New Performance-Driven FPGA Routing Algorithms *

Michael J. Alexander† and Gabriel Robins‡

† School of Electrical Engineering and Computer Science, Washington State University, Pullman, WA 99165-2752
‡ Department of Computer Science, University of Virginia, Charlottesville, VA 22903-2442

Abstract

Motivated by the goal of increasing the performance of FPGA-based designs, we propose new Steiner and arborescence FPGA routing algorithms. Our Steiner tree constructions significantly outperform the best known ones and have provably-good performance bounds. Our arborescence heuristics produce routing solutions with optimal source-sink pathlengths, and with wirelength on par with the best existing Steiner tree heuristics. We have incorporated these algorithms into an actual FPGA router, which routed a number of industrial circuits using channel width considerably smaller than is achievable by previous routers. Our routing results for both the 3000 and 4000 -series Xilinx parts are currently the best known in the literature.

1 Introduction

Field-Programmable Gate Arrays (FPGAs) are flexible and reusable high-density circuits that can be easily (re)configured by the designer, enabling the VLSI design/validation/simulation cycle to be performed more quickly and cheaply [38]. Unfortunately, the flexibility provided by FPGAs is achieved at a substantial performance penalty due to signal delay through the programmable routing resources, and this is currently a primary concern to both FPGA designers and users [35]. In order to increase FPGA performance, partitioning and technology mapping have been extensively studied by e.g. [13, 20, 33], where a typical goal is to minimize the length of critical paths. On the other hand, less attention has been focused on the actual routing, which is surprising since it was observed that FPGA performance is frequently limited by routing delays, rather than by combinational logic delays [10, 18].

Routing affects the performance of FPGA-based systems in two major ways. First, a typical design must be partitioned and mapped onto several FPGAs. Since off-chip signal propagation delays are significantly longer than on-chip delays, we seek to minimize the number of such partitions. Because the FPGA size is fixed, the ability to pack larger partitions onto a single FPGA can reduce the total number of partitions

*Professor Gabriel Robins is supported by a Packard Foundation Fellowship and by NSF Young Investigator Award MIP-9457412. Correspondence should be directed to Professor Gabriel Robins, Department of Computer Science, Thornton Hall, University of Virginia, Charlottesville, VA 22903-2442, USA, Email: robins@cs.virginia.edu, WWW URL http://www.cs.virginia.edu/~robins/, phone: (804) 982-2207.

(and hence FPGAs) required to implement a design. The feasibility of implementing a piece of the design on a single FPGA is often limited by routing resource availability; this motivates Steiner routing constructions for minimizing the usage of routing resources.

Second, since FPGA utilization typically does not exceed 80%, considerable flexibility remains onboard the FPGA for optimizing the routing. For example, we may wish to reduce signal propagation delay through critical paths by using the most direct interconnections (i.e., shortest paths), where a secondary criterion is to minimize overall wirelength in order to reduce circuit capacitance and conserve routing resources. This motivates *Steiner arborescence* constructions (i.e., shortest paths trees having minimum wirelength) for critical-net routing.

Our first contribution is a new class of algorithms for non-critical-net routing, which in practice significantly outperform the best known graph Steiner tree heuristics, i.e., those of Kou, Markowsky and Berman [26], and of Zelikovsky [39]. Our graph Steiner construction is based on an iterative template that utilizes an existing Steiner tree heuristic H by greedily selecting Steiner nodes that induce maximum wirelength savings with respect to H. The performance bound of this new Steiner construction is guaranteed to be at least as good as that of the heuristic H thus used, and in particular we achieve a performance bound of $\leq \frac{11}{6}$ times optimal when using our template in concert with the graph Steiner heuristic of [39].

Our second contribution is a pair of arborescence-based constructions for critical-net routing. Given an arbitrary weighted routing graph, our arborescence algorithms produce a Steiner tree where all source-sink paths are the shortest possible, and where total wirelength is optimized as a secondary objective. Our first graph Steiner arborescence heuristic is based on an effective *path-folding* strategy that overlaps and merges shortest paths in order to reduce the overall wirelength. Our second arborescence heuristic iteratively selects Steiner nodes which improve the total wirelength with respect to an optimal spanning arborescence algorithm. Our constructions can be easily tuned to the specific parasitics of the underlying technology (the advantages of technology-sensitive routing were discussed and analyzed in, e.g., [11, 15]).

Finally, we incorporated our algorithms into an actual FPGA router, and successfully routed several large industry benchmark circuits, using considerably less channel width than is achievable by other routers (e.g., the CGE/SEGA routers of [12, 27] require channel width 22% larger than our own router). Our routing benchmarks on both 3000- and 4000- series Xilinx parts are currently the best known among all published results. Our experimental results also indicate that the total wirelength used by our arborescence constructions is on par with the best known Steiner tree heuristics. This is particularly significant, since our arborescence solutions have optimal source-sink pathlengths, while Steiner tree heuristics are designed to *only* optimize total wirelength (and thus yield source-sink pathlengths that are frequently far from optimal).

The remainder of the paper is organized as follows: Section 2 describes a typical FPGA architecture, reviews previous FPGA routing work, and formulates the FPGA routing problem. Section 3 describes our graph Steiner routing construction for routing non-critical nets, while Section 4 presents our graph Steiner arborescence algorithms for critical-net routing. Section 5 outlines the experimental results, and Section 6 concludes with future research directions. Parts of this work appeared in preliminary form in [6, 7, 8].

2 Problem Formulation

An FPGA architecture typically consists of a *symmetrical array* of user configurable logic "blocks", and a set of programmable interconnection resources used for routing [12, 36] (See Figure 1). Each logic block implements a portion of the design logic, and the routing resources are used to interconnect the logic blocks. This paper focuses on the routing phase of FPGA design; thus, we assume that partitioning, technology mapping, and placement have already been performed.

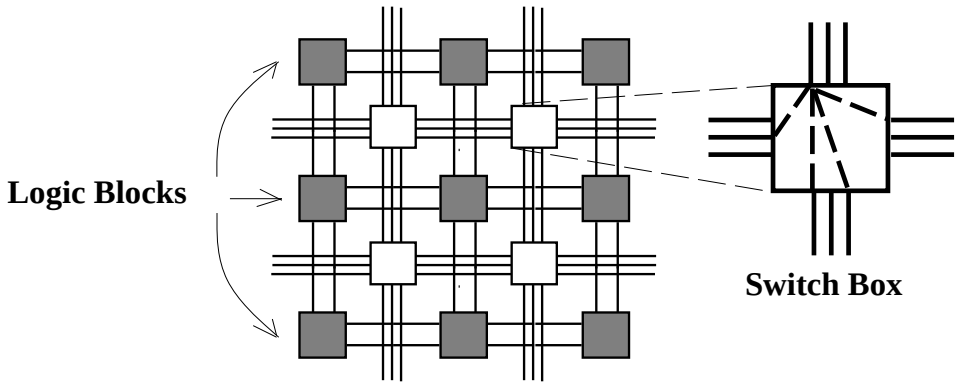

Figure 1: A symmetrical-array FPGA showing some of the logic blocks and programmable interconnection resources.

Previous work on FPGA routing has primarily concentrated on producing feasible solutions that use the fewest routing resources. For example, the CGE [12], and SEGA [27] detailed routing algorithms route nets based on demand and assign critical nets a higher routing priority. Other research has adopted a more abstract model of FPGA routing connections [28], studied FPGA routing with switch blocks of limited flexibility [37], or explored modified architectures [34] in order to reduce the number of programmable switches required. More recently, [4, 7] developed a routing framework where mutually competing objectives (such as congestion, wirelength, and jog minimization) may be simultaneously optimized. However, none of these works *directly* minimizes the source-sink signal propagation delays. While these approaches implicitly equate delay minimization with wirelength optimization [23, 31], it has recently become clear that these

two goals are not synonymous [11], especially for deep submicron VLSI technologies.

The bounded-radius bounded-cost (BRBC) method of [14] and the AHHK method of [9] both achieve wirelength-radius tradeoffs in weighted graphs, but can not directly produce a shortest paths tree with minimum wirelength. Rather, with the tradeoff parameter tuned completely towards pathlength minimization, the methods of [14] and [9] both produce the same shortest-paths tree as would Dijkstra's algorithm [16]. The recent A-Tree algorithm of [15] for rectilinear arborescence Steiner trees depends heavily on the Manhattan norm, and is therefore not suitable for the graph-based domains that arise naturally in FPGA routing.

Before we can apply graph-based techniques to FPGA routing, we must first model the FPGA as a graph, where the overall graph topology mirrors the complete FPGA architecture; paths in this graph correspond to feasible routes on the FPGA, and conversely. Figure 2 illustrates how the routing graph is constructed for symmetrical-array FPGAs. Let $G = (V, E)$ denote such a graph, where each graph edge $e_{ij} \in E$ has a weight w_{ij}, which typically corresponds to the wirelength of the associated FPGA routing wire segment (weights may also reflect parasitics, congestion, jog penalties, etc.). A *net* $N = \{n_0, n_1, ..., n_k\} \subseteq V$ is a set of pins that are to be electrically connected, where n_0 is the signal source and the remaining pins are sinks. A routing solution for a net is a tree $T \subseteq G$ which spans N, and the *cost* of a tree T, denoted $cost(T)$, is the sum of the weights of its edges.

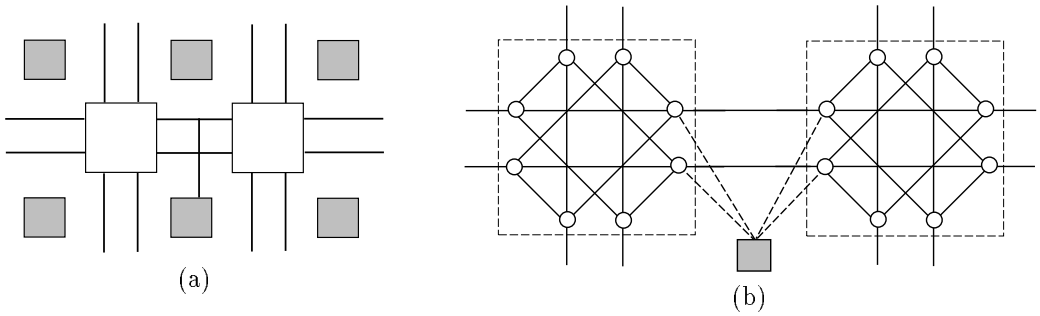

Figure 2: Construction of a routing graph for modeling symmetrical-array FPGAs. In (a) we see a portion of the FPGA architecture, while (b) illustrates the corresponding induced routing graph.

