VLSI Design
1994, Vol. 1, No. 2, pp. 99–111
Reprints available directly from the publisher
Photocopying permitted by license only

Building Rectangular Floorplans—A Graph Theoretical Approach

MARWAN A. JABRI
Systems Engineering and Design Automation Laboratory, Sydney University Electrical Engineering, New South Wales 2006, Australia

(Received November 22, 1988, Revised March 25, 1990)

Rectangular dualisation is a technique used to generate rectangular topologies for use in top-down floorplanning of integrated circuits. In order for this technique to be used in a floorplanning system, its input, the connectivity graph representing an integrated circuit has to fulfill a number of conditions. This paper presents an efficient algorithm that transforms an arbitrary connected graph, representing an integrated circuit, into another graph that is guaranteed to fulfill these conditions and to admit rectangular duals. Effectively, the algorithm solves the global routing problem by using three techniques: passthrough, wiring blocks and collapsed wiring blocks. Resulting floorplans may be passed to a chip assembler and detailed router package to complete the layout. This paper also introduces a novel technique to transform a tree of biconnected sub-graphs into a block neighbourhood graph that is a path.

Key Words: *Integrated Circuits, Rectangular Dualisation, Algorithms*

The design process of an IC starts by decomposing the function to be implemented on silicon into a *functional block diagram* (FBD, see Figure 1(a)). This FBD has a "blocks and buses" structure where blocks represent sub-functions and buses represent the interconnections that carry data and other information between blocks. The decomposition of the function into sub-functions is hierarchical and aims at reducing the complexity of the design problem at any one hierarchical level. When the FBD is known, the floorplanning process may be carried out. When this task is performed manually, the designer searches for a relative placement of the blocks and for an area and shape for each block which minimises the overall chip layout area while at the same time meeting design constraints such as layout design tool limitations, interconnection type and technological design rules. (see Figure 1(b)). Thus, the problem of floorplanning a chip, represented by a single rectangle at the top level of the hierarchy, corresponds to partitioning of the enclosing rectangles at each level of the hierarchy into subrectangles that enclose subblocks. Adjacent rectangles then represent communicating blocks. Blocks which do communicate but are not adjacent must have their communications carried by intermediate blocks (indirect communication). The ability of a block to carry foreign signals depends on several factors such as its functionality and the availability of design tools to add the signal wires. The criteria for choosing between direct and indirect schemes for communication routing are based on specific knowledge of the particular communication requirements.

An important aspect of the floorplanning process should be noted when the design is approached top-down, a strategy we encourage. In such cases the designer has incomplete knowledge of the exact content of each block in the floorplan and therefore has to proceed in terms of rectangle shapes and area according to his or her experience.

It is important to note here the difference between top-down floorplanning and block placement which is used in bottom-up design approaches. The basic difference resides in that the former assumes that information such as positions of block communication ports, exact block dimensions or shapes need not be known, while the latter is considered as a problem of optimally placing arbitrarily shaped blocks with defined port positions and dimensions. This paper is concerned with the automation of top-

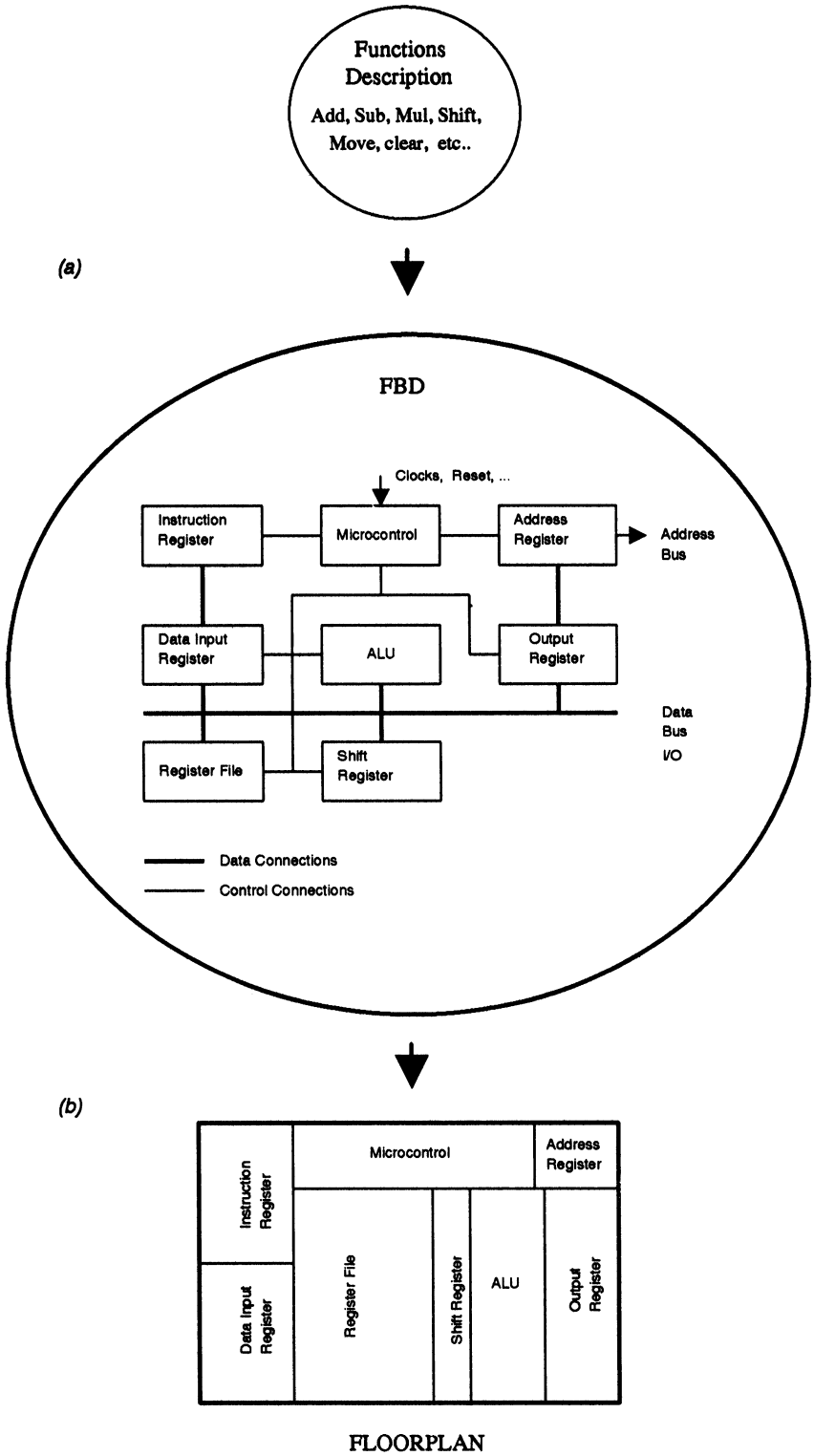

FIGURE 1 (a) Decomposition of functions into an FBD. (b) The building of a rectangular floorplan from the FBD.

down floorplanning which is considered much more difficult as more variable dimensions are included.

