VLSI DESIGN
1998, Vol. 7, No. 1, pp. 97–110
Reprints available directly from the publisher
Photocopying permitted by license only

Placement and Routing for Performance-Oriented FPGA Layout†

MICHAEL J. ALEXANDER[a,*], JAMES P. COHOON[b], JOSEPH L. GANLEY[c] and GABRIEL ROBINS[b]

[a]*School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA 99164-2752;*
[b]*Department of Computer Science, University of Virginia, Charlottesville, VA 22903-2442;*
[c]*Cadence Design Systems, Inc., San Jose, CA 95134-1937*

This paper presents a performance-oriented placement and routing tool for field-programmable gate arrays. Using recursive geometric partitioning for simultaneous placement and global routing, and a graph-based strategy for detailed routing, our tool optimizes source-sink pathlengths, channel width and total wirelength. Our results compare favorably with other FPGA layout tools, as measured by the maximum channel width required to place and route several benchmarks.

Keywords: FPGAs, placement, routing, performance-driven layout, Steiner trees, arborescences, multi-weighted graphs

1. INTRODUCTION

Field-programmable gate arrays, or FPGAs, provide a versatile and inexpensive way to implement and test VLSI designs [7, 16]. FPGAs are available in a number of styles and configurations [40]. One of the most common FPGA architectures [9, 43] consists of a matrix of user-configurable logic blocks interconnected by a set of programmable routing resources (Fig. 1). FPGA reprogrammability is achieved at the expense of performance, as there may be long signal delays through the reconfigurable routing resources [39]. To increase FPGA performance, partitioning and technology mapping have been extensively studied [11, 20, 27, 35]. However, the observation that circuit performance is impacted more by routing delays rather than by device delays [6, 26] has focused recent attention on routing [8, 15, 31, 32, 42].

This paper presents a performance-oriented FPGA Placement and Routing (FPR) tool. FPR is based on a recursive geometric strategy for simultaneous placement and global routing, followed by a graph-based detailed-routing phase. FPR heuristically minimizes both wirelength and source-sink pathlengths. Thus, FPR optimizes the number of FPGAs required to implement a given design, as well as the performance of the

*Corresponding author.
†Our benchmarks and additional related papers may be found at WWW URL http://www.cs.virginia.edu/~vlsicad/.

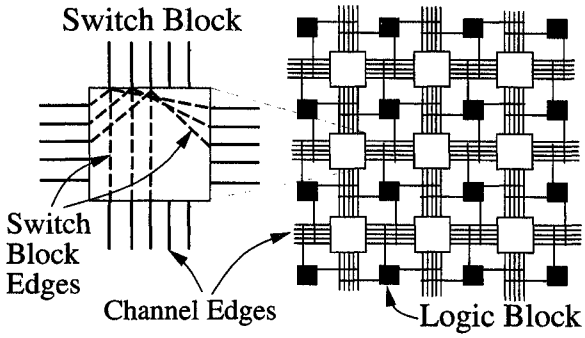


FIGURE 1 A typical FPGA architecture.

implementation. In particular, FPR successfully lays out a number of large industrial benchmark circuits using smaller channel widths than other FPGA layout tools, and also optimizes source-sink pathlengths as a secondary criterion.

The rest of the paper is organized as follows. Section 2 provides an overview of our methodology. Section 3, Section 4 and Section 5 detail the main phases of FPR, namely placement, global routing and detailed routing, respectively. Section 6 establishes the efficacy of our implementation on industrial benchmark designs, and we conclude in Section 7. The Appendix develops some theoretical results for multi-weighted graphs used in the multi-objective optimization phase of detailed routing. Preliminary versions of this work have appeared in [1, 2, 3] .

2. OVERVIEW

FPGA logic blocks typically contain a programmable look-up table, which enables arbitrary combinational-logic functions of up to four variables to be implemented. Each logic block thus contains a small portion of the overall circuit logic. The logic blocks are interconnected by channel segments, which are linked together by switch blocks. The switch blocks contain programmable internal connections among certain subsets of incident channel segments. Switch-block edges are often implemented as pass transistors, which can be "turned-on" to interconnect incident channel edges. Finally, connection edges allow logic-block pins to latch onto adjacent channel segments.

During the FPGA design process, placement and routing are performed following the technology mapping phase. Technology mapping decomposes the circuit design into units of logic, which are then assigned to specific logic blocks during placement. Thus, the input to FPR consists of unplaced logic blocks and a set of *nets* (a net is a set of logic block I/O pins that must be interconnected). FPR performs simultaneous placement and global routing using a recursive geometric technique called *thumbnail partitioning*, which decomposes the circuit area into an $m \times n$ grid, for some small fixed m and n. This grid is called the *partitioning template*. The placement is then optimized and a global routing is determined relative to the partitioning template using optimal *rectilinear Steiner arborescences* (RSAs) [34] (i.e., minimum-weight shortest path trees). Since m and n are small and fixed, these optimal RSAs (called *thumbnails*) may be precomputed for efficient lookup during execution. Setting $m=n=3$ yields the basic 3×3 partitioning template that is used in our implementation (Fig. 2(a)). Thumbnail partitioning is a generalization of *sharp partitioning* [5], which in turn is a generalization of *quadrisection* [38].

Our strategy consists of placement and global routing, followed by detailed routing. During placement and global routing, a partitioning heuristic is used to assign the logic blocks to regions in the partitioning template, minimizing

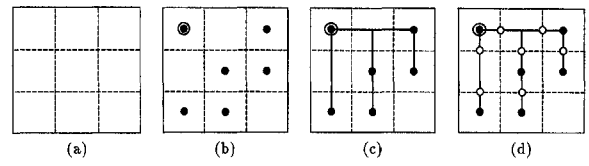


FIGURE 2 (a) Partitioning template for $m=n=3$; (b) a sample pointset (the source is at the upper-left); (c) one of its possible thumbnails; and (d) the associated virtual pins.

source-sink pathlengths as well as the total length of the thumbnails. When the circuit area is divided according to the partitioning template, each logic block lies in one of the $m \times n$ regions. For each net, we construct a pointset in the $m \times n$ grid, where a point is present in a region if some logic block associated with the net lies in that region (Fig. 2(b)). A thumbnail over this pointset is then determined (Fig. 2(c)).

To reduce overall routing congestion, alternative thumbnails are selected in order to balance the number of thumbnail edges that cross each edge of the partitioning template. "Virtual" pins are then created at the intersections of thumbnails and partitioning-template edges (Fig. 2(d)), and the algorithm is then applied recursively to each subregion of the partitioning template. This scheme simultaneously produces both a placement and a global routing in which source-sink pathlengths, total wirelength, and maximum channel congestion are all heuristically minimized. The resulting placement and global routing is then used in the detailed-routing phase to produce a complete routing solution.

During the detailed-routing phase, nets are assigned specific routing resources based on global routes. By modeling the FPGA routing architecture as a graph, efficient graph-based algorithms may be used to produce detailed-routing solutions. Nets are routed one at a time; as resources are committed to nets, the corresponding edges in the underlying graph are made unavailable to subsequent nets.

The next three sections detail the main phases of FPR, namely: (1) logic-block placement and thumbnail selection for balancing congestion, (2) global routing, and (3) detailed routing.

