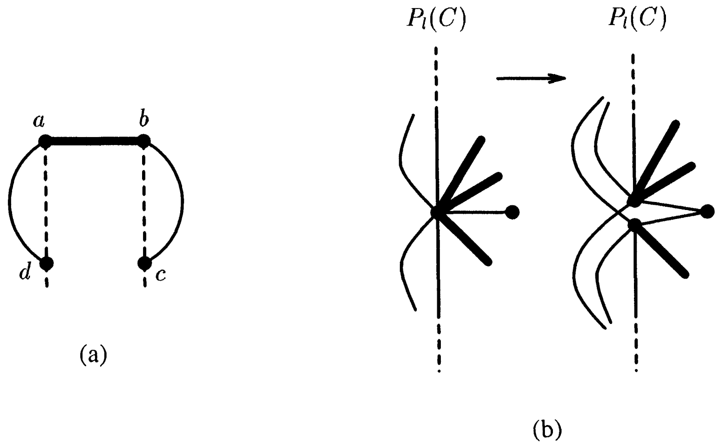


FIG. 19. *Finding $MC_4(a, b, c, d)$, where $(a, b), (c, d) \in C$.* (a) *finding MC_4 in Case* 1.1; (b) *duplicating vertices to maintain monotonic property.*

Let $[i, j]$ denote the set of integers between i and j (inclusive). Consider the min and max indices carried on vertices c and d. There are two possibilities:

(1) $[c_{\min}, c_{\max}] \cap [d_{\min}, d_{\max}] = \phi$;

(2) $[c_{\min}, c_{\max}] \cap [d_{\min}, d_{\max}] = K \neq \phi$.

In the first case, edge (c, d) does not exist. If $c_{\min} > d_{\max}$, we delete the external edge (b, c) in G_d. By the monotonic property, any subsequent vertex visited by the algorithm $d' \in P_l(C)$ incident to vertex a will have

$$d'_{\max} < d_{\min} \leq d_{\max} < c_{\min},$$

because d' precedes d in $P_l(C)$. Thus d' is not adjacent to c. Conversely, if $d_{\min} > c_{\max}$, we delete edge (a, d) for the same reason.

For the second case, $MC_4(a, b, c, d)$ exists with edge (c, d) in C. However, K must contain exactly one integer. Otherwise, we would have at least two distinct edges e_i, e_j with $i, j \in K, i \neq j$, simultaneously incident to vertices c and d. This is a contradiction since there are no duplicate edges in G.

If an MC_4 is found, all vertices and incident edges in the area bounded by the MC_4 are deleted. After all external edges incident to vertices a and b are exhausted, we delete (a, b) and select the first remaining edge in C (which is an external edge).

When $P_l(C)$ contains nondistinct vertices, the monotonic property fails. However, by duplicating the nondistinct vertices, we are able to maintain the monotonic property. Each time we encounter a vertex v which has been traversed in $P_l(C)$, we make duplicate vertices v' at a very small distance from v and redistribute the incident edges. The resulting graph may not be planar but it does not affect the solution. The operation is shown in Fig. 19(b). The thick edges represent edges in C. A similar procedure is applied to $P_r(C)$.

Case 2 (corner-sliced: $(a, b), (b, c) \in C$). Assume $a, c, d \in G_l$ and $b \in G_r$ (see Fig. 10). First, we delete the following edges and vertices since they will not be part of any $MC_4(a, b, c, d)$ we are searching for:

(a) all edges not incident to vertices of $P_l(C) \cup P_r(C)$;

(b) all remaining edges of G_r;

(c) all remaining vertices of degree less than 2;

(d) all remaining vertices not incident to the infinite face.

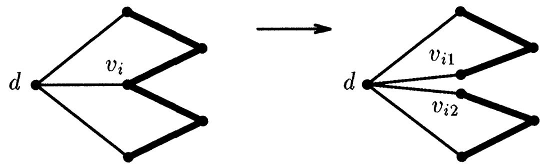


FIG. 20. *Duplicating vertices and edges to find $MC_4(a,b,c,d)$, where $(a,b),(b,c) \in C$.*

Let G_d be the resulting graph. Consider a vertex $d \in G_l - P_l(C)$. Let $\{(d,v_1),\ldots,(d,v_n)\}$ be the incident edges in G_d. Since $d \notin P_l(C), d$ cannot be adjacent to vertices of G_r. If $v_i \notin P_l(C)$ for some i, then the edge (d,v_i) would have been deleted in step (a). Thus $v_i \in P_l(C)$ for all i.