The automation of the floorplanning process has been the subject of intense research and powerful algorithms have been produced in recent years. The most notable are: min-cut partitioning [1, 2, 3], force-directed placement [4], simulated annealing based placement optimisation [5, 6], graph clustering [7], rectangular topology generation [8, 9] and the building of graphs for rectangular dualisation [10]. Note that min-cut, force-directed and simulated annealing algorithms were developed originally for bottom-up placement but may be used in some top-down floorplanning tasks.

Top-down floorplanning algorithms use very little information on circuit design. Moreover, algorithmic approaches do not deal with the whole problem in a complete way in as much as steps which cannot be solved with algorithms are left to the designer. The limitations of algorithmic approaches have led to the consideration of Knowledge-Based Systems (KBS) programming techniques. Recently several KBS approaches have been reported. Although still experimental, results have started to emerge [11, 12].

PIAF is a knowledge-based/algorithmic floorplanning system (KBS) developed at the Systems Engineering and Design Automation Laboratory over the last few years [11, 13, 14]. It relies on a strategy that partitions the floorplanning task in a way that allows efficient use of heuristics and specialised design knowledge in the generation and pruning of the solution space.

In this paper, we discuss one of the key algorithms of PIAF, namely the automatic generation of graphs that admit rectangular duals while at the same time solving the global routing problem. The importance of this algorithm resides in that it takes as input any connected weighted graph and produces another graph, that we call a *rectangular admissible connectivity graph* (RACG), that can be rectangularly dualised. The algorithm has already been reported [10], but we present it here in more detail and with some extensions. It has been implemented as part of a package called PAF (Package for Algorithmic Floorplanning) which is used by PIAF. As shown later in Section 5, the algorithm is very fast and suitable for early interactive exploration of the floorplanning space.

The advantages of this algorithm over other approaches are:

1. It is not limited to slicing structures as opposed to the min-cut technique [1, 3],
2. It does not require the user to specify the positions of the ports (pins) around the cells as opposed to a simulated-annealing approach [6],
3. It is much faster as an explorative tool than other techniques namely simulated annealing, force-directed and min-cut [1, 2, 3, 4, 6, 9].

The paper is organised as follows. Section 3 reviews some basic graph terminology and rectangular dualisation and the conditions for a graph to admit a rectangular dual. The RACG building algorithm is then presented in Section 4. In Section 5, we evaluate the algorithm and its performance with some examples. Finally, Section 6 draws some conclusions and directions for future work.

GRAPHS AND RECTANGULAR DUALISATION

This section presents the graph notation and terminology used throughout this paper. It also presents some important definitions and theorems in graph theory. The definitions and terminology follow those used by Tarjan [15].

Graphs

Except indicated otherwise, by a graph we mean an *undirected graph*. An undirected graph G(V, E) consists of a set of vertices (nodes) V and a set of edges (arcs) E. Two nodes connected by an edge are said to be adjacent. The number of edges attached to a the node is known as *node degree*.

Graphs are commonly used to represent integrated circuit structural description (see Figure 2).

Paths

A *path* in G is a sequence of vertices v_i and edges e_i ($i \leq N$), where $e_i = (v_i, v_{i+1})$. A path between nodes v and w is denoted by: $v \Rightarrow^* w$. A *simple path* is a path where each node has been traversed once. A *cycle* is a simple path where the extremity nodes are identical. The number of edges in a cycle is known as *cycle length*.

A spanning of the graph G' by the procedure *pathfinder* presented by Tarjan [15] produces a set of paths of the form $s \Rightarrow^* f$. The first is a cycle and the others are simple. Each path (except the first) has in common with the last generated path exactly two vertices s and f.

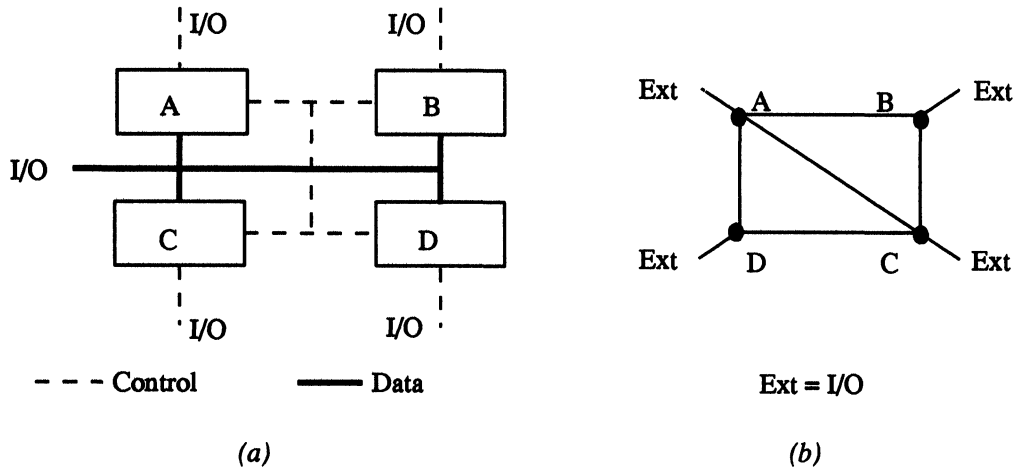

FIGURE 2 (a) A connectivity graph in the form of "boxes and buses." A, B, C and D represent sub-circuits or blocks. (b) An adjacency graph derived from (a). The existence of an edge between two blocks means that these blocks are adjacent and share a side.

Faces

When a planar graph is drawn on a plane, a face is defined by the inner area enclosed by a cycle. In this drawing, any cycle that encloses another cycle may not be a face. The number of edges in a face is known as *face degree*.

Connectivity

A graph is connected if for any pair of vertices there is at least one path joining them.

Biconnectivity, Articulation Points

Let G(V, E) be an undirected graph. Suppose that for each triplet of distinct vertices v, w, a in V, there is a path p: v $\Rightarrow^*$ w such that a is not on the path p. Then G is biconnected. If, on the other hand, there is a triplet of distinct vertices, v, w, a in V such that a is on any path p: v $\Rightarrow^*$ w, and there exists at least one such path, then a is called an articulation point (cut vertex) of G.

Planarity

A graph is planar if it can be drawn on a plane with no edges crossing.