3. PLACEMENT

The placement phase overlays the FPGA with the partitioning template and initially partitions the design logic into $m \cdot n$ regions. Cut lines of the partitioning template go through switch blocks so that each logic block lies entirely within a single region of the partitioning template. The distribution of logic blocks among regions of the partitioning template is then improved using simulated annealing [28], where a move consists of swapping two logic blocks that lie in different regions of the partitioning template. The simulated annealing objective is to minimize (1) the sum of the maximum source-sink pathlengths in the thumbnails over the nets, and (2) the total length of the thumbnails for all nets. Note that the I/O blocks on the perimeter of the FPGA are not moved during these iterative refinement steps.

Routability is a primary concern during the FPGA design process [6, 10]. An important measure of the quality of a placement and global routing is maximum *congestion*, which in our case is the number of thumbnail edges that cross any given partitioning-template edge. Thus, once logic blocks have been assigned to regions in the partitioning template, a congestion-balancing step is undertaken as follows.

A typical pointset can have many thumbnails; for example, Figure 3 illustrates a pointset and its eight thumbnails. The objective of the congestion-balancing step is to assign one of the precomputed thumbnail alternatives to each net in a manner that minimizes the maximum thumbnail congestion. This task is accomplished using the following greedy heuristic:

- Sort the nets in ascending order of the number of distinct thumbnails for each net; and

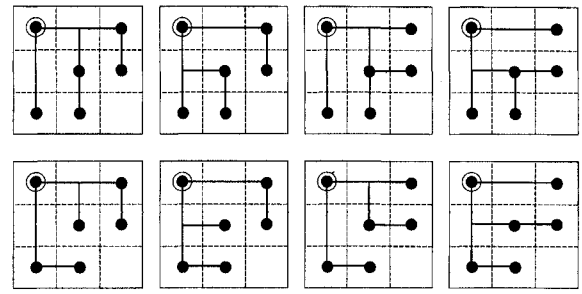


FIGURE 3 All eight thumbnails for the pointset shown in the 3×3 partitioning template (source is at upper-left).

- For each net on this list, choose the thumbnail that minimizes the maximum congestion induced by all previously processed nets.

Intuitively, this scheme postpones the global routing of nets for which there are a greater number of thumbnail choices; this enables FPR to better compensate for the less avoidable congestion incurred earlier by nets with fewer thumbnail choices.

4. GLOBAL ROUTING

After FPR has mapped the logic blocks to regions in the partitioning template and each net has been assigned a thumbnail, every edge in each thumbnail is then assigned to a specific switch block along the crossed cut-line of the partitioning template. Each such switch block is then conceptually added as a new "virtual" pin in the net. The portion of each net within each region of the partitioning template is then passed on to a lower level of the recursion (this is similar to the *virtual terminal* [5] and *terminal propagation* [14] techniques). Thus, the global routing computed for a net corresponds to the topology of its thumbnail.

Assignment of nets to switch blocks is accomplished in a manner similar to PHIroute [37]. The number of nets that can be assigned to each switch block is bounded by the number of nets crossing the cut, divided by the number of switch blocks on the cut. This construction induces a structure that may be represented by a complete bipartite graph with nets in one partition and switch blocks in the other. Edge weights in this graph model the cost of assigning a net to the corresponding switch block. Assignments are then determined by computing a minimum-cost matching [33].

Recursion terminates when a region contains at most one logic block, along with the adjacent channel segments and switch blocks. We then route nets within the channels surrounding the logic block (if it exists) while minimizing the maximum channel congestion. In our implementation, an optimal solution is computed using integer programming [30]. This is efficient in practice since the number of nets involving any single logic block is small [17].

5. DETAILED ROUTING

After placement and global-routing, FPR performs detailed routing by assigning specific channel and switch-block edges to each net. The placement and global-routing phase passes the following information to the detailed router: (1) locations of relevant logic-block pins (i.e., the net to be routed), (2) a "loose" route for the net (leaving unspecified the edges within channel segments and switch blocks), and (3) switch blocks that are likely to serve as Steiner nodes in the detailed routing (Fig. 4).

A design goal for FPR has been the ability to handle a wide variety of FPGA architectures. Towards this goal, we have adopted a graph-based approach to detailed routing. Each switch block contains internal *switch-block edges* that may be programmed to connect incoming channel edges. The routing structure of the entire FPGA is captured by a *routing graph*: detailed routes on

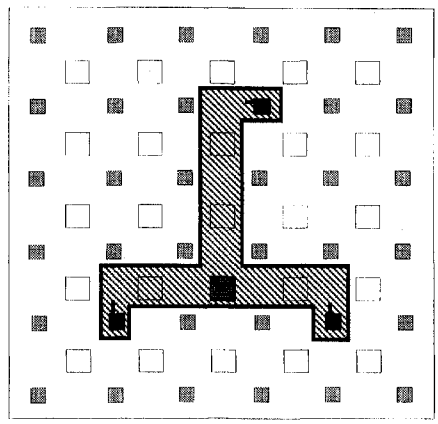
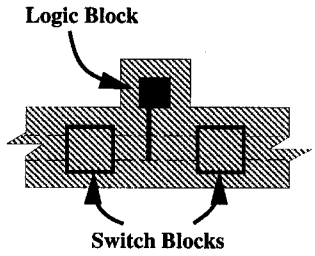


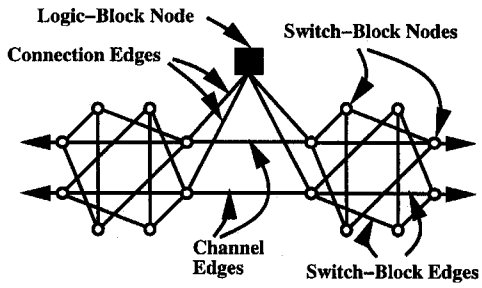
FIGURE 4 Global-routing information for a three-pin net, showing the associated logic blocks (dark squares), global route (cross-hatched region), and potential Steiner switch block (large dark square).

the FPGA correspond to paths in the routing graph, and vice-versa (Fig. 5). In a routing graph, vertices model logic-block and switch-block nodes, while the edges correspond to connection, channel, and switch-block edges. This strategy enables the detailed router to employ generic graph algorithms in order to produce detailed-routing solutions.

Using the routing-graph approach, detailed routing entails interconnecting the logic-block vertices using edges and vertices inside the corresponding global-route region. This goal is modeled by the *graph Steiner tree* (GST) problem: given graph $G=(V,E)$, where V is the vertex set and $E \subseteq V \times V$ is a set of weighted edges, find a minimum-weight tree in G that spans a subset of the vertices $N \subseteq V$ (the logic-block vertices in a net), using switch-block vertices as possible Steiner nodes. The cost of a tree T, denoted $\bar{T}$, is the sum of the costs of its edges.



(a)



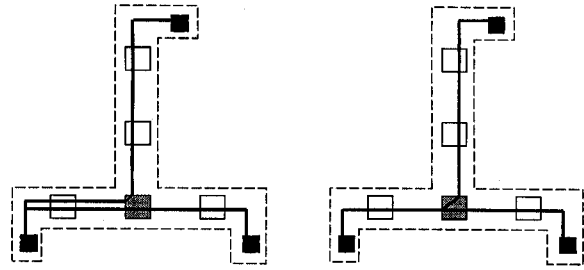
(b)

FIGURE 5 Global-routing information (a) is used to construct a routing graph (b) for a Xilinx [43] 4000-series part with channel width 2.