Prior to routing, nets may be classified as either *critical* or *non-critical* based on timing information from the higher-level design stages. The technology-mapping phase of FPGA design transforms a boolean network (the output from a high-level design tool) into a circuit consisting of logic blocks [12], where the goal is typically to minimize the maximum input-to-output circuit depth, thereby improving the overall

circuit performance. To a first approximation, nets through which long input-to-output paths pass may be designated as critical nets, with the remaining nets designated as non-critical. Determining which nets are critical vs. non-critical is outside the scope of this paper; rather, we focus on routing algorithms which may be applied to nets in *either* category.

When routing *non*-critical nets, we seek not to optimize delay but rather to maximize the overall feasibility of achieving a complete routing of all the nets onto a single FPGA; this resource-usage minimization objective motivates the following graph Steiner tree formulation:

The Graph Minimal Steiner Tree (GMST) Problem: Given a weighted graph $G = (V, E)$, and a net $N \subseteq V$, find a spanning tree $T = (V', E')$ with $N \subseteq V' \subseteq V$ and $E' \subset E$ such that $cost(T)$ is minimum.

Any node in $V - N$ may be used as a potential Steiner point in order to optimize the overall wirelength. The GMST problem is known to be NP-complete [22]. In Section 3 we address the GMST problem using an effective greedy strategy.

The high-performance requirement of critical nets dictates a shortest source-sink paths objective, with wirelength minimization being a secondary optimization criteria. For a weighted graph $G = (V, E)$ and two nodes $u, v \in V$, let $minpath_G(u, v)$ denote the *cost* of a shortest path between u and v in G. We thus formulate the graph Steiner arborescence problem as follows:

The Graph Steiner Arborescence (GSA) Problem: Given a weighted graph $G = (V, E)$, and a net $N \subseteq V$ to be routed in G, construct a *least-cost* spanning tree $T = (V', E')$ with $N \subseteq V' \subseteq V$ and $E' \subset E$ such that $minpath_T(n_0, n_i) = minpath_G(n_0, n_i)$ for all $n_i \in N$.

Since the GSA problem is NP-complete (a reduction from *Exact 3-Cover* is straightforward [19]), and the size of FPGA routing graphs is generally large (typically $|V| > 5000$), solving the GSA problem optimally is not feasible. In Section 4 we address the GSA problem with two new effective constructions.

Before the first net is routed, the routing graph resembles a grid-graph with shortest paths between nodes reflecting rectilinear distance, as illustrated in Figure 3(a). However, as nets are routed and resources are committed to specific nets, edges in the graph become unavailable. As a result, shortest paths in the graph may require detours that no longer reflect the original rectilinear distance, as illustrated in Figure 3(b). This motivates the development of algorithms that are applicable to *arbitrary* weighted graphs (rather than to rectilinear graphs).

Figure 4 illustrates the routing solutions produced by the algorithms discussed below in Sections 3 and 4. The source for the four-pin net shown is the lightly-shaded square, and the sinks are the remaining solid squares. The two solutions on the left depict Steiner trees (sub-optimal in Figure 4(a), and optimal in

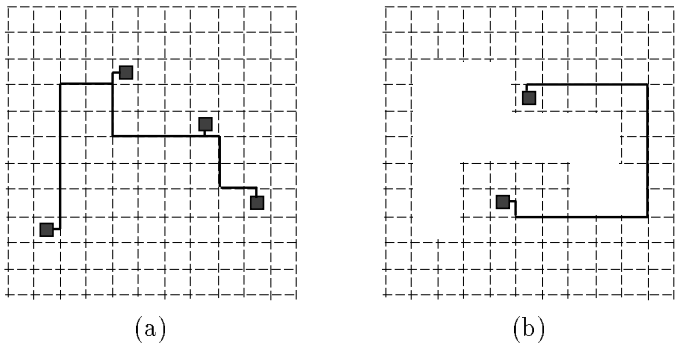

Figure 3: Initially, (a) shortest-path distances in the routing graph reflect rectilinear distance; (b) as nets are routed, paths may require detours, and distances no longer reflect the rectilinear metric.

Figure 4(b)), while the two solutions on the right are Steiner arborescences (sub-optimal in Figure 4(c), and optimal in Figure 4(d)). Figure 4(a) depicts the Steiner tree produced by the **KMB** heuristic of [26], while Figure 4(b) depicts the optimal solution produced by our new **IGMST** construction of Section 3 below. Figure 4(c) depicts the arborescence produced by **DJKA**, a variant of Dijkstra's algorithm [16], while Figure 4(d) depicts an optimal arborescence produced by our new **IDOM** construction of Section 4.2. Note that KMB uses 12.5% more wirelength than either of the solutions produced by IGMST or IDOM. Moreover, the maximum pathlength improvements of IGMST and IDOM over KMB in this example are 25% and 50%, respectively; the latter is particularly significant, since IDOM is seen to win over KMB in *both* wirelength and maximum pathlength (later we'll observe that this trend is manifested generally in our detailed experimental results in Section 5).

3 A New Graph Steiner Tree Heuristic

In routing non-critical nets we seek to minimize total wirelength, which motivates the graph Steiner tree (GMST) problem formulated in Section 2. A number of heuristics were proposed over the years for the GMST problem [22], two of which have performance bounds of a constant factor from optimal (in order to streamline the exposition, we defer the detailed description of these two methods to the Appendix):

- **KMB** – the heuristic of Kou, Markowsky and Berman [26] with a performance bound of $2 \cdot (1 - \frac{1}{L})$ where L is the maximum number of leaves in any optimal solution; and
- **ZEL** – the more recent heuristic of Zelikovsky [39] with performance bound of $\frac{11}{6}$ times optimal.

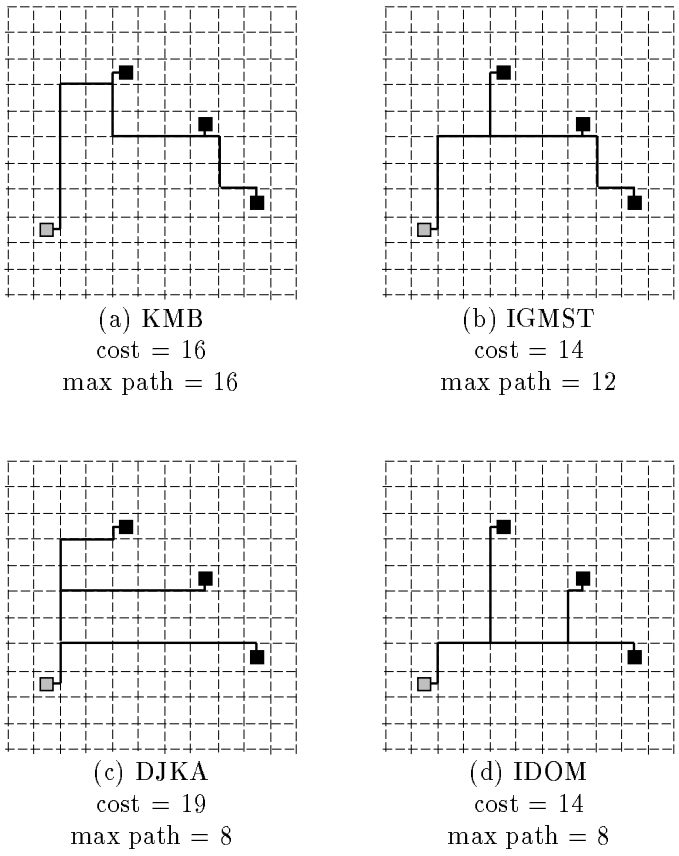

Figure 4: An example of four different routing solutions for the same four-pin net (the source is the lightly-shaded square, while the dark squares are sinks): (a) the solution produced by the KMB graph Steiner heuristic of [26]; (b) the optimal Steiner tree solution (which is also the solution produced by our IGMST algorithm described below); (c) depicts a sub-optimal Steiner arborescence produce by a variant of Dijkstra's algorithm; (d) shows the optimal Steiner arborescence solution (which is also the solution produced by our IDOM algorithm described in Section 4). In this example KMB uses 12.5% more wirelength than IGMST and IDOM, while the maximum pathlength improvements of IGMST and IDOM over KMB are 25% and 50%, respectively. Note that the IDOM solution in (d) is optimal in terms of both total wirelength *and* maximum pathlength.

We now propose a new class of *iterated* heuristics for the GMST problem. An instance of the GMST problem is $< G, N >$, where $G = (V, E)$ is a weighted graph, $N \subseteq V$ is a net, and the objective is to find a minimum-cost tree in G that spans N. For any existing graph Steiner tree heuristic H, let $H(G, N)$ denote the solution that H produces on input $< G, N >$, and let $cost(H(G, N))$ denote the cost of that solution.

Our basic algorithm template accepts as input an instance of the GMST problem and any existing GMST heuristic H. It then repeatedly finds Steiner node candidates that reduce the overall cost with

respect to H, and includes them into the growing set of Steiner nodes S.

Definition 3.1 *Given a set of Steiner candidate node $S \subseteq V - N$, we define the cost savings of S with respect to H as follows: $\Delta H(G, N, S) = cost(H(G, N)) - cost(H(G, N \cup S))$.*

Starting with an initially empty set of Steiner candidates $S = \emptyset$, our heuristic finds a node $t \in V - \{N \cup S\}$ which maximizes $\Delta H(G, N, S \cup \{t\}) > 0$ and repeats this procedure with $S \leftarrow S \cup \{t\}$. The cost for H to span $N \cup S$ will thus decrease with each added node t, and the construction terminates when there is no $t \in V - \{N \cup S\}$ such that $\Delta H(G, N, S \cup \{t\}) > 0$, with the final solution being $H(G, N \cup S)$. This method, which we call the Iterated Graph Minimal Steiner Tree (IGMST) approach, is formally described in Figure 5. Since any existing graph Steiner tree heuristic H may be used, the IGMST template represents an entire *class* of greedy iterated constructions, one corresponding to each possible H that may be used.

Iterated Graph Minimal Steiner Tree (IGMST) Algorithm.
Input: A weighted graph $G = (V, E)$, a net $N \subseteq V$, and a GMST heuristic H **Output:** A low-cost tree $T' = (V', E')$ spanning N, where $N \subseteq V' \subseteq V$ and $E' \subseteq E$
$S = \emptyset$ **Do Forever** $T = \{t \in V - N \mid \Delta H(G, N, S \cup \{t\}) > 0\}$ **If** $T = \emptyset$ **Then Return** $H(G, N \cup S)$ **Find** $t \in T$ with maximum $\Delta H(G, N, S \cup \{t\})$ $S = S \cup \{t\}$

Figure 5: The Iterated Graph Minimal Steiner Tree algorithm (IGMST) using a generic GMST heuristic H.