Short-Cuts, Corner-Implying Paths, Block Neighbourhood Graphs

Consider a planar graph G(V, E).

Definition 1 *A block is a biconnected component. A plane block is a planar block. A shortcut in a plane block G, is an edge that is incident to two vertices on the outermost cycle of G that is not part of this cycle. A Corner-Implying Path (CIP) in a plane block G, is a segment* $v_1, v_2, \ldots, v_k$ *of the outermost cycle of G with the property that* (v_1, v_k) *is a shortcut and that* $v_2, \ldots, v_{k-1}$ *are not endpoints of any shortcut. The Block Neighbourhood Graph (BNG) of a planar graph G, is a graph in which vertices represent the biconnected components of G, and where an edge between two vertices in the BNG exists if the corresponding biconnected components have a vertex in common. A Critical Corner Implying Path (CCIP) in a biconnected component* G_i *of G is a CIP of* G_i *that does not contain cut vertices of G.*

Rectangular Duals of a Graph

Definition 2 *The duality relationship between a graph G and a rectangular topology is defined by the following:*

1. *A node in G is a rectangle in the rectangular dual (RD).*
2. *The infinity node (representing the I/O) in G corresponds to the region enclosing the rectangle containing the RD.*
3. *Two nodes joined by an edge in G are two adjacent rectangles in the RD.*
4. *An edge between a node and the infinity node represents a rectangle on the periphery of the RD.*

Figure 3 shows two rectangular duals of the graph of Figure 2(a).

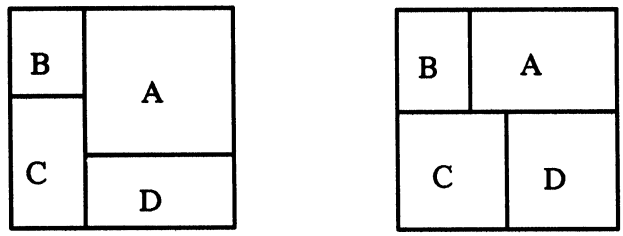

FIGURE 3 These two floorplans represent two distinct rectangular topologies. Note the variations in the block surrounds.

Theorem 1 [17] *A planar graph G admits a rectangular dual if:*

1. *Every face, except the external, has a degree of 3 (triangular).*
2. *All internal nodes have degree ≥ 4.*
3. *All cycles that are not faces have length ≥ 4.*
4. *One of the following is true:*
   (a) *G is biconnected and has no more than four CIPs.*
   (b) *G has n, n ≥ 2, biconnected components; the BNG of G is a path; the biconnected components that correspond to the ends of this path have at most two CCIPs; and no other biconnected component contains CCIP.*

Figure 4 shows examples of graphs that contradict Theorem 1.

In the last few years, several researchers have reported techniques to tackle subproblems of the RACG generation process (see for example [18]) but none has produced an algorithm that generates RACGs from *connected* graphs, as the algorithm we are reporting here does.

As mentioned earlier, the properties of an RACG are formally presented by Kozminski and Kinen [17], and are summarised in Section 3.8.

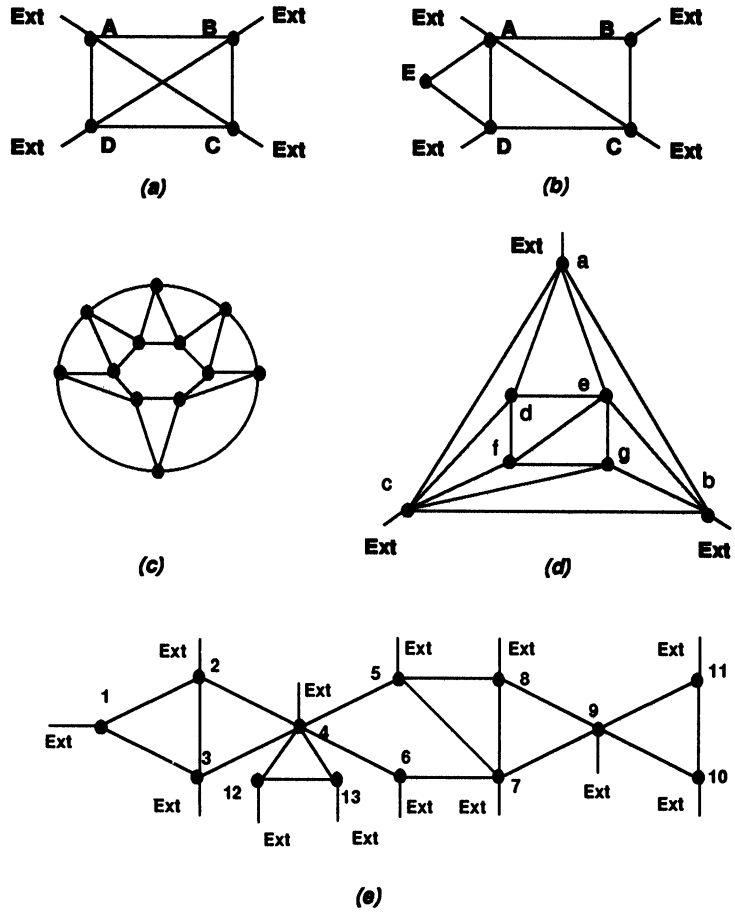

FIGURE 4 (a) A nonplanar graph. (b) Node E has a degree shortfall. (c) The face at the centre of the ring has a degree larger than 3. (d) The cycle of length 3, (a, b, c, a) is not a face. (e) This graph has a BNG that is a unique path. The 3 biconnected components (where Ext connection is not included) are connected at node 4.

The following conjecture makes the automatic generation of condition 4 for Theorem 1 considerably more efficient.

Conjecture 1 *Consider a planar connected graph G(V, E) embedded on the plane, and consider an additional node present in the external face of the embedding and connected to each node at the periphery of the embedding thus defining a set of faces F. The BNG of G is a path iff F does not contain a face of degree > 3.*

Thus, to force a BNG path for a planar embedding we ensure that the faces in F are triangular.

To illustrate the conjecture, we apply it here to the graph of Figure 5(a). This graph has three biconnected components (1, 2, 4), (3, 4, 5) and (4, 6, 7) which do not form a path. To force a path, the application of Conjecture 1 requires the introduction of a node O (see Figure 5(b)) and the "destruction" of face (O, 6, 4, 6) into two faces of degree 3 (see Figure 5(c)). As edge (O-4) already exists, the only possibility would be to create the defacto edge (5-6). The new graph will have only two biconnected components, (1, 2, 4) and (3, 4, 5, 6, 7), which correspond to a path.