Since the GST problem is NP-complete [24], we utilize the heuristic of Kou, Markowsky and Berman [29] (KMB), which approximately solves the GST problem in polynomial time, and is guaranteed to yield solutions with cost less than twice the optimal. While the KMB heuristic always finds a feasible detailed routing if one exists, it often does not "branch" at the appropriate Steiner nodes (Fig. 6(a)). This potential drawback is effectively ameliorated using the greedy strategy described below.

Our detailed-routing algorithm is based on combining a greedy, iterated heuristic [21, 25] with the KMB algorithm; we refer to this hybrid method as the *Iterated-KMB* (IKMB) algorithm [1]. Given a routing graph $G=(V,E)$, a net $N \subseteq V$, and a set S of potential Steiner nodes, we define the savings of S with respect to N as $\Delta\overline{\mathrm{KMB}}_z G(N,S)=\overline{\mathrm{KMB}}_G(N)-\overline{\mathrm{KMB}}_G(N \cup S)$. Intuitively, $\Delta\overline{\mathrm{KMB}}_G(N,S)$ represents the interconnect savings incurred by KMB when the Steiner nodes in S are included into the node set N to be spanned. This is illustrated in Figure 6(b), where using a candidate Steiner node from the shaded switch block results in an optimal solution. In order to efficiently find such Steiner nodes, a set of *candidate Steiner nodes* is determined for each net. Candidate Steiner nodes are switch-block nodes that correspond to Steiner switch blocks (Fig. 4).

The IKMB method operates by repeatedly finding candidate Steiner nodes that reduce the



(a) (b)

FIGURE 6 Detailed-routing solutions; (a) a KMB solution containing unnecessary parallel paths, while (b) the IKMB solution reduces total number of channel edges by 22%.

overall KMB cost by the largest amount, and then including them into a growing set S of Steiner nodes. The cost of the KMB tree over $N \cup S$ decreases with each added node, and the construction terminates when there is no $x \in V$ with $\Delta\overline{\text{KMB}}(N \cup S, \{x\}) > 0$. The final topology is obtained by computing the KMB construction using $N \cup S$ as the pins and the remaining $V - (N \cup S)$ nodes as potential Steiner nodes. The overall IKMB method is more formally described in Figure 7.

The placement and global-routing phases seek to minimize congestion, thereby enabling the detailed router to find a feasible (and high-quality) solution more easily. However, since it is NP-complete to determine whether there exists a feasible detailed-routing solution for all nets [41], we use a deterministic net-ordering scheme to route nets one at a time. When a detailed-routing solution for a net is found, the corresponding routing resources are committed to that net and are made unavailable for subsequent nets (i.e., they are removed from the underlying graph). If infeasibility is encountered during the detailed routing of a net (i.e., some logic-block pin is unreachable in the routing graph from the other pins of the net), the following two heuristics are employed.

First, an incremental "wavefront-expansion" technique is used to gradually "loosen" the global route, allowing the detailed route to detour around local blockages caused by previously-routed nets (Fig. 8). Note that wavefront expansion determines the region searched by the routing algorithm, as opposed to the order in which graph edges are explored [22]. Second, we strive to minimize congestion, which is a measure of resource utilization. To gauge congestion, we divide routing resources into disjoint groups according to functional similarity and physical proximity. For example, all channel edges interconnecting the same two switch blocks form a group, as do all edges inside a particular switch block. As nets are routed, the detailed router updates each group's congestion information (i.e., the number of edges in each group taken by all previously routed nets). Multi-objective optimization is used in the IKMB graph searches to heuristically minimize a combination of wirelength and congestion (See the Appendix for additional details). Thus, within the region specified by the global route, our detailed router searches for a feasible solution minimizing both congestion and wirelength.

We found that in practice, the majority of those nets that fail to route using the initial global route become routable after only a single loosening operation. In cases where wavefront expansion fails to produce a routing solution, we employ a "move-to-front" heuristic [36], where unroutable nets are moved to the beginning of the net-routing order and the new routing order is attempted.

The Iterated-KMB (IKMB) Algorithm
Input: A weighted graph $G = (V, E)$ and net $N \subseteq V$
Output: A low-cost tree spanning N
$S = \emptyset$
While $C = \{x \in V - N \mid \Delta\overline{\text{KMB}}_G(N \cup S, \{x\}) > 0\} \neq \emptyset$
Do Find $x \in C$ with maximum $\Delta\overline{\text{KMB}}_G(N \cup S, \{x\})$
$S = S \cup \{x\}$
Return $\text{KMB}_G(N \cup S)$

FIGURE 7 Iterated-KMB algorithm (IKMB).

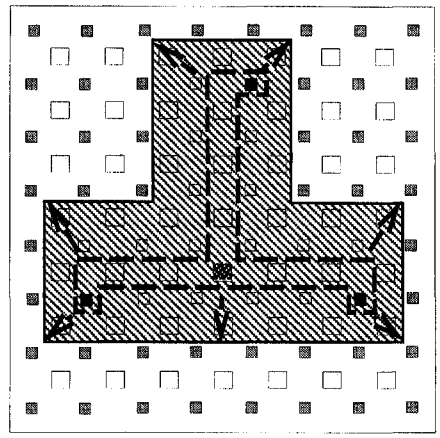


FIGURE 8 Wavefront expansion is used to "loosen" global routes when infeasibility is encountered.

6. EXPERIMENTAL RESULTS

Our algorithms have been implemented using C++ in the Sun/UNIX environment and incorporated into FPR. Two FPGA architectures, corresponding to Xilinx 3000-series and 4000-series parts, were modeled [7, 43] (these architectures are identical to the ones used by CGE [8], SEGA [32] and GPB [42], respectively). We compared the performance of these tools on fourteen large benchmark circuits: the suite of five 3000-series benchmarks used by [8], and the suite of nine 4000-series benchmarks used by [32] and [42]. The 3000-series benchmarks were routed on FPGAs with switch-block flexibility $F_s = 6$ and connection flexibility $F_c = \lceil 0.6 \times W \rceil$, where W is the the channel width. The 4000-series benchmarks use FPGAs with $F_s = 3$ and $F_c = W$.

During FPGA physical design, a common objective is to minimize maximum channel width. (Smaller channel width implies the ability to route larger designs on a fixed-size part). Table I shows the maximum channel widths of actual *complete* placement and routing solutions produced by FPR; these compare favorably with CGE [8] for the 3000-series benchmarks, and with SEGA [32] and GBP [42] for the 4000-series benchmarks. The channel width required by FPR is smaller than that required by CGE, SEGA, and GPB in 8 of the 14 benchmark circuits, and is equal on all but one of the remaining 6 benchmark circuits (further improvements have been recently obtained in [4]).