If an edge (d,v_i) is incident to the infinite face of G_d, we do nothing. If (d,v_i) is not incident to the infinite face, we duplicate the vertex v_i and the edge (d,v_i) as shown in Fig. 20. (The thick edges represent edges in C.) The duplicate vertices v_{i1} and v_{i2} are spaced a sufficiently small distance apart. Let (d,v_{i1}) and (d,v_{i2}) be the duplicate edges. Since the original v_i is incident to the infinite face (by step (d)), edges (d,v_{i1}) and (d,v_{i2}) are now incident to the infinite face. We perform such modification for all vertices v_i and all vertices $d \in G_l - P_l(C)$.

Consider an $MC_4(a,b,c,d)$ in G_d, where $a,c,d \in G_l$ and b in G_r. Because of the above modification, the edges (a,d) and (c,d) of the MC_4 are now incident to the infinite face. $MC_4(a,b,c,d)$ can be easily identified by considering all vertices b in $P_r(C)$. There is at most one MC_4 corresponding to each vertex $b \in P_r(C)$.

The algorithm is repeated with left and right subgraphs interchanged for $MC_4(a,b,c,d)$, where $b \in G_l$ and $a,c,d \in G_r$.

Combining the two cases above, we find all MC_4 with edges in the cutset C. The algorithm can be recursively applied to G_l and G_r to find other MC_4's. The algorithm is shown in pseudoinstructions below.

```
ALGORITHM FIND_MC4($G$)

INPUT: A planar triangulated graph $G$.
OUTPUT: All $MC_4$ of $G$.
BEGIN
    1. Find a balanced cutset $C$ where the left subgraph $G_l$ and right subgraph $G_r$
       are connected. For example, use breadth first search until half of the vertices
       in $G$ are visited.
    2. FIND_MC4_CASE_1($G_l, G_r, C$).
    3. FIND_MC4_CASE_2($G_l, G_r, C$).
    4. FIND_MC4_CASE_2($G_r, G_l, C$).
    5. FIND_MC4($G_r$).
    6. FIND_MC4($G_l$).
END.

PROCEDURE FIND_MC4_CASE_1($G_l, G_r, C$)

INPUT: A cutset $C$ and left and right subgraphs $G_l, G_r$.
OUTPUT: All $MC_4(a,b,c,d)$ where $a,d \in G_l$ and $b,c \in G_r$.
BEGIN
    1. Delete all vertices $v \notin P_l \cup P_r$.
```

```
  2. If $P_l(C)$ or $P_r(C)$ is nondistinct, make duplicate vertices and redistribute the
     edges. See Fig. 19(b).
  3. Let $G_d$ be the resulting graph. The vertices of $P_l(C)$ and $P_r(C)$ now have
     monotonic property.
  4. FOR $v \in P_l(c) \cup P_r(C)$ DO
     4.1. $v_{\min}$ = min index of incident edges in $C$.
     4.2. $v_{\max}$ = max index of incident edges in $C$.
     END FOR.
  5. WHILE exist $(a,b) \in C$ which is an external edge DO
     5.1. Let $a \in G_l, b \in G_r$.
     5.2. Let $(a,d) \in G_l$ and $(b,c) \in G_r$ where $(a,d)$ and $(b,c)$ are external edges.
     5.3. IF $[c_{\min}, c_{\max}] \cap [d_{\min}, d_{\max}] = \phi$ THEN
          5.3.1 IF $c_{\min} > d_{\max}$ THEN delete the external edge $(b,c)$.
          5.3.2 IF $d_{\min} > c_{\max}$ THEN delete the external edge $(a,d)$.
     5.4. ELSE
          5.4.1. IF there are vertices in the cycle $(a,b,c,d)$ THEN
                 5.4.1.1. Report $MC_4(a,b,c,d)$.
                 5.4.1.2. Delete edges and vertices inside $MC_4(a,b,c,d)$.
     END WHILE.
END.
```