The performance bound of the IGMST method is clearly no worse than the performance bound of the heuristic H that it uses, since if no improving Steiner nodes can be found, the output of IGMST will be identical to the output of H (usually it is considerably better). For example, we may use the KMB heuristic [26] as H inside the IGMST algorithm to yield the Iterated KMB (IKMB) method, which inherits the performance bound of ≤ 2 times optimal. The KMB heuristic operates by constructing a minimum spanning tree on the *distance graph*, a new complete graph over N where edge weights correspond to shortest-path distances in $G = (V, E)$ (see the Appendix for further details on the KMB method). Figure 6 illustrates how the IKMB method greedily adds Steiner points to construct the solution. Note that IGMST generalizes the Iterated 1-Steiner heuristic of Kahng and Robins [21, 24, 25] where H is an ordinary rectilinear minimum spanning tree construction.

Experimental results in Section 5 indicate that iterating a heuristic H in this fashion yields significantly improved solutions as compared with the non-iterated version of H. Note that IGMST is not a "post-

processor" for the solution produced by heuristic H, but rather a technique for exploiting H in order to effectively navigate through the very large solution space of possible Steiner tree candidates.

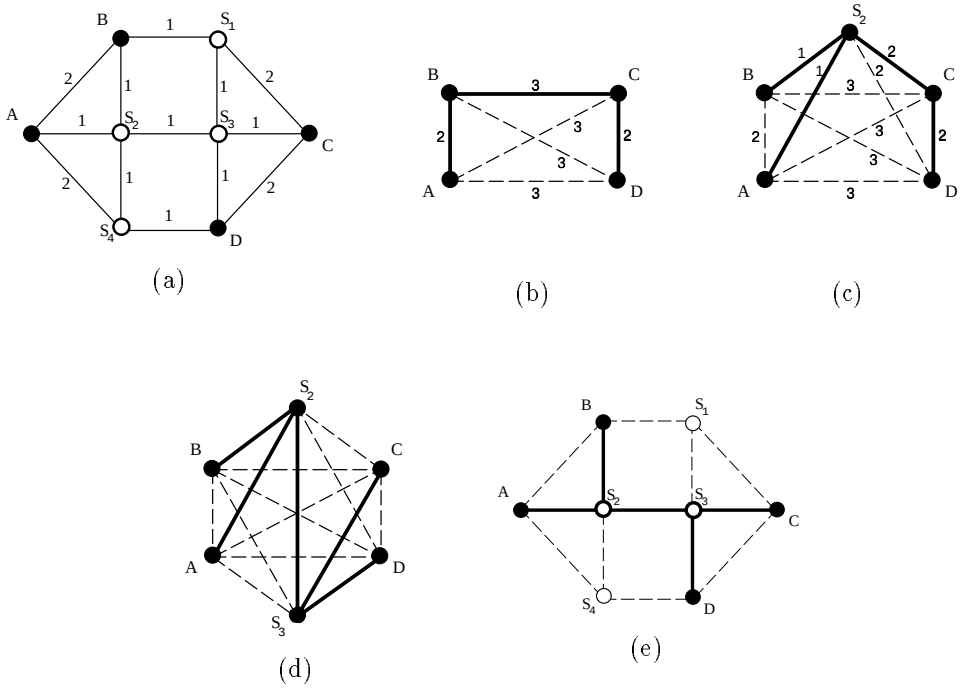

Figure 6: Example of the IKMB algorithm execution: (a) GMST problem instance with nodes to be spanned $N = \{A, B, C, D\}$; (b) initial KMB solution over the distance graph, having cost 7; (c) Steiner candidate S_2 produces savings ΔKMB= 1, and S_2 is thus retained as a Steiner point; (d) Steiner candidate S_3 is the final Steiner point with positive ΔKMB, reducing the overall tree cost to 5; (e) the final IKMB solution with cost 5, depicted with respect to the original graph (IKMB finds the optimal solution for this example).

The time complexity of the IGMST heuristic depends on the particular GMST heuristic H that is used. A naive implementation (which treats H as a "black box" subroutine) will have time complexity $O(|V| \cdot t(H))$ per iteration, where $t(H)$ is the time complexity of the given GMST heuristic. In practice, this general time complexity may be substantially reduced by factoring out of H common computations, such as computing shortest-paths, and thereby avoiding duplication of effort among multiple calls to H. Another way of reducing the time complexity follows from the observation that rather than adding Steiner points one at a time, they may be added in "batches" based on a *non-interference* criterion similar to the one proposed by Kahng and Robins [21, 24, 25]. In practice, the number of such rounds tends to be very small (≤ 3 for typical instances).

4 Two Graph Steiner Arborescence Heuristics

This section addresses the problem of constructing arborescences, or shortest-paths trees, where wirelength minimization is the secondary optimization objective.

4.1 The Path-Folding Arborescence (PFA) Heuristic

Constructing an arborescence can intuitively be viewed as "folding" (i.e., overlapping) paths in a shortest paths tree, in order to yield the greatest possible wirelength savings while still maintaining the shortest paths property. For pointsets in the Manhattan plane, a particularly effective arborescence heuristic along these lines is the RSA construction of [32], which has a performance ratio of ≤ 2 times optimal, as well as good empirical performance. However, this method relies on the underlying geometry of the Manhattan metric. In order to achieve the same effect in FPGA routing graphs, we first define the notion of dominance in arbitrary weighted graphs as follows.

Definition 4.1 *Given a weighted graph $G = (V, E)$, and nodes $\{n_0, p, s\} \subseteq V$, we say that p dominates s if* $\text{minpath}_G(n_0, p) = \text{minpath}_G(n_0, s) + \text{minpath}_G(s, p)$

In other words, a node p dominates a node s if there exists a shortest path from the source to p that also passes through s (recall that $minpath_G(u, v)$ denotes the *cost* of a shortest path between u and v in G).

Before we extend the RSA heuristic to graphs, we first review how it operates in the Manhattan plane (we assume that the source is located at the origin). A point p with coordinates (x_1, y_1) is said to *dominate* point s with coordinates (x_2, y_2) if $x_1 \geq x_2$ and $y_1 \geq y_2$, as shown in Figure 7(a). Define $MaxDom(p, q)$ to be the farthest point from the source that is dominated by both p and q (Figure 7(b)). The RSA construction iteratively replaces a pair of points $\{p, q\}$ with the single point $MaxDom(p, q)$, where $\{p, q\}$ are chosen as to maximize the distance from the source to $MaxDom(p, q)$. The algorithm terminates when only the origin remains, and the final Steiner arborescence solution is formed by connecting each produced $MaxDom(p, q)$ to both p and q.

Dominance in arbitrary weighted graphs is illustrated in Figure 8(a), where intuitively, p dominates s if a shortest path from the source n_0 to p can pass through s. Note that the shortest path between a pair of nodes in an FPGA graph is *not* necessarily unique. We define $MaxDom(p, q)$ as a node in V dominated by both p and q which maximizes $minpath_G(n_0, MaxDom(p, q))$, as shown in Figure 8(b). The reason we prefer that $MaxDom$ be as far from the origin as possible, is to maximize the overlap (i.e., the wirelength savings) among the two paths, while still maintaining the shortest-paths property of the construction.

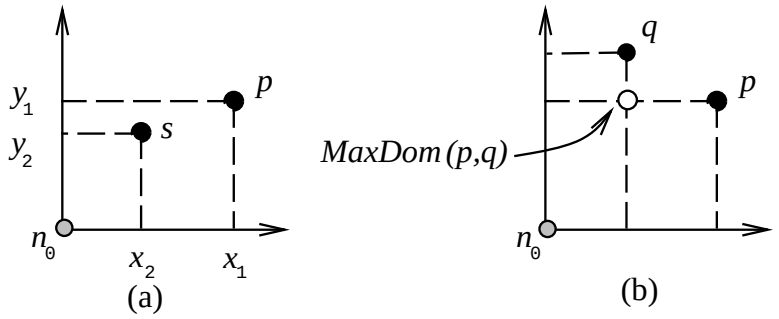

Figure 7: Example of rectilinear dominance: (a) p dominates s, and (b) the point $MaxDom(p, q)$, which is the farthest point from the source that is dominated by both q and p.

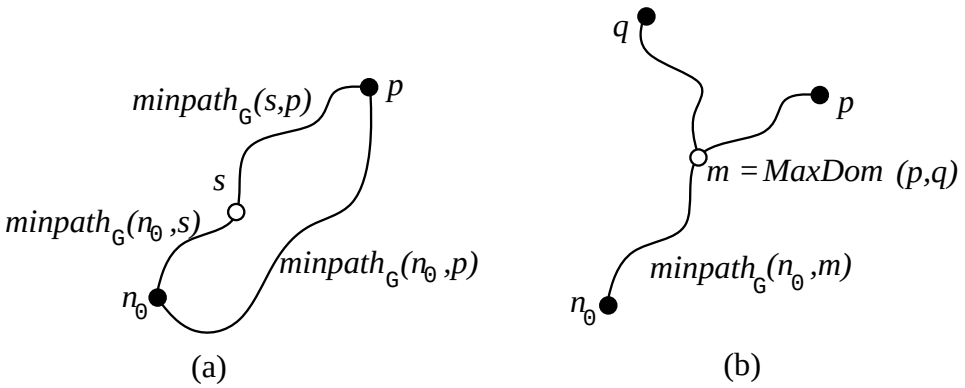

Figure 8: Example of dominance: (a) p dominates s when $minpath_G(n_0, p)$ $= minpath_G(n_0, s)$ + $minpath_G(s, p)$; (b) shows $MaxDom(p, q)$ with respect to p and q.

The above definitions enable the following *Path-Folding Arborescence* (PFA) heuristic: starting with the set of nodes that initially contains the source and all sinks, we find a pair of nodes p and q such that $m = MaxDom(p, q)$ is farthest away from the source among all such pairs; we then replace p and q by m and iterate until only the source remains. The overall graph Steiner arborescence solution is formed by using shortest paths in G to connect each $MaxDom(p, q)$ to both p and q (see Figure 9).

Since there are at most $O(|N|)$ elements in set N, the time to compute all shortest-paths trees is bounded by $O(|N| \cdot (|E| + V \log V))$, and the total number of *MaxDom* computations performed is at most $O(|V| \cdot |N|^2)$. The *MaxDom* value for each pair $\{p, q\} \subseteq N$ are maintained in a list ordered by decreasing *MaxDom* values, with the $\{p, q\}$ pair for each *MaxDom* being also stored in this list. This makes finding the next *MaxDom* relatively efficient, as it is the first element on the list with $\{p, q\} \in N$ (i.e., neither p nor q for this *MaxDom* have been removed from N). Using a heap to maintain the ordered list results in

an overall time complexity for PFA of $O(|N| \cdot |E| \ + \ |V| \cdot |N|^2 \cdot \log |V|)$.