THE RACG BUILDING ALGORITHM

The Algorithm's Input

Consider a connected graph G(V, E) that represents the FBD of a circuit. The user can select the preferred set of blocks to occupy corners in the rectangular topology. However, corner selection has the lowest priority in the process, as in a top-down design strategy block communication port locations are not yet defined. The user can also select weights for the edges. These weights are used in a cost evaluation function in cases of competing solutions and to determine a critical path to be preserved during the planar embedding process.

In order to represent the external connections (I/O pads), an additional node is introduced. This node will be referred to as node 1 in this paper. Note that the insertion of node 1 as the external node is done with the condition that there is at least one biconnected component of the circuit graph that has more than two nodes connected to the exterior. If this condition is not met, then there is no need to introduce the extra external node. For the simplicity of the presentation, the remainder of this paper considers that the condition holds, as the opposite case is more simple and is treated similarly.

The Algorithm

The algorithm has eight phases:

1. Forcing a planar embedding
2. Merging biconnected components
3. Restoration of deleted edges
4. External defacto connections
5. Solving node degree shortfall
6. Forcing a BNG path
7. Destruction of illegal cycles
8. Selection of corner blocks

In the first phase, a planar embedding of the connectivity graph is found. In this process, edges might be removed, and as a result, the graph might be

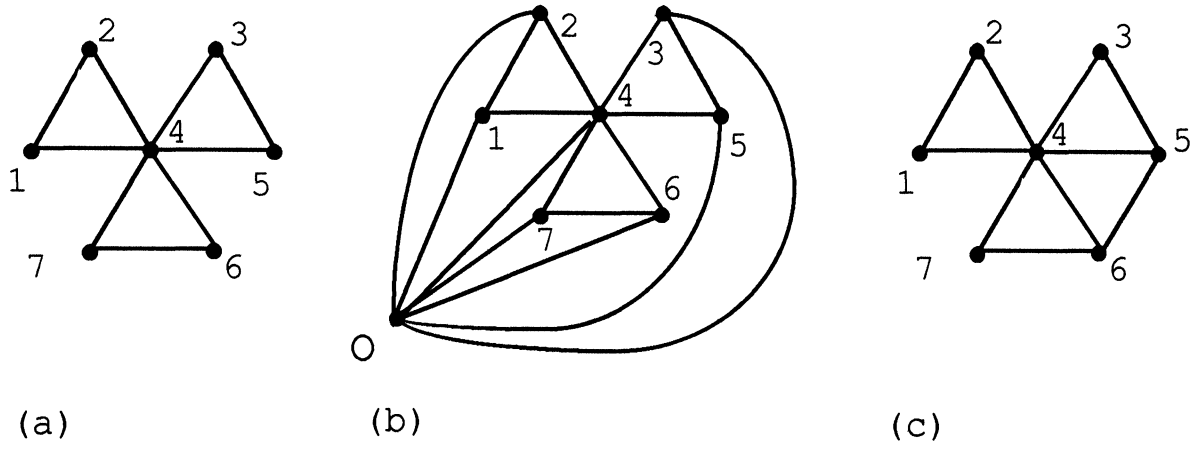

FIGURE 5 Embedding biconnected components into a path.

broken in a number of biconnected components. After that a planar embedding is found, the biconnected components are drawn on the plane and therefore all faces are determined. These faces represent regions of the planar graphs enclosed by cycles and not containing any nodes.

The second phase assembles the biconnected components if there is more than one. This is necessary as biconnected components may be considered as floating sub-graphs and it is required to *fix* them to positions in the plane.

In the third phase, edges which may have been removed during planarisation (phase 1) are restored by the means of passthrough (connections are routed through existing nodes and the path is selected with an objective function based on the weights of the graph nodes), wiring blocks (a node representing a wiring block is inserted to permit edge crossing without violating graph planarity) or collapsed wiring blocks (if more than one edge is being crossed by the edge being restored, then the nodes inserted as wiring blocks are collapsed into a single one).

In the fourth phase, edges are established between nodes present on the periphery (external cycle) and the node representing the exterior (i/o pins). These are *defacto* connections as these cells have been *forced* to reside on the periphery.

In the fifth phase, the minimum degree of nodes is satisfied by introducing more *defacto* edges. This is due to the fact that we are dealing with rectangular shapes and a rectangle needs at least four other adjacent rectangles (except for rectangle on the periphery where the minimum requirements are three for a non-corner block and two for a corner block respectively).

In the sixth phase, we force the graph to have a BNG that is a path. To do this, the algorithm uses Conjecture 1 which reduces the task to destroying the external faces (made of edges between the external node and the nodes on the periphery of the graph) that have more than three nodes by inserting more *defacto* edges.

In the seventh phase, the algorithm destroys illegal faces in the graph (non-external faces) that have more than three nodes and illegal cycles (cycles with three nodes that are not faces).

Finally in the eighth phase, blocks to occupy the four corners of the floorplan are selected. Note that if the user has already specified some blocks to be corner-blocks, then the algorithm will maximise the number of these blocks in the selection process. The output of the algorithm is the final adjacency lists of the graph, and restoration paths for any edges that have been removed from the original connectivity graph, the list of corners and faces. Rectangular dualisation may then be applied to produce the floorplans.

We present in the following paragraphs each of these phases in detail.

Phase 1: Forcing a Planar Embedding

In this phase, the search for a planar embedding of the input graph is carried out by a recursive procedure derived from [16]. This procedure planarises the graph by deleting a minimal set of edges if needed (see Figure 6). As the set of edges is not unique, two types of control upon edge selection are used. The first is done via the initial ordering of the adjacency lists that represent the graph (first path in Figure 6). The second type of control is done via the decision of which edge to delete in a crossing pair. An edge weighting function is used to select the appropriate one. Each time an edge is deleted, we restart the procedure on the biconnected components of the new graph. This procedure ends when the graph is planarised, producing a list of planar biconnected components, a list of deleted edges and a list of the generated paths as if they were produced by *pathfinder* [15].

Following the planar embedding of the connectivity graph, the algorithm proceeds in this phase to draw the planar biconnected components on the plane and to generated the faces of the embedding. As several drawings may exist, the current procedure is exhaustive in its search. This permits backtracking to generate different RACG solutions. The drawing

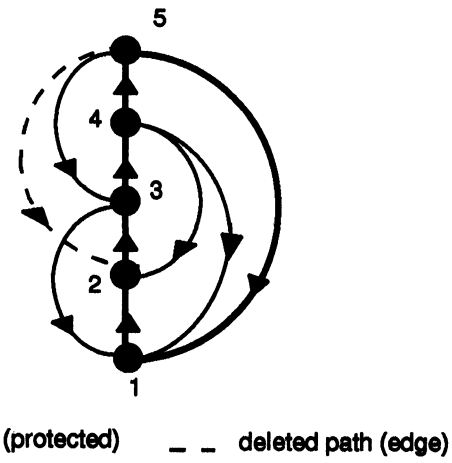

FIGURE 6 Planar embedding of graphs. The first path (1, 2, 3, 4, 5, 1) is protected. For the embedding in the figure, removal of edge 5-2 will planarise the graph. An alternative solution is the removal of edge 3-1.