PROCEDURE **FIND__MC4__CASE__2**(G_l, G_r, C)

```
INPUT: A cutset $C$ and left and right subgraphs $G_l, G_r$.
OUTPUT: All $MC_4(a,b,c,d)$ where $a,b,d \in G_l$ and $c \in G_r$.
BEGIN
  1. Delete all edges $(u,v)$ where both $u,v \notin P_l(C) \cup P_l(C)$.
  2. Delete all edges in $G_r$.
  3. Delete all vertices with degree less than 2.
  4. Delete all vertices not incident to infinite face.
  5. Let $G_d$ be the resulting graph.
  6. FOR $d \in (G_d \cap G_l) - P_l(C)$ DO
     6.1. FOR $v_i$ incident to $d$ DO
          6.1.2. IF $(d, v_i)$ is not incident to the infinite face THEN
                 Duplicate vertices $v_{i1}, v_{i2}$, edges $(d, v_{i1}), (d, v_{i2})$ as shown in
                 Fig. 20.
          END FOR.
     END FOR.
  7. FOR $b \in G_d \cap G_r \cap P_r(C)$ DO
     7.1. If external edges $(a,b)$ and $(b,c)$ exist THEN
        7.1.1. Let $(b,d_1)$ and $(c,d_2)$ be the external edges where $d_1, d_2 \in (G_d \cap$
               $G_l) - P_l(C)$.
        7.1.2. IF $d_1 = d_2$ and there are vertices in the cycle $(a,b,c,d)$ THEN
             Report $MC_4(a,b,c,d_1)$.
    END FOR.
END.
```

At each recursive step, the algorithm **FIND__MC4()** visits the edges of G at most a constant number of times. Although step 6 of **FIND__MC4__CASE__2()** contains two nested loops, the time complexity remains linear because each edge (d, v_i) is visited a constant number of times in **6.1.2.** Therefore, the time complexity of the algorithm is $O(n \log n)$.

REFERENCES

[1] K. Kozminski and E. Kinnen, *Rectangular dual of planar graphs*, Networks, 15 (1985), pp. 145–157.

[2] ———, *Rectangular dualization and rectangular dissection*, IEEE Trans. Circuits Systems, 35 (1988), pp. 1401–1416.

[3] Y. T. Lai and S. M. Leinwand, *Algorithms for floorplan design via rectangular dualization*, IEEE Trans. Comput. Aided Design, 7 (1988), pp. 1278–1289.

[4] J. Bhasker and S. Sahni, *A linear algorithm to find a rectangular dual of a planar triangulated graph*, Algorithmica, 3 (1988), pp. 247–278.

[5] Y. Sun and M. Sarrafzadeh, *Floorplanning by graph dualization: L-shaped models*, in Proc. IEEE Internat. Symposium on Circuits and Systems, New Orleans, LA, 1990, pp. 2845–2848; Algorithmica, 10 (1993), pp. 429–456.

[6] K. H. Yeap and M. Sarrafzadeh, 2-*concave rectilinear polygons—necessary and sufficient for graph dualization*, in Proc. 29th Annual Allerton Conference on Communication, Control, and Computing, Urbana, IL, 1991, University of Illinois at Urbana-Champaign.

[7] S. Tsukiyama, K. Tani, and T. Maruyama, *A condition for a maximal planar graph to have a unique rectangular dual and its application to VLSI floorplan*, in Proc. IEEE Internat. Symposium on Circuits and Systems, Portland, OR, 1989, pp. 931–934.

[8] S. Tsukiyama, K. Koike, and I. Shirakawa, *An algorithm to eliminate all complex triangles in a maximal planar graph for use in VLSI floorplan*, in Proc. IEEE Internat. Symposium on Circuits and Systems, San Jose, CA, 1986, pp. 321–324.

[9] R. Otten, *Efficient floorplan optimization*, in Proc. Internat. Conference on Computer-Aided Design, Santa Clara, CA, 1984, pp. 499–502.

[10] L. Stockmeyer, *Optimal orientation of cells in slicing floorplan designs*, Inform. Control, 57 (1983), pp. 91–101.

[11] F. Preparata and M. Shamos, *Computational Geometry—An Introduction*, Springer-Verlag, 1985.