We also measured how well FPR optimizes total wirelength and maximum source-sink pathlengths or *radius*. Since previous works do not report these statistics, we have implemented a modified version of FPR, called FPR-S, that uses unrooted Steiner trees as thumbnails [17], instead of the preferred arborescence thumbnails de-

TABLE I Maximum channel width required by SEGA [32], GBP [42] and FPR on the benchmark circuits

3000-Series Benchmarks				
Name	Size	Nets	CGE	FPR
busc	13 × 12	151	10	**9**
dma	18 × 16	213	10	**9**
bnre	22 × 21	352	12	**11**
dfsm	23 × 22	420	10	**11**
z03	27 × 26	608	13	**13**
	Total		**55**	**53**

4000-Series Benchmarks					
Name	Size	Nets	SEGA	GBP	FPR
9symml	11 × 10	79	10	9	**9**
terml	10 × 9	88	10	10	**8**
apex7	12 × 10	115	13	11	**9**
alu2	15 × 13	153	11	11	**10**
too_large	14 × 14	186	12	12	**11**
example2	14 × 12	205	17	13	**13**
vda	17 × 16	225	13	13	**13**
alu4	19 × 17	255	15	14	**13**
k2	22 × 20	404	17	17	**17**
	Total		**118**	**110**	**103**

TABLE II Comparison of arborescence-based FPR against Steiner-tree-based FPR-S. Wirelength statistics reflect average number of channel segments used by nets in the circuit; radius statistics reflect average number of channel segments encountered on longest source-sink path for each net. The Δ% column gives the percent change from FPR-S to FPR

3000-Series Benchmarks						
	Avg. Wirelength			Avg. Max Radius		
Name	FPR-S	FPR	Δ%	FPR-S	FPR	Δ%
busc	9.3	9.1	−2.2	6.6	6.0	−9.1
dma	13.2	13.0	−1.5	8.6	7.7	−10.5
bnre	14.0	14.0	0.0	9.0	7.9	−12.2
dfsm	12.0	12.7	5.8	7.3	7.1	−2.7
z03	13.7	14.1	2.9	8.9	8.7	−2.2
Average	12.4	12.6	1.0	8.1	7.5	−7.3

4000-Series Benchmarks						
	Avg. Wirelength			Avg. Max Radius		
Name	FPR-S	FPR	Δ%	FPR-S	FPR	Δ%
9symml	11.4	10.9	−4.4	7.1	6.1	−14.1
terml	7.0	7.4	5.7	5.2	5.2	0.0
apex7	9.1	9.5	4.4	6.3	6.7	6.3
alu2	12.2	12.5	2.5	7.5	7.2	−4.0
too_large	11.9	11.5	−3.4	8.5	7.2	−15.3
example2	9.3	9.4	1.1	7.2	6.8	−5.6
vda	15.0	15.0	0.0	10.6	9.4	−11.3
alu4	14.5	14.9	2.8	9.5	9.0	−5.3
k2	17.7	17.7	0.0	13.1	12.1	−7.6
Average	12.0	12.1	1.0	8.3	7.7	−6.3
Overall	12.2	12.3	**1.0**	8.2	7.6	**−6.7**

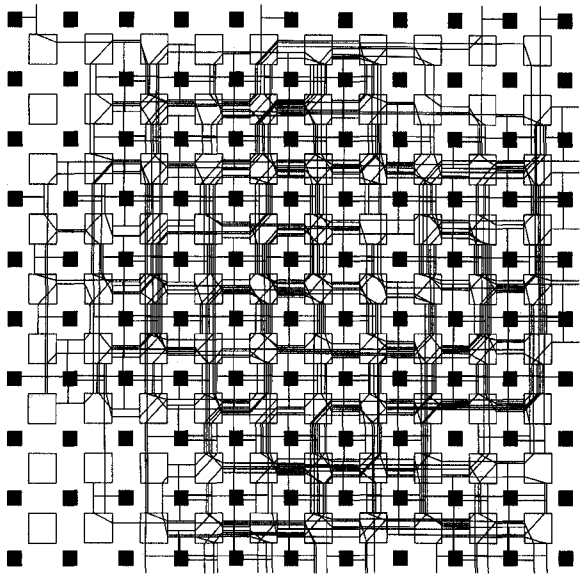


FIGURE 9 FPR solution for 9symml circuit.

scribed in Section 3. We compared the solutions produced by FPR-S against performance-oriented solutions produced by the unmodified FPR tool. We observe that the additional 1.0% in wirelength used by FPR yields a 6.7% decrease in radius (Tab. II). We believe the 1.0% total wirelength difference is insignificant but the 6.7% difference in average radius is significant. Therefore we recommend the use of FPR with its use of RSA's over FPR-S and other similar tree-based tools. The time to run FPR is comparable to other tools: CPU times to completely lay out the circuits on a Sun SparcServer 10/514 workstation ranged from several minutes for the smallest circuit to several hours for the largest. Figure 9 shows the solution produced by FPR for the smallest of the benchmark circuits.

7. CONCLUSION

We have developed FPR, a placement and routing tool for FPGAs that combines a recursive geometric strategy for simultaneous placement and global routing with a general graph-based detailed-routing algorithm. FPR addresses performance issues by minimizing source-sink pathlengths as well as total wirelength and maximum channel width. FPR compares favorably to existing tools on both 3000-series and 4000-series Xilinx-type parts, as measured by the maximum channel width required for complete layout of a number of industrial benchmarks.

Acknowledgement

We thank Matt Saltzman for the use of his matching code, and Steve Brown and Jonathan Rose for supplying the benchmark circuits. We are grateful to Dr. Bob Grafton of the National Science Foundation for his support and advice. This work is supported by NSF grants CCR-9224789 and MIP-9107717 (Cohoon), a Virginia Space Grant Fellowship (Ganley), a Packard Foundation Fellowship and NSF Young Investigator Award MIP-9457412 (Robins).

References

[1] Alexander, M. J., Cohoon, J. P., Ganley, J. L. and Robins, G. An Architecture-Independent Approach to FPGA Routing Based on Multi-Weighted Graphs, in *Proc. European Design Automation Conf.*, Grenoble, France, pp. 259–264, September 1994.

[2] Alexander, M. J., Cohoon, J. P., Ganley, J. L. and Robins, G. Performance-Oriented Placement and Routing for Field-Programmable Gate Arrays, in *Proc. European Design Automation Conf.*, Brighton, England, September 1995.

[3] Alexander, M. J. and Robins, G. A New Approach to FPGA Routing Based on Multi Weighted Graphs, in *Proc. ACM/SIGDA Intl. Workshop on Field-Programmable Gate Arrays*, Berkeley, CA, February 1994.

[4] Alexander, M. J. and Robins, G. (1996). New Performance-Driven FPGA Routing Algorithms, *IEEE Trans. Computer-Aided Design*, **15**, pp. 1505 – 1517.

[5] Bapat, S. and Cohoon, J. P. A Parallel VLSI Circuit Layout Methodology, in *Proc. IEEE Intl. Conf. VLSI Design*, pp. 236–241, January 1993.

[6] Bhat, N. B. and Hill, D. D. (1992). Routable Technology Mapping for LUT FPGAs, in *Proc. IEEE Intl. Conf. Computer-Aided Design*, pp. 95–98.

[7] Brown, S. D., Francis, R. J., Rose, J. and Vranesic, Z. G. (1992). Field-Programmable Gate Arrays, Kluwer Academic Publishers, Boston, MA.

[8] Brown, S. D., Rose, J. and Vranesic, Z. G. (1992). A Detailed Router for Field-Programmable Gate Arrays, *IEEE Trans. Computer-Aided Design*, **11**, pp. 620–628.

[9] Carter, W. S., Duong, K., Freeman, R. H., Hsieh, H. C., Ja, J. Y., Mahoney, J. E., Ngo, L. T. and Sze, S. L. (1986).