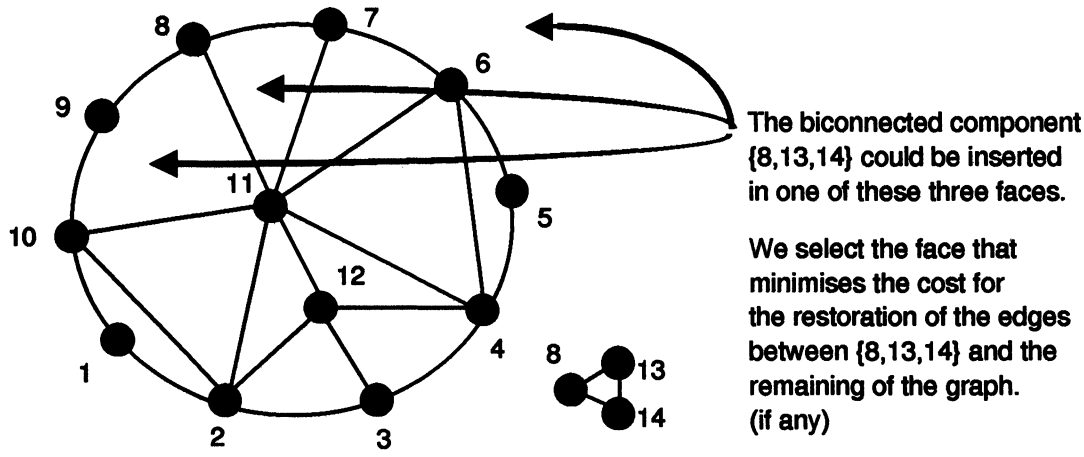

FIGURE 7 Biconnected components merging.

procedure outputs the external face and the set of internal faces that define the planar embedding for each biconnected component.

Phase 2: Merging the Biconnected Components

In this phase, the planar biconnected components are merged together. The algorithm carries out this step as follows. A biconnected component host is selected. If node 1 is the external node then the biconnected component containing it is selected. Then for each of the remaining components we find all the faces in the host that contain the articulation node (node 8 in Figure 7). We select as a host the face that maximises the number of deleted edges between its nodes and the biconnected guest, where we insert it. If several faces satisfy this selection criteria, then we look for the face (in the path) that minimises the cost of restoring the most expensive deleted edge (see Phase 3 for more details on edge restoration). The external face of the guest is combined with the hosting face to produce a set of new faces. The hosting face is replaced by the set. In addition, a defacto edge is inserted to make the graph resulting from the merging a single block.

Phase 3: Restoration of Deleted Edges

This phase restores deleted edges using one of three options. In the first, the algorithm gives the user the possibility of restoring the deleted edges using techniques other than the insertion of wiring blocks. In the second option, the deleted edges are restored by the insertion of wiring blocks each time a crossover occurs. In the third option, adjacent crossovers are grouped into one wiring block. The deleted edges restoration algorithm uses a "branch and bound" search to determine an appropriate restoration path. For each edge to be restored, the restoration algorithm finds the "faces path" that separates the endpoints of the edge (see Figure 8). A "face path" is a set of graph faces which need to be crossed in order to join two nodes. The path selection is based upon the least edge weight separating two adjacent faces. Then, the edge is restored with the injection of wiring blocks at its crossing with the edges in the faces path. The faces of the embedding are updated by this operation.

Phase 4: External Defacto Connections

The faces generated in the previous phase and containing node 1 as external node are checked for their degree. If such a face has a degree greater than 3 and a set of its nodes not connected to the external node, then the nodes in this set will have a forced (defacto) connection to the exterior. This face will be destroyed (generating new faces) by the introduction of new edges between these nodes and node 1.

Phase 5: Node Shortfall Solving

In this phase we solve the shortfall of node degree (node degree < 3). The algorithm proceeds as follows. First, it tries to increment the degree of a node by injecting a defacto edge if the node belongs to a face with degree > 3. The node with minimal degree in the face is selected for the defacto edge. If such a face does not exist, then a neighbouring face of a

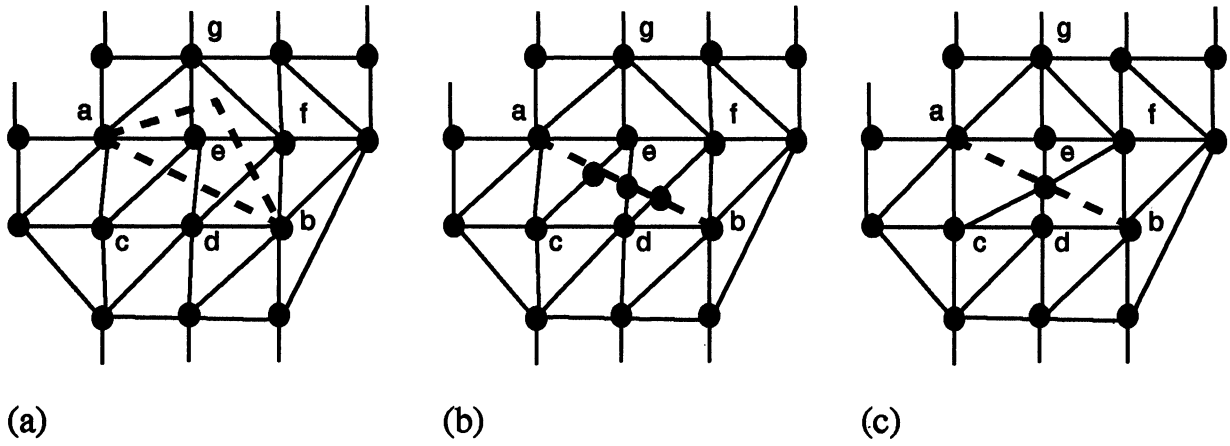

FIGURE 8 (a): Two different faces paths to restore the edge (a-b): (a, e, c), (e, c, d), (e, d, f), (f, d, b) and (a, g, e), (g, e, f), (e, d, f), (f, d, b). (b): The first "face path" is selected and wiring blocks inserted on edge crossings (option 2). (c): Only one wiring block is used (Option 3).

face that the node belongs to is found using least cost branch and bound search. Then the node with least connection in that face is selected as the end node of an edge that the algorithm installs using wiring blocks.