Path-Folding Arborescence (PFA) algorithm
Input: Weighted graph $G = (V, E)$ and net $N \subseteq V$ **Output:** A low-cost shortest-paths tree spanning N

$M = N$
While $N \neq \{n_0\}$ **Do**
  Find a pair $\{p, q\} \subseteq N$ such that $m = MaxDom(p, q)$
  has maximum $minpath(n_0, m)$ over all $\{p, q\} \subseteq N$
  $N = \{N - \{p, q\}\} \cup \{m\}$
  $M = M \cup \{m\}$
Output the tree formed by connecting each node $p \in M$
  (using a shortest path in G) to the nearest node in M that p dominates

Figure 9: Path-Folding Arborescence (PFA) heuristic; M initially holds all the nodes to be spanned, and is then augmented with the *MaxDom* Steiner points found at each iteration.

The empirical results in Section 5 indicate that the PFA method is effective in producing shortest-paths trees with low wirelength (i.e. PFA's wirelength is on par with the best existing graph Steiner heuristics). However, in considering the worst-case behavior of PFA, we find examples of graphs where PFA can perform as badly as $O(N)$ times optimal (see Figure 10). Even on rectilinear grid-graphs, PFA may still produce solutions with cost approaching 2 times the optimal, so its performance ratio of 2 is tight [32] (see Figure 11). In the next section we therefore present another heuristic for the graph arborescence problem which escapes such worst-case examples (in fact, it *optimally* solves these particular worst-case examples).

4.2 The Iterated Dominance (IDOM) Heuristic

This section presents a new heuristic for the GSA problem which escapes the worst-case behavior of the PFA heuristic presented in Section 4.1. Our second GSA heuristic greedily iterates over a given *spanning* arborescence construction: it repeatedly finds Steiner candidates that reduce the overall spanning arborescence cost, and includes them into the growing set of Steiner nodes. The reason that we iterate a *spanning* arborescence construction in order to produce a *Steiner* arborescence construction is that the former is easy to compute while the latter is NP-complete. The heuristic which we use for producing spanning arborescences is the Dominating (DOM) heuristic, described as follows:

- **DOM** – This heuristic is a restricted version of the PFA heuristic of Figure 9, where $MaxDom(p, q)$ is constrained to be from N, rather than an arbitrary node from V. An arborescence is constructed by using a shortest path to connect each sink to the closest sink/source that it dominates, and then computing (Dijkstra's [16]) shortest paths tree over the graph formed by the union of these paths.

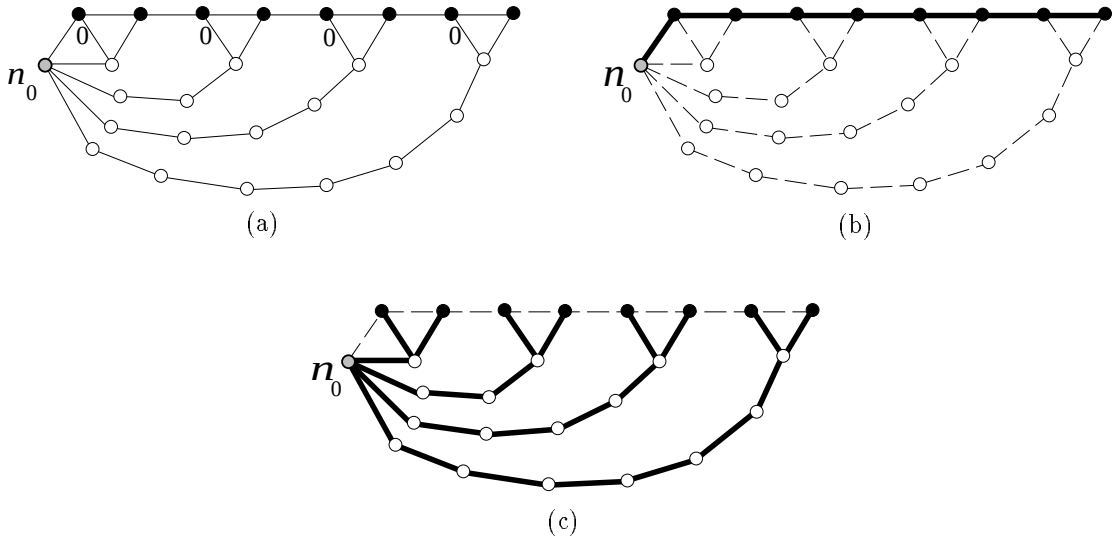

Figure 10: Worst-case example for the PFA on arbitrary weighted graphs. The source is the gray-shaded circle n_0, the sinks are darkened circles, potential Steiner points are white circles, and all edges have unit length except those marked as zero. (a) The input weighted graph; (b) the optimal solution, and (c) the worst-case behavior of the PFA heuristic. For an instance with $O(N)$ sinks, worst-case behavior in (c) is $O(N)$ times costlier than the optimal solution (b).

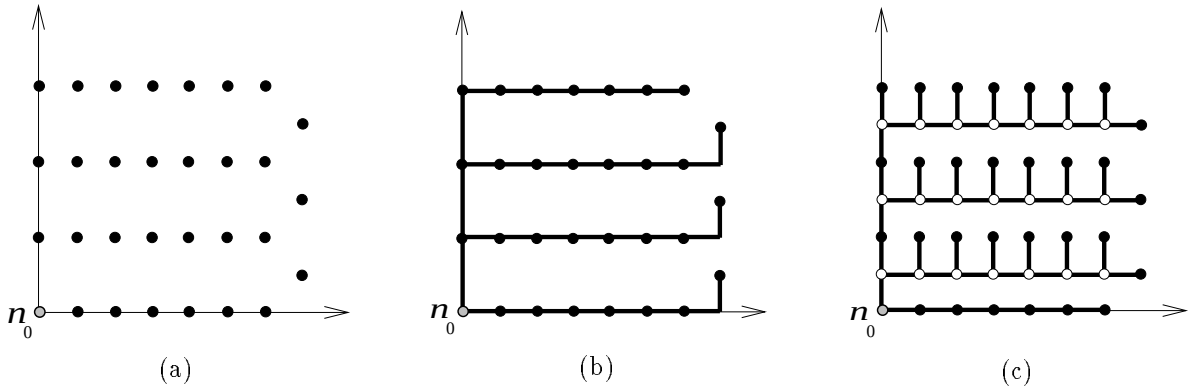

Figure 11: Worst-case example for PFA for a grid-graph (i.e., Manhattan plane geometry) [32]; (a) the source pin of this net / pointset is located at the origin, with horizontal interpoint distances being one unit, and vertical interpoint distances being two units; (b) shows the optimal solution; (c) depicts the worst-case behavior of the PFA heuristic (white circles denote Steiner points introduced by the PFA heuristic). The worst-case behavior in (c) is arbitrarily close to twice the optimal cost of (b).

Definition 4.2 *Given a set of Steiner candidate node $S \subseteq V - N$, we define the* cost savings *of S with respect to DOM as $\Delta DOM(G, N, S) = cost(DOM(G, N)) - cost(DOM(G, N \cup S))$.*

Starting with an initially empty set of Steiner candidates $S = \emptyset$, our heuristic finds a node $t \in V - N$ which maximizes $\Delta\text{DOM}(G, N, S \cup \{t\}) > 0$ and repeats this procedure with $S \leftarrow S \cup \{t\}$. The cost required by DOM to span $N \cup S$ will decrease with each added node t, and the overall construction terminates when there is no $t \in V - (N \cup S)$ such that $\Delta\text{DOM}(G, N, S \cup \{t\}) > 0$, with the overall solution being $\text{DOM}(G, N \cup S)$. This method, which we call the Iterated Dominance (IDOM) approach, is formally described in Figure 12.

Iterated Dominance (IDOM) Algorithm.
Input: A weighted graph $G = (V, E)$, a net $N \subseteq V$ **Output:** A low-cost arborescence $T' = (V', E')$ spanning N, where $N \subseteq V' \subseteq V$ and $E' \subseteq E$
$S = \emptyset$ **Do Forever** $T = \{t \in V - N \mid \Delta\text{DOM}(G, N, S \cup \{t\}) > 0\}$ **If** $T = \emptyset$ **Then Return** $\text{DOM}(G, N \cup S)$ **Find** $t \in T$ with maximum $\Delta\text{DOM}(G, N, S \cup \{t\})$ $S = S \cup \{t\}$

Figure 12: The Iterated Dominance (IDOM) algorithm.

The DOM subroutine can be implemented by constructing a shortest-paths tree with minimum cost on the distance graph (which is a new complete graph over N where edge weights correspond to shortest-path distances in $G = (V, E)$). Figure 13 illustrates how the IDOM method greedily adds Steiner points to construct a solution. The IDOM algorithm requires at most $O(|N| \cdot (|E| + |V| \log |V|))$ time to compute all shortest-paths trees and calls DOM as a subroutine $O(|V| \cdot |N|)$ times; each call constructs a minimum-cost shortest-paths tree over the (complete) distance graph on N and requires time $O(|N|^2)$. Thus, the IDOM heuristic can be easily implemented within time $O(|N| \cdot |E| \ + \ |N| \cdot |V| \cdot (\log |V| + |N|^2))$, and this time complexity can be further reduced by combining together common computations (as was discussed for the PFA heuristic in the previous section).

A worst-case example for IDOM is shown in Figure 14(b), which forces a performance bound on IDOM of logarithmic factor times optimal. However, this is not discouraging for the following reason. First, the NP-complete *Set Cover* problem can be reduced to the GSA problem (see Figure 14). Secondly, it is known that the Set Cover problem can not be approximated within a factor of better than $log_e n$ unless every problem in NP can be solved in deterministic time $O(n^{\log \log n})$ [17, 29]. This means that the GSA problem is also not likely to be polynomial-time approximable better than $O(\log N)$ times optimal, and thus worst-case examples such as those of Figure 14 are to be expected. We conjecture that IDOM has a performance ratio of $O(\log N)$.