A User Programmable Reconfigurable Logic Array, in *Custom Integrated Circuits Conf.*, pp. 233–235.
[10] Chan, P. K., Schlag, M. D. F. and Zien, J. Y. (1993). On Routability Prediction for Field-Programmable Gate Arrays, in *Proc. ACM/IEEE Design Automation Conf.*, pp. 326–330.
[11] Chen, K. C., Cong, J., Ding, Y., Kahng, A. B. and Trajmar, P. (1992). DAG-Map: Graph-Based FPGA Technology Mapping for Delay Optimization, *IEEE Design and Test of Computers*, **9**, pp. 7–20.
[12] Cohoon, J. P. and Richards, D. S. (1988). Optimal Two-Terminal α–β Wire Routing, *Integration: The VLSI Journal*, **6**, pp. 35–57.
[13] Collier, W. C. and Weiland, R. J. (1994). Smart Cars, *Smart Highways*, *IEEE Spectrum*, **31**, pp. 27–33.
[14] Dunlop, A. E. and Kernighan, B. W. (1985). A Procedure for Placement of Standard-Cell VLSI Circuits, *IEEE Trans. Computer-Aided Design*, **4**, pp. 92–98.
[15] Frankle, J. (1992). Iterative and Adaptive Stock Allocation for Performance-driven Layout and FPGA Routing, in *Proc. ACM/IEEE Design Automation Conf.*, pp. 536–542.
[16] Gamal, A. E., Greene, J., Reyneri, J., Rogoyski, E., El-Ayat, K. and Mohsen, A. (1989). An Architecture for Electrically Configurable Gate Arrays, *IEEE J. Solid State Circuits*, **24**, pp. 394–398.
[17] Ganley, J. L. (1995). Geometric Interconnection and Placement Algorithms, PhD thesis, Department of Computer Science, University of Virginia, Charlottesville, Virginia.
[18] Ganley, J. L. and Cohoon, J. P. Routing a Multi-Terminal Critical Net: Steiner Tree Construction in the Presence of Obstacles, in *Proc. IEEE Intl. Symp. Circuits and Systems*, London, England, May 1994, pp. 113–116.
[19] Ganley, J. L., Golin, M. J. and Salowe, J. S. (1995). The Multi-Weighted Spanning Tree Problem, in *Proc. First Intl. Computing and Combinatorics Conf.*, Xian, China, pp. 141–150.
[20] Gao, T., Chen, K. C., Cong, J., Ding, Y. and Liu, C. L. Placement and Placement Driven Technology Mapping for FPGA Synthesis, in *Proc. IEEE Intl. ASIC Conf.*, Rochester, NY, September 1993, pp. 87–91.
[21] Griffith, J., Robins, G., Salowe, J. S. and Zhang, T. (1994). Closing the Gap: Near-Optimal Steiner Trees in Polynomial Time, *IEEE Trans. Computer-Aided Design*, **13**, pp. 1351–1365.
[22] Heyns, W., Sansen, W. and Beke, H. (1980). A Line-Expansion Algorithm for the General Routing Problem with a Guaranteed Solution, in *Proc. ACM/IEEE Design Automation Conf.*, Minneapolis, pp. 243–249.
[23] Hu, T. C. and Shing, T. (1985). The α–β Routing, in *VLSI Circuit Layout: Theory and Design*, New York, IEEE Press, pp. 139–143.
[24] Hwang, F. K., Richards, D. S. and Winter, P. (1992). The Steiner Tree Problem, North Holland.
[25] Kahng, A. B. and Robins, G. (1992). A New Class of Iterative Steiner Tree Heuristics with Good Performance, *IEEE Trans. Computer-Aided Design*, **11**, pp. 893–902.
[26] Kahng, A. B. and Robins, G. (1995). On Optimal Interconnections for VLSI, Kluwer Academic Publishers, Boston, MA.
[27] Karplus, K. (1991). Xmap: a Technology Mapper for Table-lookup Field-Programmable Gate Arrays, in *Proc. ACM/IEEE Design Automation Conf.*, pp. 240–243.
[28] Kirkpatrick, S., Gelatt, C. D. and Vecchi, M. P. (1983). Optimization by Simulated Annealing: An Experimental Evaluation (PART 1), *Science*, **220**, pp. 671–680.
[29] Kou, L., Markowsky, G. and Berman, L. (1981). A Fast Algorithm for Steiner Trees, *Acta Informatica*, **15**, pp. 141–145.
[30] Lawler, E. L. (1976). Combinatorial Optimization: Networks and Matroids, Holt Rinehart and Winston, New York.
[31] Lee, Y.-S. and Wu, A. C.-H. A Performance and Routability Driven Router for FPGAs Considering Path Delays, in *Proc. ACM/IEEE Design Automation Conf.*, San Francisco, CA, June 1995, pp. 557–561.
[32] Lemieux, G. G. and Brown, S. D. A Detailed Routing Algorithm for Allocating Wire Segments in Field-Programmable Gate Arrays, in *Proc. ACM/SIGDA Physical Design Workshop*, Lake Arrowhead, CA, April 1993.
[33] Papadimitriou, C. H. and Steiglitz, K. (1982). *Combinatorial Optimization*, Prentice-Hall.
[34] Rao, S. K., Sadayappan, P., Hwang, F. K. and Shor, P. W. (1992). The Rectilinear Steiner Arborescence Problem, Algorithmica, pp. 277–288.
[35] Roy, K., Guan, B. and Sechen, C. FPGA MCM Partitioning and Placement, in *Proc. ACM/SIGDA Physical Design Workshop*, Lake Arrowhead, CA, April 1993, pp. 211–212.
[36] Shin, H. and Sangiovanni-Vincentelli, A. (1987). A Detailed Router Based on Incremental Routing Modifications: Mighty, *IEEE Trans. Computer-Aided Design*, **6**, pp. 942–955.
[37] Spruth, H., Johannes, F. and Antreich, K. (1994). PHIroute: A Parallel Hierarchial Sea-of-Gates Router, in *Proc. IEEE Intl. Symp. Circuits and Systems*, pp. 487–490.
[38] Suaris, P. R. and Kedem, G. (1989). A Quadrisection-Based Place and Route Scheme for Standard Cells, *IEEE Trans. Computer-Aided Design*, **8**, pp. 234–244.
[39] Trimberger, S. Effects of FPGA Architecture on FPGA Routing, in *Proc. ACM/IEEE Design Automation Conf.*, San Francisco, CA, June 1995, pp. 574–578.
[40] Trimberger, S. M. (1994). Field-Programmable Gate Array Technology, *S. M. Trimberger, editor*, Kluwer Academic Publishers, Boston, MA.
[41] Wu, Y.-L. and Marek-Sadowska, M. (1993). Graph Based Analysis of FPGA Routing, in *Proc. European Design and Test Conf.*, pp. 104–109.
[42] Wu, Y.-L. and Marek-Sadowska, M. (1994). An Efficient Router for 2-D Field Programmable Gate Arrays, in *European Design and Test Conf.*, pp. 412–416.
[43] Xilinx, The Programmable Gate Array Data Book, (1994). Xilinx, Inc., San Jose, California.