Phase 6: Forcing a BNG Path

The biconnected components generated in phase 1 have the property that node 1 (when it is the exterior node) could not be an articulation node for any pair of biconnected components. This is the result of the DFS operation and the condition for the insertion of node 1 as an external node as described above. This result is used in the following manner. First consider B_e to be the biconnected component including the external node, and let P_b be the BNG of the node set (B_e − *external node*). If P_b is a path then no further action is taken. If P_b is not a path then we search in B_e for all faces that have a degree ≥ 4 and we destroy them by creating a defacto adjacency. The destruction of the faces will ensure that P_b becomes a path. According to the BNG, a list of valid user corners is built for later use. The criteria of validity at this stage concerns only the structure of the BNG. Each user corner that is an articulation node, or that is not in the extremities blocks of a BNG path (with path length > 1), is considered invalid.

Phase 7: Illegal Face and Cycle Destruction

As stated in the RACG conditions earlier, faces not adjacent to the periphery should have a degree of 3 and thus faces with degree ≥ 4 have to be destroyed. To do this, we find in the faces that do not contain node 1 (the exterior node) the nodes that have a shortfall and we establish defacto edges between them. This is applied until no more faces are illegal. In addition, cycles that have a length ≤ 3 and that are not faces have also to be destroyed. An example of such a cycle would be enclosing faces. To destroy these cycles, the algorithm uses first a DFS algorithm to find them. Then a wiring block is inserted in the weakest edge of each cycle (see Figure 9). Two additional connections are added to the wiring block, one on each of the sides defined by the cycle. If the edge is on the periphery, then one of these two connections would be with the exterior. The illegal cycle

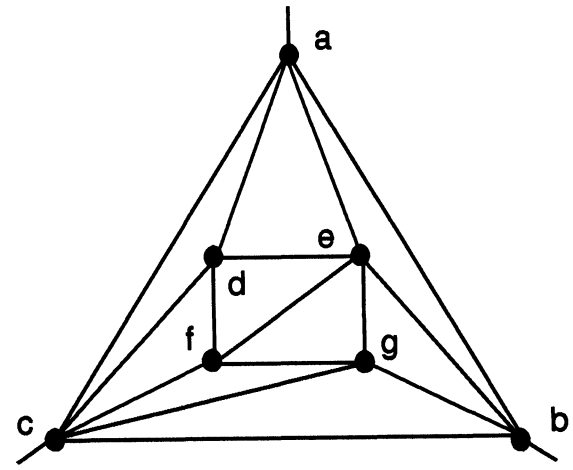

FIGURE 9 An illegal cycle of length 3: (a-b-c). To destroy this cycle is the same as restoring a zero weight edge between one of (d, e, f, g) and the external face and injecting a wiring block at the crossing with one of the illegal cycle edges.

removal is performed after the illegal faces destruction. This is to avoid the creation of unwanted cycles by the insertion of defacto edges when destroying faces.

Phase 8: Selecting the Corners

In this phase, we find first the shortcuts and the CIPs (or CCIPs, depending on the BNG). The algorithm then checks if any of the valid user corners belong to the CIPs, and the corresponding CIPs are removed. If the user selected a node to occupy two corners, then only one is searched for in a CIP. If the graph's BNG is a path of length > 1, then only the corner nodes at the path extremities are considered, and as stated previously, the other corners would have been revoked in the BNG path forcing phase. After that all CIPs (or CCIPs) are satisfied, if any shortfall of corners still exists then valid double corners are considered before the consideration of nodes from the external faces to adjust the number of corners to 4 (or 2 for each extremity when dealing with CCIPs). At the end of this phase, the graph will be an RACG and ready for dualisation.

ALGORITHM EVALUATION

As mentioned previously, the algorithm was implemented as part of a library of algorithms used by PIAF, a knowledge-based/algorithmic floorplanning system. It is written in Pascal and runs on Sun Workstations. The RACG produced by the algorithm is dualised by an algorithm based on [8] and developed by the author. The input to the algorithm (the edge weights) can be used to influence the solutions according to design needs. Experiments with the algorithm on real graphs (FBDs) have shown that the number of wiring blocks injected for communication restoration is slightly larger than in the case where wiring blocks are only injected for node shortfall solving and/or illegal cycle destruction. Another approach to block insertion would consider the use of previously inserted blocks in the routing. The algorithm could easily be modified to do so.

We were unable to obtain benchmarks from MCNC as none are available for the floorplanning approach we are using.

Note that the order of the phases of the algorithm can to some extent be altered. The current ordering was intuitively devised and has produced the best results on our test data.

An Example

The graph of the example is shown in Figure 10 and corresponds to the FBD of a LISP chip. It contains 18 nodes. The corresponding rectangular dual of the algorithm's RACG is shown in Figure 11. The algorithm has added 18 wiring blocks in order to produce the RACG (they are named "paf_wb_N", where N is a number).

Run Time Examples

Table 1 shows the run time in seconds for the example presented above and several other non planar graphs.

When edges restoration is disabled, the algorithm has shown an expected peak run time proportional to N*M, where N and M are the number of nodes and edges of the input graph respectively. When enabled, local optimisation ("Branch and Bound") processing for edges restoration makes the definition of a timing model difficult.

Branch-and-Bound Control

The branch-and-bound search algorithm used here has a parameter that controls the number of alternative solutions that it can produces. The introduction of this parameter appeared necessary to avoid the problem of memory limitations encountered with very large graphs and to give the user some control over the output graph size. Of course, if the number of tries is set to a small value in the case of large input graph, then the graph size may grow much larger. As an example, if the maximal number of tries for the branch-and-bound is set to 5, then the number of wiring blocks introduced in the case of the LISP chip increases from 18 to 30.

Improvements

There are a number of possible improvements that can speed up the algorithm further and reduce the size (number of wiring blocks) of the final RACG.