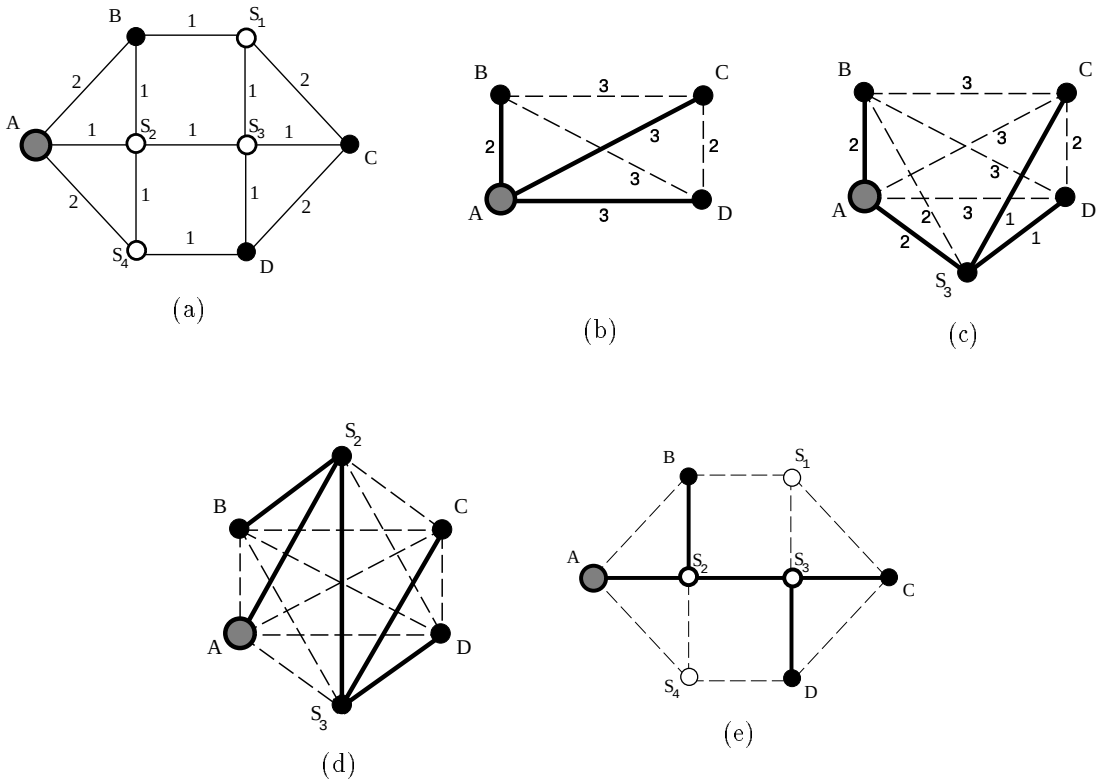

Figure 13: Execution example of the IDOM algorithm: (a) GSA problem instance with source node A (gray) and sink nodes $\{B, C, D\}$ (solid); (b) initial DOM solution over the distance graph, having cost 8; (c) Steiner candidate S_3 produces a savings of ΔDOM= 2, which reduces the overall tree cost to 6 – thus S_3 is retained as a Steiner point; (d) Steiner candidate S_2 is the final Steiner point with positive ΔDOM, and reduces the solution cost to 5; (e) the final IDOM solution (with cost 5) depicted with respect to the original input graph.

5 Experimental Results

We have implemented the IGMST, PFA and IDOM algorithms using C++ in the SUN Unix environment (our code and benchmarks are available upon request). We have also implemented the KMB and ZEL heuristics (see the Appendix for a detailed description of these), and used each of these as H inside the inner loop of IGMST, yielding the IKMB and IZEL constructions, respectively. For comparison, we have implemented two additional graph Steiner arborescence algorithms, a stand-alone version of the DOM heuristic described in Section 4.1, and also the following heuristic:

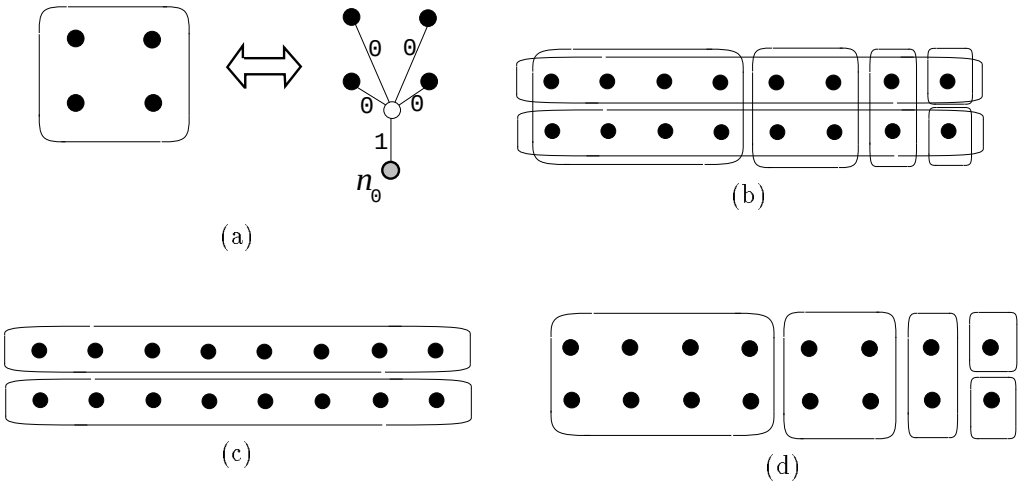

Figure 14: Worst-case example for IDOM on arbitrary weighted graphs. (a) A set of nodes enclosed by a box (left) represents a "macro" which expands to a set of edges with weight 0 connecting the sink nodes (solid circles) to a single Steiner node (hollow circle), and a single edge with weight 1 connecting the Steiner node to the source node, n_0 (right). Using this macro encoding, the input to IDOM is shown in (b). The optimal solution shown in (c) has cost of 2, consisting of the two long horizontal boxes, each containing $\frac{N}{2}$ of the nodes. On the other hand, IDOM can select a logarithmic number of boxes shown in (d) (the boxes from left to right, in order of exponentially decreasing size), resulting in an arborescence solution with cost $\Omega(\log N)$ times optimal.

- **DJKA** – This is an adaptation of Dijkstra's shortest-paths tree algorithm [16] to the GSA problem (Dijkstra's algorithm spans all of V, while the GSA problem seeks to span only $N \subseteq V$). DJKA first computes a shortest-paths tree rooted at the source using Dijkstra's algorithm [16], and then deletes edges from this tree which are not contained in any source-to-sink path.

We compared all of these methods (KMB, ZEL, IKMB, IZEL, DJKA, DOM, PFA, IDOM) on the same inputs, both in terms of total wirelength as well as maximum source-sink pathlength. The inputs consisted of random nets, uniformly distributed in 20×20 weighted grid graphs, where the edge weights represented congestion induced by previously-routed nets. Congestion was modeled as follows: starting with a grid graph having unit weights ($w = 1.00$) on all edges, k uniformly-distributed nets (2-5 pins each) were routed using KMB. As each net was routed, the weights of the corresponding graph edges were incremented, thus raising the average routing-graph edge weight to $\bar{w} > 1.00$. Three different levels of congestion were thus modeled: (a) none ($k = 0, \bar{w} = 1.00$), (b) low ($k = 10, \bar{w} = 1.28$), and (c) medium ($k = 20, \bar{w} = 1.55$).

For each of these three congestion levels and net size (5 and 8 pins), 50 uniformly-distributed nets were routed on a congested graph (newly-generated for each net), using all eight algorithms. For each net,

we normalized the wirelength produced by each heuristic with respect to the wirelength used by KMB; similarly, the maximum source-sink pathlength of each heuristic was normalized to optimal. Table 1 gives the average percent improvement for each congestion level, where a positive value represents an increase (i.e., disimprovement) in the total wirelength (resp. maximum pathlength) with respect to KMB (resp. optimal), while a negative number represents a decrease (i.e., improvement).

Average Wirelength and Maximum Pathlength % For various congestion levels, over 50 nets				
	5-pin nets		**8-pin nets**	
Algorithm	Wire Length (w.r.t. KMB)	Max Path (w.r.t. OPT)	Wire Length (w.r.t. KMB)	Max Path (w.r.t. OPT)
No Congestion (no pre-routed nets) Average routing graph edge weight $\bar{w} = 1.00$				
KMB	0.00	23.51	0.00	40.30
ZEL	-6.22	11.07	-7.85	23.42
IKMB	**-6.47**	**10.83**	**-8.19**	**24.04**
IZEL	**-6.79**	**8.85**	**-8.31**	**21.47**
DJKA	29.23	0.00	30.53	0.00
DOM	17.51	0.00	18.48	0.00
PFA	**-5.59**	**0.00**	**-5.02**	**0.00**
IDOM	**-5.59**	**0.00**	**-4.89**	**0.00**
Low Congestion ($k = 10$ pre-routed nets) Average routing graph edge weight $\bar{w} = 1.28$				
KMB	0.00	27.61	0.00	47.66
ZEL	-4.64	19.14	-4.10	34.17
IKMB	**-5.68**	**17.12**	**-4.50**	**33.35**
IZEL	**-5.98**	**14.56**	**-5.52**	**22.29**
DJKA	26.64	0.00	32.48	0.00
DOM	22.27	0.00	28.09	0.00
PFA	**8.95**	**0.00**	**13.91**	**0.00**
IDOM	**8.95**	**0.00**	**13.91**	**0.00**
Medium Congestion ($k = 20$ pre-routed nets) Average routing graph edge weight $\bar{w} = 1.55$				
KMB	0.00	30.67	0.00	52.67
ZEL	-4.37	21.54	-3.35	44.95
IKMB	**-5.09**	**17.77**	**-4.42**	**42.42**
IZEL	**-5.57**	**15.26**	**-4.97**	**40.20**
DJKA	22.94	0.00	36.79	0.00
DOM	21.78	0.00	33.89	0.00
PFA	**13.93**	**0.00**	**22.65**	**0.00**
IDOM	**13.93**	**0.00**	**22.59**	**0.00**

Table 1: The average wirelength % (normalized w.r.t. KMB) and average maximum pathlength (normalized w.r.t. optimal) for the various algorithms, run over grid graphs with three different levels of congestion.

Among the four Steiner heuristics (KMB, ZEL, IKMB, IZEL), IZEL has superior performance. The ranking IZEL<IKMB<ZEL<KMB is very consistent across all net sizes in terms of both wirelength *and* maximum pathlength, indicating that our iterated constructions outperform the stand-alone, non-iterated versions. Among the four arborescence constructions (DJKA, DOM, PFA, IDOM), PFA and IDOM consistently use the least wirelength (these all yield optimal maximum pathlength). Here too, the ranking is quite consistent in terms of wirelength across all net sizes, namely IDOM<PFA<DOM<DJKA. Thus in practice, both PFA and IDOM perform quite well, despite the worst-case examples of Figures 10, 11 and 14.