8. APPENDIX: MULTI-OBJECTIVE OPTIMIZATION

During the detailed-routing phase we seek to simultaneously optimize multiple (competing) objectives (i.e., wirelength, congestion, jogs, etc.). We accomplish this by generalizing the IKMB heuris-

tic to operate on multi-weighted graphs, where each of the k optimization criteria is modeled by a separate set of edge weights. The simultaneous optimization is accomplished by transforming these multiple edge weights into a single weighted average, which is then used by IKMB in the normal way. The relative magnitudes of the weighting factors $d_1, d_2, \ldots, d_k$ (i.e., tradeoff parameters) are designer controlled, enabling a smooth tradeoff among the various competing objectives.

This technique is flexible in that new criteria are easily incorporated into the model by introducing additional weight sets into the graph. Such a framework subsumes e.g., "alpha-beta" routing (which has been used for jog minimization in IC design [12, 23]), and also has practical application in non-VLSI domains [13].

Let $V = \{v_1, v_2, \ldots, v_n\}$ be a set of nodes, and let $E \subseteq V \times V$ be a set of edges. We define a k-weighted graph $G = (V, E)$ to be a weighted graph with a vector-valued weight function $\vec{w} : E \rightarrow \Re^k$. In other words, associated with each edge $e_{ij} \in E$ is a vector of k real-valued weights $\vec{w}_{ij} = (w_{ij1}, w_{ij2}, \ldots, w_{ijk})$. Note that ordinary weighted graphs are a special case of k-weighted graphs, with $k = 1$.

Let $\vec{d} = (d_1, d_2, \ldots, d_k)$ be a vector of k real-valued *tradeoff parameters*, where $0 \leq d_i \leq 1$ for $0 \leq i \leq k$, and $\Sigma_{i=1}^{k} d_i = 1$. From the k-weighted graph $G = (V, E)$ and the tradeoff parameters $\vec{d}$ we construct a new weighted *tradeoff graph* $\hat{G}(\vec{d}) = (V, E)$ with weight function $w'_{ij} = \vec{d} \cdot \vec{w}_{ij} = \Sigma_{m=1}^{k} d_m \cdot w_{ijm}$. The tradeoff graph $\hat{G}$ is an ordinary weighted graph having the same topology as G, but whose single edge weights represent the weighted averages of the multi-weights of G, with respect to $\vec{d}$.

Let $\vec{u} = (1, \ldots, 1)$, and $\vec{v}_i = (0, \ldots, 0, v_i, 0, \ldots, 0)$ denote the vector obtained from the vector $\vec{v}$ by using v_i in the i^{th} place, and the rest of the places set to zero. Thus, $\vec{u}_i$ denotes the vector consisting of zeros everywhere except the i^{th} place, which will contain a 1. A k-weighted graph G induces k distinct graphs $G_i = \hat{G}(\vec{u}_i)$, each with an identical topology but with edge weights restricted to only one of the k components of vector-valued weight function $\vec{w}$.

We define the minimum spanning tree for a multi-weighted graph G with respect to the tradeoff parameters $\vec{d}$ as the ordinary MST over the tradeoff graph $\hat{G}(\vec{d})$, and denote it by $\text{MST}(\hat{G}(\vec{d}))$. Similarly, we can compute the MST on each of the k induced graphs G_i, and we denote these $\text{MST}(G_i)$. For convenience we will use $\overline{\text{MST}}$ to denote the cost of the MST.

We start by showing a general lower bound for the cost of $\overline{\text{MST}}(\hat{G}(\vec{d}))$ in terms of $\overline{\text{MST}}(G_i)$'s, $\vec{d}$, and k:

THEOREM 8.1 *For any k-weighted graph G and tradeoff parameters $\vec{d}$,*

$$\sum_{i=1}^{k} d_i \cdot \overline{\text{MST}}(G_i) \leq \overline{\text{MST}}(\hat{G}(\vec{d})).$$

Proof Consider an arbitrary edge e_{ij} in $\text{MST}(\hat{G}(\vec{d}))$ having cost of $\Sigma_{m=1}^{k} d_m \cdot w_{ijm}$. If every $\text{MST}(G_m)$, $1 \leq m \leq k$, also contains edge e_{ij}, then clearly the cost of edge e_{ij} in all k trees is $\Sigma_{m=1}^{k} w_{ijm}$, and the cost of this edge scaled by the tradeoff parameters $\vec{d}$ is $\Sigma_{m=1}^{k} d_m \cdot w_{ijm}$, which is equal to the cost of this edge in $\text{MST}(\hat{G}(\vec{d}))$. Clearly, if all of the k $\text{MST}(G_m), 1 \leq m \leq k$ contain the same edges as $\text{MST}(\hat{G}(\vec{d}))$, then equality holds and the theorem is true. On the other hand, if $\text{MST}(\hat{G}(\vec{d}))$ contains an edge that is not in $\text{MST}(G_m)$, $1 \leq m \leq k$, then the cost of $\text{MST}(\hat{G}(\vec{d}))$ relative to $\Sigma_{m=1}^{k} d_m \cdot \overline{\text{MST}}(G_m)$ can only increase. □

Next, we prove the non-existence of general upper bounds. Ideally, we would like to bound the MST cost of arbitrary multi-weighted graphs in terms of only the costs of the $\text{MST}(G_i)$'s, $\vec{d}$, and n. Unfortunately, this is impossible to in general:

THEOREM 8.2 *For any k-weighted graph G over n vertices, and tradeoff parameters $\vec{d}$, the tradeoff graph cost $\overline{\text{MST}}(\hat{G}(\vec{d}))$ can not be bounded from above by any function of only $\overline{\text{MST}}(G_i)$'s, $\vec{d}$, n, and k.*

Proof Consider the 2-weighted graph $G = (V, E)$ over $n = 3$ nodes, where $k = 2$. Fix $\vec{d}$ by setting

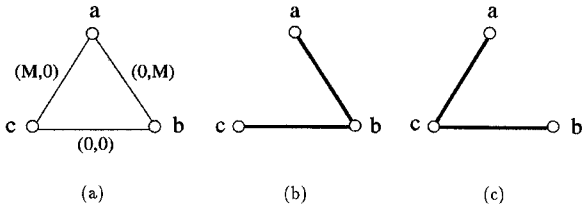


FIGURE 10 An example showing that $\overline{\text{MST}}(\hat{G}(\vec{d}))$ can not be bounded from above by any function strictly in terms of $\overline{\text{MST}}(G_i)$'s, $\vec{d}$, n, and k: (a) The 2-weighted graph G; (b) MST($G((1,0))$) has cost 0; (c) MST($G((0,1))$) has cost 0. On the other hand, MST($G((\frac{1}{2},\frac{1}{2}))$) has cost $\frac{M}{2}$, which can be arbitrarily large.