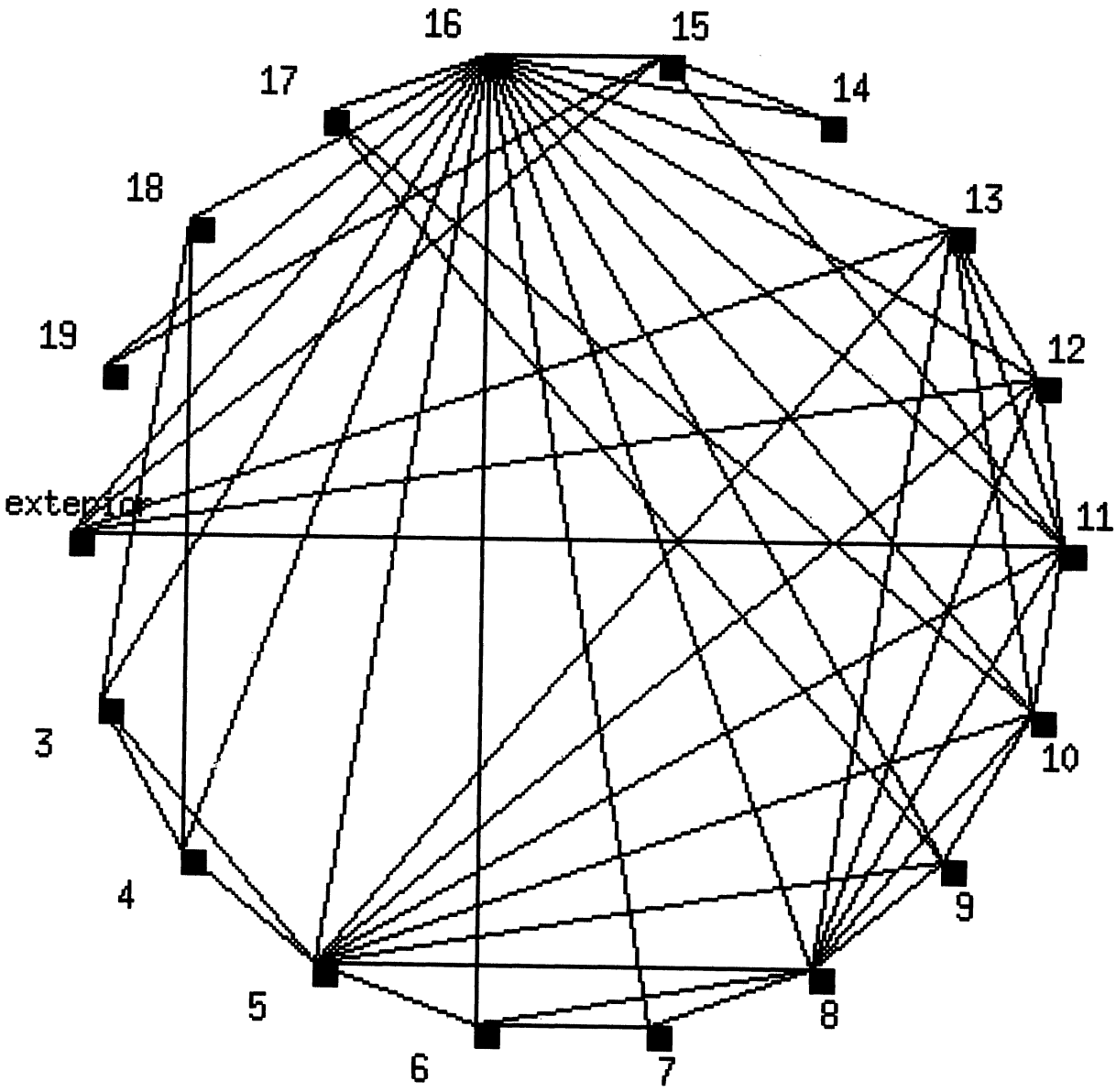

FIGURE 10 The input graph of the example, corresponding to a LISP chip.

These improvements include:

- Recent work on graph planarisation [19], for example can be exploited to speed up Phase 1.
- If the selection of corners specified by the user is critical, short cuts of the external cycle can be eliminated, and the BNG can be reduced to a single block.

CONCLUSION

We have presented an algorithm for transforming a connectivity graph representing an integrated circuit into another graph that admits rectangular duals. The algorithm solves the global routing problem by using three techniques: passthrough, wiring blocks and collapsed wiring blocks. Resulting floorplans may be passed to a chip assembler and detailed router package to complete the layout. The algorithm offers three options for communication crossover solving, wiring blocks, grouped wiring blocks or passthrough. The grouping option in particular prevents fast increases in the number of wiring blocks in the circuit. We have also presented a novel procedure to transform a tree of biconnected sub-graphs into a block neighbourhood graph that is a path. The algorithm