On uncongested graphs, both PFA and IDOM outperform KMB in term of wirelength by up to 5.6%. This is interesting since KMB minimizes wirelength *only*, yet it uses more wirelength than either PFA and IDOM, which yield optimal pathlengths and only optimize wirelength as a *secondary* criterion. For uncongested graphs, both PFA and IDOM yield optimal maximum pathlength at almost no wirelength penalty over IZEL; thus, these seem to afford favorable tradeoffs between wirelength and maximum pathlength. Note that IKMB and the Iterated 1-Steiner heuristic of Kahng and Robins [21, 24, 25] yield identical solutions for geometric instances (i.e., when using the Hanan grid as the underlying graph). CPU times for IKMB, PFA and IDOM on random graphs with $|V| = 50|$, $|E| = 1000$ and $|N| = 5$ are in the range of several dozen milliseconds on a Sun/4 workstation.

To further validate the effectiveness of our algorithms for FPGA routing, we built an actual FPGA router based on these constructions, and used it to route fourteen industrial FPGA benchmark circuits, containing up to 608 nets each (see Tables 2 and 3 for details regarding these benchmarks). Our graph-based constructions easily adapt to a variety of PFGA architectures. In particular, we have modeled two distinct FPGA architectures, the first corresponding to Xilinx 3000-series parts [38] (Table 2), and the second corresponding to 4000-series parts [38] (Table 3); these architectures are identical to those used by the CGE router [12], and the SEGA [27] and GBP [37] routers, respectively.

The *switch block flexibility* (denoted by F_s) reflects the pre-specified fanout of a channel edge inside a switch block, i.e., the number of different channel edges to which it may be connected [12]. Similarly, the *connection flexibility* (denoted by F_c) refers to the number of adjacent channel edges to which a logic-block pin may connect; finally, W denotes the channel width. The 3000-series FPGAs which are used to route the circuits in Table 2 have $F_s = 6$ and $F_c = \lceil 0.60 \cdot W \rceil$, while the 4000-series FPGAs in Table 3 have $F_s = 4$ and $F_c = W$.

As discussed in Section 2, our FPGA router uses a graph-based framework, where the topology of the routing graph reflects the underlying FPGA architecture (See Figure 2). Edge weights in this graph reflect wirelength, as well as the congestion induced by previously-routed nets. Our router operates directly on this graph and routes the nets one at a time. After the routing of each net, the edge weights are updated to reflect the new congestion values; edges used to route the net are removed from the graph, so that subsequent nets remain electrically disjoint from previously routed ones. We employ a net ordering scheme with a move-to-front heuristic: when infeasibility is encountered in routing a particular net, that net will be routed earlier in subsequent routing phases, thereby increasing the probability of a successful routing of all the nets. We found that typically only a few (i.e., less than five) such passes are required to completely route each of the benchmark circuits.

A common criteria used to evaluate the quality of FPGA routers is the maximum channel width required to successfully route all nets of a design [12]. This is important for designs which span multiple FPGAs since a router which requires smaller maximum channel width has the ability to pack larger portions of the overall circuit design onto a single FPGA, which tends to reduce the total number of FPGAs required. In our router, maximum channel width serves as an upper-bound input parameter when routing a circuit. The router attempts to route the circuit using this specified channel width, and if a complete routing solution cannot be found in a user-specified maximum number of passes (we arbitrarily set this feasibility threshold to 20 passes), the router decides that the circuit is unroutable at that given channel width. Thus, for each circuit we find the smallest maximum channel width necessary to completely route the circuit. CPU times to completely route the industry benchmark circuits on a Sun/4 workstation varied from several minutes for the smallest circuit to several hours for the largest; these times are quite competitive with commercial routers, which sometimes require several days to route large circuits [35].

For each of the circuits, we compared the maximum channel width required by our router using the IKMB algorithm to the best reported results from CGE [12] using the 3000-series architecture (See Table 2), as well as to the best reported values for SEGA [27] and GBP [37] using the 4000-series architecture (See Table 3). Note that for *both* the 3000-series and 4000-series architectures we are able to route all of the benchmark circuits using significantly smaller channel width than CGE, SEGA and GBP (with these other routers requiring an average of 22%, 26%, and 17% more channel width, respectively, than our own router).

3000-series Circuits		Breakdown of nets by number of pins				Maximum channel width required for a **complete** routing of each circuit	
Name	FPGA size	#nets	#2-3	#4-10	#over 10	CGE	**Our Router**
busc	12×13	151	115	28	8	10	**7**
dma	16×18	213	139	52	22	10	**9**
bnre	21×22	352	255	70	27	12	**9**
dfsm	22×23	420	361	26	33	10	**9**
z03	26×27	608	398	176	34	13	**11**
Totals:		1744	1268	352	124	55	**45**
Ratios:		1.00	0.73	0.20	0.07	1.22	**1.00**

Table 2: Results for a set of industry FPGA benchmark circuits, all routed on a Xilinx 3000-type part, with switch-block flexibility $F_s = 6$ and connection flexibility $F_c = \lceil 0.6 \cdot W \rceil$, where W is the channel width. On the left are the total number of nets in each circuit, the FPGA size, and the number of nets with 2-3 pins, 4-10 pins, and over 10 pins in each circuit. On the right are the maximum channel widths required to route all nets using the CGE router of [12], as well as our own router using the IKMB algorithm. For each of the benchmark circuits our router requires a significantly smaller channel width than CGE; on average, CGE requires 22% more channel width than our router.

The data in Tables 2 and 3 indicates that our IKMB algorithm effectively minimizes channel width. Reduced channel widths are a result of routing multi-pin nets as complete units, rather than breaking

4000-series Circuits		Breakdown of nets by number of pins				Maximum channel width required for a **complete** routing of each circuit		
Name	FPGA size	#nets	#2-3	#4-10	#over 10	SEGA	GBP	**Our Router**
alu4	19×17	255	165	69	21	15	14	**11**
apex7	12×10	115	83	30	2	13	11	**10**
term1	10×9	88	65	21	2	10	10	**8**
example2	14×12	205	171	25	9	17	13	**11**
too_large	14×14	186	128	46	12	12	12	**10**
k2	22×20	404	241	146	17	17	17	**15**
vda	17×16	225	132	80	13	13	13	**12**
9symml	11×10	79	60	11	8	10	9	**8**
alu2	15×13	153	109	26	18	11	11	**9**
Totals:		1710	1154	454	102	118	110	**94**
Ratios:		1.00	0.67	0.27	0.06	1.26	1.17	**1.00**

Table 3: Results for a set of industry FPGA benchmark circuits, all routed on a Xilinx 4000-type part, with switch-block flexibility $F_s = 3$ and connection flexibility $F_c = W$, where W is the channel width. On the left are the total number of nets in each circuit, the FPGA size, and the number of nets of each pin count. On the right are the maximum channel width required to route all nets for the SEGA router of [27], the GBP router of [37], and our own router using the IKMB algorithm. For each of the benchmarks our router requires a significantly smaller channel width than either SEGA or GBP; on average, SEGA and GBP require 26% and 17% more channel width than our router, respectively.

them into multiple two-pin nets (as is done by other routers). First, our constructions use Steiner points to effectively reduce overall wirelength. Routing resources which are saved by such reductions are used to route subsequent nets, which tends to improve resource utilization and reduce channel width. Second, our routing-graph construction is designed so that wirelength minimization *directly* reduces channel width: Steiner points which are placed adjacent to logic-block pins simultaneously reduce both channel width and wirelength (See Figure 15). Recall that the IKMB algorithm minimizes *only* wirelength, while the PFA and IDOM algorithms minimize both maximum pathlength *and* wirelength. To illustrate how minimizing maximum pathlength affects wirelength (and thus improve the maximum channel width), Table 4 shows the maximum channel width required for a successful routing using the IKMB, PFA and IDOM algorithms for each of the circuits (for comparison, the channel widths required by the SEGA [27] and GBP [37] routers are shown also). As expected, both PFA and IDOM require larger channel width than IKMB. However, neither PFA nor IDOM require larger channel width than SEGA or GBP, routers which do *not* directly minimize maximum source-to-sink pathlength. This illustrates the ability of the PFA and IDOM algorithms to simultaneously minimize wirelength and maximum pathlength in an effectively manner.

Table 5 shows the increase in wirelength vs. the decrease in maximum pathlength for the IKMB, PFA and IDOM algorithms on the benchmark circuits. Here the algorithms operate on FPGAs with the *same* channel width (i.e., the smallest channel width that results in a successful routing for all algorithms). By comparing the various algorithms using the same channel width, the wirelength usage is not unduly biased

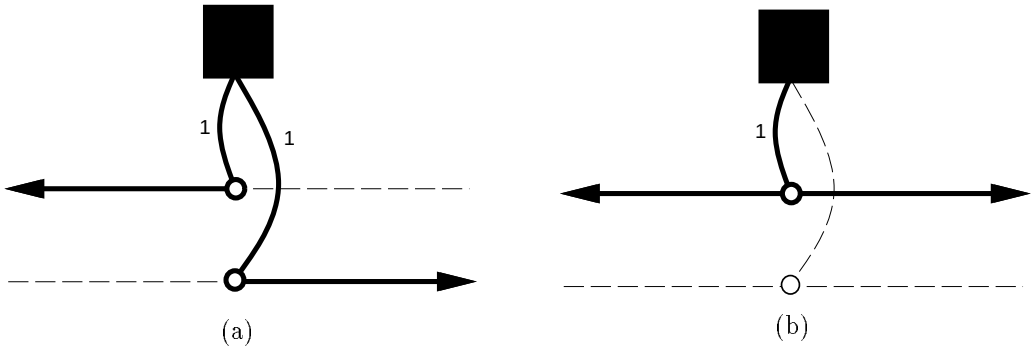

Figure 15: An example that illustrates how routing multi-pin nets as a single unit during wirelength minimization can also enhance channel width minimization. (a) a solution requiring channel width of two; (b) Steiner-based wirelength minimization reduces the required channel width to one.

Xilinx 4000-Series Circuits	Maximum required channel width for a **complete** routing				
	Other Routers		**Our Router**		
Name	SEGA	GPB	IKMB	PFA	IDOM
alu4	15	14	11	14	13
apex7	13	11	10	11	11
term1	10	10	8	9	9
example2	17	13	11	13	13
too_large	12	12	10	12	12
k2	17	17	15	17	17
vda	13	13	12	14	13
9symml	10	9	8	9	8
alu2	11	11	9	11	10
Totals:	118	110	94	110	106
Ratios:	**1.26**	**1.17**	**1.00**	**1.17**	**1.13**

Table 4: Maximum channel width required for a successful routing of each entire benchmark circuit using our IKMB, PFA and IDOM algorithms. Note that both PFA and IDOM minimize wirelength *and* maximum pathlength, while IKMB, SEGA [27] and GBP [37] minimize only wirelength.

by the more circuitous routes which may be required with small channel widths. We observe that the increase in wirelength for PFA and IDOM (18.21% and 12.79%, respectively) corresponds quite closely to the increase in channel width observed for PFA and IDOM in Table 4. Table 5 indicates that both the PFA and IDOM algorithms effectively reduce the maximum pathlength (by 9.51% and 10.19% on average, respectively, as compared to IKMB). Figure 16 illustrates the solution produced by our router for the bus-controller benchmark circuit (busc).