$0 \le d_1, d_2 \le 1$. Let M be some very large constant, $V = \{a, b, c\}$, and $E = V \times V$, with $w_{ab1} = 0$, $w_{bc1} = 0$, $w_{ac1} = M$, and let $w_{ab2} = M$, $w_{bc2} = 0$, $w_{ac2} = 0$ (see Fig. 10). Observe that $\overline{\text{MST}}(G_1) = \overline{\text{MST}}(G_2) = 0$, $k = 2$, $n = 3$, d_1, and d_2 are all constants. On the other hand, $\overline{\text{MST}}(\hat{G}) = \min(d_1 \cdot M, d_2 \cdot M)$, which can be made arbitrarily large for any fixed $\vec{d}$ by making M large enough. Since any expression involving only constants must also be bounded by a constant, $\overline{\text{MST}}(\hat{G})$ can not be bounded from above by any function strictly in terms of only $\overline{\text{MST}}(G_1)$, $\overline{\text{MST}}(G_2)$, k, n, and $\vec{d}$. □

The negative result of Theorem 8.2 only applies to non-metric graphs. We now give a general upper bound for metric graphs:

THEOREM 8.3 *For any metric k-weighted graph G over n vertices, and tradeoff parameters $\vec{d}$, $\overline{\text{MST}}(\hat{G}(\vec{d})) \le (n-1) \cdot \Sigma_{i=1}^{k} d_i \cdot \overline{\text{MST}}(G_i)$.*

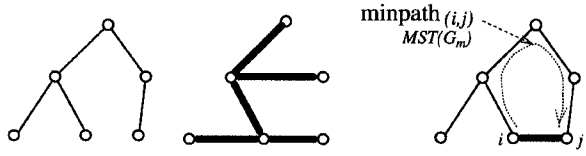


FIGURE 11 A general upper bound in the metric case for $\overline{\text{MST}}(\hat{G}(\vec{d}))$ in terms of $\overline{\text{MST}}(G_i)$'s, $\vec{d}$, n, and k: (a) depicts MST (G_m); (b) depicts MST($\hat{G}(\vec{d})$); and (c) shows how the cost of the m^{th} weight component of each e_{ij} can be bounded by $d_m \cdot \overline{\text{MST}}(G_m)$.

Proof Consider an arbitrary edge e_{ij} in MST($\hat{G}(\vec{d})$)and its cost, $\Sigma_{m=1}^{k} d_m \cdot w_{ijm}$. Consider the m^{th} element in this summation, and the corresponding MST of G_m. MST(G_m) spans vertices v_i and v_j, but does not necessarily contain the edge e_{ij}. Rather, a path must exist in MST(G_m) from v_i to v_j, denoted $\text{minpath}_{\text{MST}(G_m)}(i,j)$ with cost denoted by $\text{dist}_{\text{MST}(G_m)}(i,j)$. By metricity, $w_{ijm} \le \text{dist}_{\text{MST}(G_m)}(i,j)$. Therefore:

$$\begin{aligned}\text{cost of edge } e_{ij} \text{ in } \text{MST}(\hat{G}(\vec{d})) &= \sum_{m=1}^{k} d_m \cdot w_{ijm} \\ &\le \sum_{m=1}^{k} d_m \cdot \text{dist}_{\text{MST}(G_m)}(i,j) \\ &\le \sum_{m=1}^{k} d_m \cdot \overline{\text{MST}}(G_m)\end{aligned}$$

Since e_{ij} is an arbitrary edge of MST($\hat{G}(\vec{d})$), this holds for all $n-1$ edges in MST($\hat{G}(\vec{d})$). Thus, $\overline{\text{MST}}(\hat{G}(\vec{d})) \le (n-1) \cdot \Sigma_{m=1}^{k} d_m \cdot \overline{\text{MST}}(G_m)$. □

Since most nets in typical VLSI designs contain three pins or less [18], we derive a tighter upper bound for 3-pin nets where metricity holds (i.e., graphs with weight functions satisfying the triangle inequality $\text{dist}(a,b) + \text{dist}(b,c) \ge \text{dist}(a,c)$, $\forall a, b, c \in V$):

THEOREM 8.4 *For 2-weighted metric graphs with three nodes, and any scaling vector $\vec{d} = (d_1, d_2)$, the following holds:*

$$\begin{aligned} d_1 \cdot \overline{\text{MST}}(G_1) + d_2 \cdot \overline{\text{MST}}(G_2) &\le \overline{\text{MST}}(\hat{G}(\vec{d})) \\ &\le \frac{4}{3} \cdot [d_1 \cdot \overline{\text{MST}}(G_1) + d_2 \cdot \overline{\text{MST}}(G_2)]. \end{aligned}$$

Proof Let $G = (V, E)$ be a 3-node 2-weighted graph, with edge weights (a, x), (b, y), and (c, z). Let $\vec{d} = (d_1, d_2)$ be an arbitrary constant vector, such that $0 \le d_1$, $d_2 \le 1$, and $d_1 + d_2 = 1$ (see Fig. 12(i)).

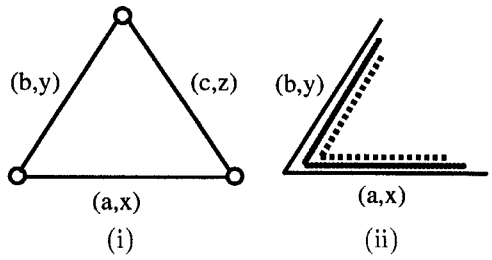


FIGURE 12 A tighter upper bound for 3-pin nets; (i) a 3-node 2-weighted graph, with edge weights (a,x), (b,y), and (c,z); (ii) topology of the three spanning trees MST(G_2) (inner), MST(G_1) (middle) and MST($\hat{G}(\vec{d})$) (outer) corresponding to case 1.

The lower bound $d_1 \cdot \overline{\text{MST}}(G_1) + d_2 \cdot \overline{\text{MST}}(G_2) \leq \overline{\text{MST}}(\hat{G}(\vec{d}))$ holds by Theorem 8.1. Assume without loss of generality that $a \leq b \leq c$, which implies that $\overline{\text{MST}}(G_1) = a + b$. The following three cases must be considered:

1. Assume $x, y \leq z$, which implies that $\overline{\text{MST}}(G_2) = x + y$ (see Fig. 12(ii)). Thus,

$$\begin{aligned} &d_1 \cdot a + d_2 \cdot x \leq d_1 \cdot c + d_2 \cdot z \text{ and} \\ &d_1 \cdot b + d_2 \cdot y \leq d_1 \cdot c + d_2 \cdot z \\ \text{Now } \overline{\text{MST}}(\hat{G}(\vec{d})) &= d_1 \cdot a + d_2 \cdot x \\ &\quad + d_1 \cdot b + d_2 \cdot y \\ &= d_1 \cdot (a+b) + d_2 \cdot (x+y) \\ &= d_1 \cdot \overline{\text{MST}}(G_1) \\ &\quad + d_2 \cdot \overline{\text{MST}}(G_2) \end{aligned}$$

   and the theorem holds.
2. Assume $x, z \leq y$, which implies that $\overline{\text{MST}}(G_2) = x + z$. Let $\overline{G} = d_1 \cdot \overline{\text{MST}}(G_1) + d_2 \cdot \overline{\text{MST}}(G_2)$, and consider the three possible sub-cases illustrated in Figure 13.

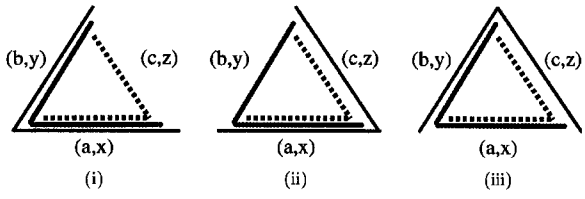


FIGURE 13 Topology of the three spanning trees MST(G_2) (inner), MST (G_1) (middle) and MST($\hat{G}(\vec{d})$) (outer) corresponding to (i) case 2(a), (ii) case 2(b), and (iii) 2(c).