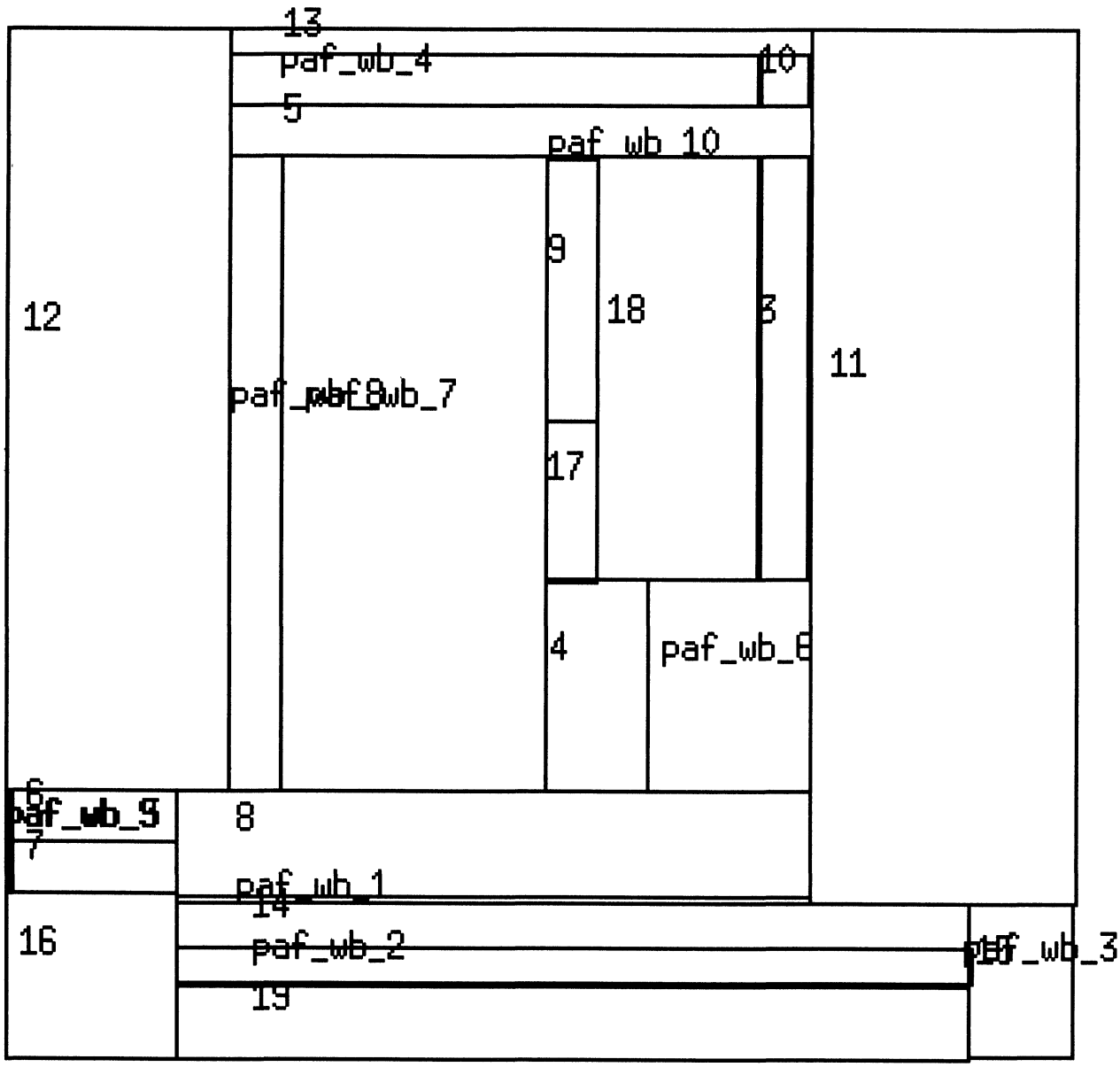

FIGURE 11 A rectangular dual of the RACG generated for the graph shown in Figure 10.

TABLE I
Approximate CPU time for some graph examples (in seconds, on a Sun 4/370)

Graph	Number of Nodes	Wiring Blocks Added	Branch and Bound Tries	RACG Building CPU Time
LISP chip	18	18	20	15
Encryption chip	13	2	5	0.9
Read-Solomon chip	10	3	5	0.5
K_5	6	3	5	0.5
$K_{3,3}$	7	2	5	0.4

is efficient and is very well suited for interactive applications.

References

[1] M.A. Breuer. A class of min-cut placement algorithms. In *Proc. 14th IEEE/ACM Design Automation Conference,* pages 284–290, 1977.

[2] U. Lauther. A min-cut placement algorithm for general cell assemblies based on a graph representation. In *Proc. IEEE/ACM 16th Design Automation Conference,* pages 1–10, 1979.

[3] David P. Lapotin and Stephen W. Director. Mason: A global floor-planning tool. In *Proc. ICCAD-85 Conference,* pages 143–145, Santa Clara, California, USA, 1985.

[4] K. Ueda, H. Kitazawa, and I. Harada. CHAMP: Chip floor plan for hierarchical VLSI layout design. *IEEE Trans. on Computer-Aided Design,* CAD-4(1):12–22, Jan 1985.

[5] C. Sechen and A. Sangiovanni-Vincentelli. The timberwolf placement and routing package. In *Proc. IEEE Custom Integrated Circuits Conference,* 1984.

[6] C. Sechen. Chip planning, placement, and global routing of macro/custom cell integrated circuits using simulated annealing. In *Proceedings of the ACM/IEEE 25th Design Automation Conference, Anaheim,* pages 73–80, June 1988.

[7] W. Heller, G. Sorkin, and K. Maling. The planar package planner for system designers. In *Proc. 19th IEEE/ACM Design Automation Conference,* pages 253–259, 1982.

[8] K. Maling, S. Mueller, and W. Heller. On finding the most optimal rectangular package plans. In *Proc. 19th IEEE/ACM Design Automation Conference,* pages 663–670, 1982.

[9] R. Otten. Automatic floorplan design. In *Proc. 19th IEEE/ACM Design Automation Conference,* pages 261–267, 1982.

[10] M.A. Jabri. Automatic building of graphs for rectangular dualisation. In *Proceedings of the ACM/IEEE 25th Design Automation Conference, Anaheim,* pages 638–641, June 1988.

[11] M.A. Jabri and D.J. Skellern. A mixed knowledge-based/algorithmic approach to custom integrated circuit floorplanning. In *Proc. IEEE Custom Integrated Circuits Conference,* pages 289–292, 1986.

[12] H. Watanabe and B. Ackland. FLUTE—A floorplanning agent for full custom VLSI design. In *Proc. 23rd IEEE/ACM Design Automation Conference,* pages 601–607, June 1986.

[13] M.A. Jabri and D. Skellern. PIAF—Efficient IC Floorplanning. *IEEE Expert,* 4(2):33, 45, 1989.

[14] M.A. Jabri and D.J. Skellern. PIAF: A Knowledge-Based/Algorithmic top-down floorplanning system. In *Proceeding of the 26th ACM/IEEE Design Automation Conference,* pages 582–585, Las Vegas, USA, 1989.

[15] R.E. Tarjan. An efficient planarity algorithm. Stan-cs-244-71, Computer Science Department, School of Humanities and Sciences, Stanford University, 1971.

[16] J. Hopcroft and R. Tarjan. Efficient planarity testing. *ACM,* 21(4):549–568, October 1974.

[17] K. Kozminski and E. Kinnen. An algorithm for finding a rectangular dual of a planar graph for use in area planning for VLSI integrated circuits. In *Proc. IEEE/ACM 21st Design Automation Conference,* pages 655–656, June 1984.

[18] Shuji Tsukiyama, Keiichi Koike, and Isao Shirakawa. An algorithm to eliminate all complex triangles in a maximal planar graph for use in vlsi floor-plan. In *ISCAS,* pages 321–324, 1986.

[19] R. Jakayumar, K. Thulasiraman, and Swamy M. $o(n^2)$ algorithms for graph planarization. *IEEE Transactions on Computer-Aided Design,* 8(3):257–267, March 1989.

Biography

MARWAN A. JABRI is Director of the Systems Engineering and Design Automation Laboratory, Sydney University Electrical Engineering. He has a research interests in artificial neural networks, integrated circuit design and computer aided design, formant speech synthesis, image processing, digital signal processing for speech applications.

He joined the University of Sydney in 1983 working as a research assistant with the Fleurs Radio Telescope Group where he designed and developed the Post Correlation Processor (a real-time data processing system). He has since worked on the design of ASICs for image reconstruction and multi-chennel formant speech synthesis, computer aided design for VLSI and artificial neural networks for pattern recognition and signal processing.

Marwan Jabri has authored or co-authored a book and over 30 technical papers. He was Chairman of the First Australian Conference on Neural Networks and appointed as Chairman of the forthcoming Second Australian Conference on Neural Networks (Feb. 4–6, 1991). He is Member of the Advisory Board on Design of the International Conference on the application of artificial intelligence in Engineering to be held in Boston (USA) in 1990.