Xilinx 4000 Circuits	Channel Width	Wirelength		Max Path	
		PFA	IDOM	PFA	IDOM
alu4	14	20.9	15.8	-15.2	-16.9
apex7	11	15.3	9.2	-4.2	-6.8
term1	9	11.4	12.0	-6.2	-2.0
example2	13	13.1	8.1	-4.6	-5.6
too_large	12	17.9	15.2	-9.7	-9.4
k2	17	24.5	17.6	-7.1	-7.2
vda	14	18.7	11.9	-9.9	-11.5
9symml	9	18.3	11.4	-14.0	-14.4
alu2	11	23.9	14.1	-14.7	-18.0
Averages:		**18.2**	**12.8**	**-9.5**	**-10.2**

Table 5: Percent increase in wirelength and decrease in maximum pathlength for PFA and IDOM (with respect to IKMB) on the benchmark circuits. We observe that both the PFA and IDOM algorithms significantly reduce the maximum source-sink pathlength (by 9.5% and 10.2% on average, respectively).

6 Conclusion

We have developed new performance-driven FPGA routing algorithms. For routing non-critical nets, we minimize wirelength using an effective new class of iterative graph Steiner tree constructions. For critical-net routing, we presented two new arborescence heuristics that produce shortest-paths trees with low overall wirelength. Our methods afford routing trees with optimal source-sink pathlengths, using total wirelength on par with the best existing graph Steiner tree heuristics (which is rather surprising). Our FPGA router based on these algorithms effectively routes industrial benchmarks using reduced maximum channel width, and our experimental results are currently the best known in the literature. Our routing algorithms easily integrate into existing layout frameworks to yield combined place-and route tools [3, 5]. Moreover, all of our methods generalize to three-dimensional FPGAs [1, 2]. Future research directions include proving a non-trivial performance bound on the IDOM heuristic, further improving the time complexities of the various constructions, and finding a graph Steiner heuristic with performance ratio $< \frac{11}{6}$.

7 Acknowledgments

We would like to thank the anonymous referees for their diligence and thoughtful feedback. Andrew Kahng, Chuck Alpert, and Darren Chi provided valuable advice on this work. We thank Jonathan Rose and Stephen Brown for the use of their benchmark circuits. We are indebted to Dr. Bob Grafton of the National Science Foundation, as well as to the Packard Foundation for their generous support. Our benchmarks and other related works may be found at WWW URL http://www.cs.virginia.edu/~robins/.

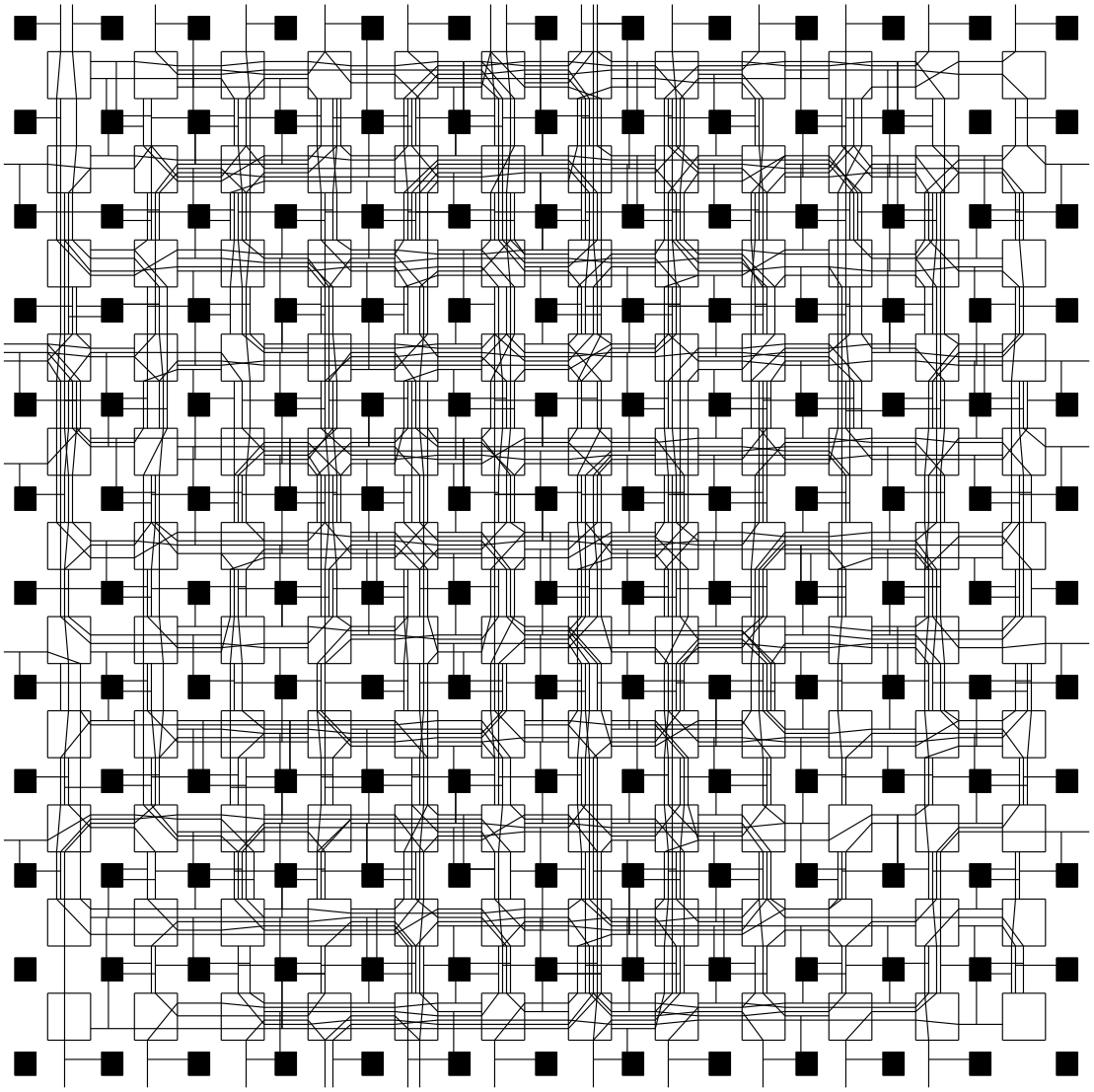

Figure 16: Solution produced by our router for the busc circuit.

References

[1] M. J. Alexander, J. P. Cohoon, J. L. Coleflesh, J. Karro, E. L. Peters, and G. Robins, *Placement and Routing for Three-Dimensional FPGSs*, in Proc. ACM/SIGDA Physical Design Workshop, Reston, VA, April 1996, pp. 142–149.

[2] M. J. Alexander, J. P. Cohoon, J. L. Coleflesh, J. Karro, and G. Robins, *Three-Dimensional Field-Programmable Gate Arrays*, in Proc. IEEE Intl. ASIC Conf., Austin, TX, September 1995, pp. 253–256.

[3] M. J. Alexander, J. P. Cohoon, J. L. Ganley, and G. Robins, *Placement and Routing for Performance-Oriented FPGA Layout*, VLSI Design (to appear).

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

[5] M. J. Alexander, J. P. Cohoon, J. L. Ganley, and G. Robins, *Performance-Oriented Placement and Routing for Field-Programmable Gate Arrays*, in Proc. European Design Automation Conf., Brighton, England, September 1995, pp. 80–85.

[6] M. J. Alexander and G. Robins, *High-Performance Routing for Field-Programmable Gate Arrays*, in Proc. IEEE Intl. ASIC Conf., Rochester, NY, September 1994, pp. 138–141.

[7] M. J. Alexander and G. Robins, *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.

[8] M. J. Alexander and G. Robins, *New Performance-Driven FPGA Routing Algorithms*, in Proc. ACM/IEEE Design Automation Conf., San Francisco, CA, June 1995, pp. 562–567.

[9] C. J. Alpert, T. C. Hu, J. H. Huang, A. B. Kahng, and D. Karger, *Prim-Dijkstra Tradeoffs for Improved Performance-Driven Routing Tree Design*, IEEE Trans. Computer-Aided Design, 14 (1995), pp. 890–896.

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

[11] K. D. Boese, A. B. Kahng, B. A. McCoy, and G. Robins, *Near-Optimal Critical Sink Routing Tree Constructions*, IEEE Trans. Computer-Aided Design, 14 (1995), pp. 1417–1436.

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

[13] K. C. Chen, J. Cong, Y. Ding, A. B. Kahng, and P. Trajmar, *DAG-Map: Graph-Based FPGA Technology Mapping for Delay Optimization*, IEEE Design & Test of Computers, 9 (1992), pp. 7–20.

[14] J. Cong, A. B. Kahng, G. Robins, M. Sarrafzadeh, and C. K. Wong, *Provably Good Performance-Driven Global Routing*, IEEE Trans. Computer-Aided Design, 11 (1992), pp. 739–752.

[15] J. Cong, K. S. Leung, and D. Zhou, *Performance-Driven Interconnect Design Based on Distributed RC Delay Model*, in Proc. ACM/IEEE Design Automation Conf., Dallas, June 1993, pp. 606–611.

[16] E. W. Dijkstra, *A Note on Two Problems in Connection With Graphs*, Numerische Mathematik, 1 (1959), pp. 269–271.

[17] U. Feige, *A Threshold of ln n for Approximating Set Cover*, in Proc. ACM Symp. the Theory of Computing, May 1996, pp. 314–318.

[18] J. Frankle, *Iterative and Adaptive Slack Allocation for Performance-driven Layout and FPGA Routing*, in Proc. ACM/IEEE Design Automation Conf., 1992, pp. 536–542.

[19] J. L. Ganley, private communication, April, 1994.

[20] T. Gao, K. C. Chen, J. Cong, Y. Ding, and C. L. Liu, *Placement and Placement Driven Technology Mapping for FPGA Synthesis*, in Proc. IEEE Intl. ASIC Conf., Rochester, NY, September 1993, pp. 87–91.