(2a) Assume MST($\hat{G}(\vec{d})$) contains the "a/x" and "b/y" edges (see Fig. 13(i)). Then,

$$\begin{aligned} \overline{\text{MST}}(\hat{G}(\vec{d})) &= d_1 \cdot (a+b) + d_2 \cdot (x+y) \\ &\leq d_1 \cdot (a+b) + d_2 \cdot (x+x+z) \\ &= d_1 \cdot \overline{\text{MST}}(G_1) \\ &\quad + d_2 \cdot \overline{\text{MST}}(G_2) + d_2 \cdot x \\ &= \overline{G} + d_2 \cdot x \end{aligned}$$

(2b) Assume MST($\hat{G}(\vec{d})$) contains the "a/x" and "c/z" edges (see Fig. 13(ii)). Then,

$$\begin{aligned} \overline{\text{MST}}(\hat{G}(\vec{d})) &= d_1 \cdot (a+c) + d_2 \cdot (x+z) \\ &\leq d_1 \cdot (a+a+b) \\ &\quad + d_2 \cdot (x+z) \\ &= d_1 \cdot \overline{\text{MST}}(G_1) + d_1 \cdot a \\ &\quad + d_2 \cdot \overline{\text{MST}}(G_2) \\ &= \overline{G} + d_1 \cdot a \end{aligned}$$

(2c) Assume MST($\hat{G}(\vec{d})$) contains the "b/y" and "c/z" edges (see Fig. 13(iii)). Then,

$$\begin{aligned} \overline{\text{MST}}(\hat{G}(\vec{d})) &= d_1 \cdot (b+c) + d_2 \cdot (y+z) \\ &\leq d_1 \cdot (b+a+b) \\ &\quad + d_2 \cdot (x+z+z) \\ &= d_1 \cdot \overline{\text{MST}}(G_1) + d_1 \cdot b \\ &\quad + d_2 \cdot \overline{\text{MST}}(G_2) + d_2 \cdot z \\ &= \overline{G} + d_1 \cdot b + d_2 \cdot z \end{aligned}$$

Now, since MST($\hat{G}(\vec{d})$) is a *minimum* spanning tree, it is the minimum of sub-cases 2(a), 2(b) and 2(c).

$$\begin{aligned} &\overline{\text{MST}}(\hat{G}(\vec{d})) \\ &\quad = \overline{G} + \min(d_2 \cdot x, d_1 \cdot a, d_1 \cdot b + d_2 \cdot z) \\ &\quad \leq \overline{G} + \frac{1}{3} \cdot (d_2 \cdot x + d_1 \cdot a \\ &\qquad + d_1 \cdot b + d_2 \cdot z) \\ &\quad = \frac{4}{3} \cdot \overline{G} \end{aligned}$$

3. Assume $y, z \leq x$, which implies that $\overline{\text{MST}}(G_2) = y + z$. Again, let $\overline{\text{G}} = d_1 \cdot \overline{\text{MST}}(G_1) + d_2 \cdot \overline{\text{MST}}(G_2)$, and consider the three possible subcases corresponding to whether $\text{MST}(\widehat{G}(\vec{d}))$ contains the 3(a) "a/x" and "b/y", 3(b) "a/x" and "c/z", or 3(c) "b/y" and "c/z" edges, which are handled using similar arguments to those in case 2 above.

The bound $\overline{\text{MST}}(\widehat{G}(\vec{d})) \leq 4/3 \cdot [d_1 \cdot \overline{\text{MST}}(G_1) + d_2 \cdot \overline{\text{MST}}(G_2)]$ holds in each one of the three possible cases (1, 2 and 3). The example $a = \varepsilon$, $x = 2-\varepsilon$, $b = c = y = z = 1$ (where $\varepsilon > 0$ is an arbitrarily small real number) shows that this bound is tight. □

For 4-node graphs the general upper bound of Theorem 8.3 implies a multiplicative factor of $n-1 = 3$; yet, an extensive computer-aided search has been unable to find an example of a 4-pin net with metric weights where the cost of the tradeoff MST exceeds the lower bound by more than a factor of 3/2. We therefore conjecture that our proven bounds can be made considerably tighter, and leave this as an open problem (recently, tighter bounds were indeed derived for MSTs over multi-weighted graphs [19]).

Authors' Biographies

Michael J. Alexander received the Ph.D. degree in Computer Science from the University of Virginia, Charlottesville in 1996, where he won a Teaching Assistant Extraordinaire award. He is currently an Assistant Professor in the School of Electrical Engineering and Computer Science at Washington State University. His primary areas of research are VLSI CAD, with research focusing on high-performance routing, FPGA architecture, reconfigurable computing, and combinatorial optimization. He serves on the program committees of the IEEE International ASIC Conference, ACM/SIGDA International Symposium on Physical Design, and the Canadian Workshop on Field-Programmable Devices. He is a member of ACM, IEEE, SIGDA, SIGARCH and Tau Beta Pi.

James P. Cohoon received his Ph.D. in Computer Science from the University of Minnesota in 1983. He then joined the Department of Computer Science at the University of Virginia where he is currently an Associate Professor. His department has twice nominated him for the University's best teacher award. His primary research interests lie in VLSI circuit layout area with particular emphasis on algorithmic aspects of routing and placement. He is the author or co-author of over fifty papers and two books. Other research interests include computational geometry, parallel algorithms, testing and visualization. He has served on the programming committees for such conferences as DAC, ICCAD, and ICCD, and was co-organizer of the first ACM Design Automation Workshop in Russia. He is Chair of ACM-SIGDA and a member of the ACM and IEEE Circuits and Systems professional societies. His honors include a Fulbright Award and the SIGDA Leadership award.

Joseph L. Ganley received his Ph.D. in Computer Science in 1995 from the University of Virginia. His primary research interests are in VLSI physical design automation, geometric and graph algorithms, scientific computing, and parallel algorithms. He is a member of ACM, SIGACT, SIGDA, SIAM, and Tau Beta Pi. His honors include a University of Virginia Dean's Fellowship and a Virginia Space Grant Fellowship, and his doctoral dissertation was nominated for the 1995 ACM Doctoral Dissertation Award. He is currently a Member of the Research and Development Staff at Cadence Design Systems.

Gabriel Robins is Associate Professor in the Department of Computer Science at the University of Virginia, where he received an NSF Young Investigator Award, a Packard Foundation Fellowship, a University Teaching Fellowship, an All-University Outstanding Teaching Award, a Faculty Mentor Award, a two-year early promotion/tenure, and the Walter N. Munster Chair. He completed his Ph.D. in Computer Science in 1992 from UCLA, where he received an IBM Fellowship and a Distinguished Teaching Award. Gabe's primary area of research is VLSI CAD, and he

co-authored a book on high-performance routing and over fifty refereed papers, including a Distinguished Paper at the 1990 IEEE International Conference on Computer-Aided Design. Gabe is a member of the Defense Science Study Group, an advisory panel to the U.S. Department of Defense, and he also served on the Navy Future Study panel of the National Academy of Sciences. He was General Chair of the 1996 ACM/SIGDA Physical Design Workshop, and a co-founder of the 1997 International Symposium on Physical Design. He serves on the technical program committees of several other leading conferences, and the Editorial Board of the IEEE Book Series. He is a member of ACM, IEEE, MAA, and SIAM.