[21] J. Griffith, G. Robins, J. S. Salowe, and T. Zhang, *Closing the Gap: Near-Optimal Steiner Trees in Polynomial Time*, IEEE Trans. Computer-Aided Design, 13 (1994), pp. 1351–1365.

[22] F. K. Hwang, D. S. Richards, and P. Winter, *The Steiner Tree Problem*, North-Holland, 1992.

[23] M. A. B. Jackson, E. S. Kuh, and M. Marek-Sadowska, *Timing-Driven Routing for Building Block Layout*, in Proc. IEEE Intl. Symp. Circuits and Systems, 1987, pp. 518–519.

[24] A. B. Kahng and G. Robins, *A New Class of Iterative Steiner Tree Heuristics With Good Performance*, IEEE Trans. Computer-Aided Design, 11 (1992), pp. 893–902.

[25] A. B. Kahng and G. Robins, *On Optimal Interconnections for VLSI*, Kluwer Academic Publishers, Boston, MA, 1995.

[26] L. Kou, G. Markowsky, and L. Berman, *A Fast Algorithm for Steiner Trees*, Acta Informatica, 15 (1981), pp. 141–145.

[27] G. G. Lemieux and S. D. Brown, *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.

[28] F. D. Lewis and W. C. Pong, *A Negative Reinforcement Method of PGA Routing*, in Proc. ACM/IEEE Design Automation Conf., 1993, pp. 601–605.

[29] C. Lund and M. Yannakakis, *On the Hardness of Approximating Minimization Problems*, Proc. ACM Symp. the Theory of Computing, 25 (1993), pp. 286–293.

[30] K. Mehlorn, *A Faster Approximation Algorithm for the Steiner Problem in Graphs*, Information Processing Letters, 27 (1988), pp. 125–128.

[31] S. Prasitjutrakul and W. J. Kubitz, *A Timing-Driven Global Router for Custom Chip Design*, in Proc. IEEE Intl. Conf. Computer-Aided Design, Santa Clara, CA, November 1990, pp. 48–51.

[32] S. K. Rao, P. Sadayappan, F. K. Hwang, and P. W. Shor, *The Rectilinear Steiner Arborescence Problem*, Algorithmica, (1992), pp. 277–288.

[33] K. Roy, B. Guan, and C. Sechen, *FPGA MCM Partitioning and Placement*, in Proc. ACM/SIGDA Physical Design Workshop, Lake Arrowhead, CA, April 1993, pp. 211–212.

[34] Y. Sun, T. C. Wang, C. K. Wong, and C. L. Liu, *Routing for Symmetric FPGAs and FPICs*, in Proc. IEEE Intl. Conf. Computer-Aided Design, Santa Clara, CA, November 1993, pp. 486–490.

[35] S. Trimberger. Manager of Advanced Development, Xilinx Inc., private communication, February, 1994.

[36] S. M. Trimberger, *Field-Programmable Gate Array Technology, S. M. Trimberger, editor*, Kluwer Academic Publishers, Boston, MA, 1994.

[37] Y.-L. Wu and M. Marek-Sadowska, *An Efficient Router for 2-D Field Programmable Gate Arrays*, in European Design and Test Conf., 1994, pp. 412–416.

[38] Xilinx, *The Programmable Gate Array Data Book*, Xilinx, Inc., San Jose, California, 1994.

[39] A. Z. Zelikovsky, *An 11/6 Approximation Algorithm for the Network Steiner Problem*, Algorithmica, 9 (1993), pp. 463–470.

[40] A. Z. Zelikovsky, *A Faster Approximation Algorithm for the Steiner Tree Problem in Graphs*, Information Processing Letters, 46 (1993), pp. 79–83.

8 Appendix: Two Existing Graph Steiner Heuristics

This appendix describes the existing graph Steiner heuristics of (1) Kou, Markowsky, and Berman (KMB) [26], and (2) Zelikovsky (ZEL) [39]; the later is currently the best known Graph Steiner heuristics, with respect to theoretical performance bounds and run time. KMB has a theoretical performance ratio[1] of $2 \cdot (1 - \frac{1}{L})$, where L is the maximum number of leaves in any optimal (Steiner tree) solution to the input instance. The ZEL heuristic has performance ratio of $\frac{11}{6}$, and thus on worst-case instances ZEL will outperform KMB.[2]

Any graph Steiner heuristic H may be used inside the IGMST template of Section 3 to yield an improved approximation algorithm for the IGMST problem. The theoretical performance bound of the composite construction is guaranteed to be no worse than that of the heuristic H that was used in the iterated construction. Thus, the performance bound of the IKMB algorithm is $2 \cdot (1 - \frac{1}{L})$, where L is the maximum number of leaves in any optimal solution, while the performance bound of IZEL is $\frac{11}{6}$.

All Steiner tree heuristics aim to approximately solve the following NP-complete problem (see Section 2 for motivation and details):

The Graph Minimal Steiner Tree (GMST) Problem: Given a weighted graph $G = (V, E)$, and a net $N \subseteq V$, find a spanning tree $T = (V', E')$ with $N \subseteq V' \subseteq V$ and $E' \subset E$ such that $cost(T)$ is minimum.

8.1 The KMB Heuristic [26]

The graph Steiner tree heuristic of Kou, Markowsky and Berman (KMB) [26] is described as follows (see Figure 17):

- Construct the *distance graph* G' over N as follows: form the complete graph G' over N with the weight of each edge e_{ij} equal to $dist_G(n_i, n_j)$, i.e., the cost of the corresponding shortest path in G between n_i and n_j.
- Compute MST(G'), the minimum spanning tree of G', and expand each edge e_{ij} of MST(G') into the corresponding shortest path, denoted $path(n_i, n_j)$, yielding a subgraph G'' that spans N.
- Finally, compute the minimum spanning tree MST(G''), and delete pendant edges from MST(G'') until all leaves are members of N.

The time complexity of the KMB heuristic in Figure 17 is $O(|N| \cdot |V|^2)$, and this can be reduced to $O(|E| + V\log|V|)$ using an alternative implementation [30].

[1] The *performance ratio* of a heuristic is the worst-case ratio of the cost of its solutions with respect to optimal.

[2] Note that in a practical setting, although KMB can produce solutions with cost arbitrarily close to twice the optimal, it typically produces solutions that are much better than twice optimal (e.g., on inputs where L is small). Thus it is not clear a priori from these general theoretical bounds that ZEL will always win over KMB on typical nets (although the empirical data in Section 5 indicates that this is indeed the case *on average*).

The Kou, Markowsky and Berman (KMB) Algorithm [26]
Input: A graph $G = (V, E)$ with edge weights w_{ij} and a net $N \subseteq V$ **Output:** A low-cost tree $T' = (V', E')$ spanning N (i.e. $N \subseteq V' \subseteq V$ and $E' \subseteq E$)
$G' = (N, N \times N)$, with edge weights $w'_{ij} = dist_G(n_i, n_j)$ **Compute** $T = (N, E'') = \text{MST}(G')$ $G'' = \cup_{e_{ij} \in E''} path_G(n_i, n_j)$ **Compute** $T' = \text{MST}(G'')$ **Delete** pendant edges from T' until all leaf nodes that are not in N **Output** T'

Figure 17: The KMB heuristic for the GMST problem [26].

8.2 The ZEL Heuristic [39]

The graph Steiner tree heuristic of Zelikovsky (ZEL) [39] is described as follows (see Figure 18). Define a *triple* to be a set of three nodes in N; we can *contract* a graph around a triple z by setting to zero the edge weights of two of the three edges connecting nodes of the triple (the contracted graph is denoted by $G'[z]$). Now perform the following steps:

- Construct the *distance graph* G' over N as follows: form the complete graph G' over N with the weight of each edge e_{ij} equal to $dist_G(n_i, n_j)$, i.e., the cost of the corresponding shortest path in G between n_i and n_j.

- For every triple $z \in N$, find $v_z \in V$ which minimizes $dist_z = \sum_{s \in z} dist_G(s, v)$; (i.e., the Steiner point which will produce the greatest savings for each triple).

- Find $z \in N$ which maximizes $0 \leq win = MST(G') - MST(G'[z]) - dist_z$, and contract G' around z to get a new $G' \leftarrow G'[z]$. The Steiner point associated with the contracting triple, v_z, becomes a Steiner point in the solution. Repeat this greedy contraction step while $win \geq 0$.

- Construct a solution using the KMB algorithm where the nodes to be spanned are the original nodes N plus the v_z's associated with the $G'[z]$ contractions above.

The time complexity of the ZEL heuristic is $O(|N| \cdot (|E| + |V| \cdot |N| + |V| \cdot \log |V|))$ [40].

The Zelikovsky (ZEL) Algorithm [39]

Input: A graph $G = (V, E)$ with edge weights w_{ij} and a net $N \subseteq V$
Output: A low-cost tree $T' = (V', E')$ spanning N (i.e. $N \subseteq V' \subseteq V$ and $E' \subseteq E$)

$G' = (N, N \times N)$, with edge weights $w'_{ij} = dist_G(n_i, n_j)$,
$W = \emptyset$, Triples $= \{z \subset N : |z| = 3\}$
For every $z \in$ Triples **Do**
    Find v which maximizes $\sum_{s \in z} dist_G(s, v)$.
    $v_z = v$ and $dist_z = \sum_{s \in z} dist_G(s, v)$
Repeat Forever
    Find $z \in$ Triples which maximizes $win =$ MST$(G')-$ MST$(G'[z]) - dist_z$
    If $win \leq 0$ **Then Return** KMB$_G(N \cup W)$
    $G' = G'[z]$
    $W = W \cup v_z$

Figure 18: The ZEL heuristic for the GMST problem [39].

Biographies

Gabriel Robins received his Ph.D. degree in 1992 from the UCLA Computer Science Department, where he won a Distinguished Teaching Award and held an IBM Graduate Fellowship. Dr. Robins then joined the Department of Computer Science at the University of Virginia in Charlottesville, where he received an NSF Young Investigator Award, a Packard Foundation Fellowship, a Lilly Foundation Teaching Fellowship, an All-University Outstanding Teacher Award, and a two-year early promotion to Associate Professor of Computer Science in 1996. His primary areas of research are VLSI CAD and geometric algorithms, with recent work focusing on performance-driven routing, pattern recognition, and computational biology. Dr. Robins is a member of the Defense Science Study Group, an advisory board to the U.S. Department of Defense. He also serves on the Navy Future Study Panel of the National Academy of Sciences. Dr. Robins has co-authored a Distinguished Paper at the 1990 IEEE International Conference on Computer-Aided Design, as well as a book on high-performance routing for Kluwer Academic Publishers. Dr. Robins was General Chair of the 1996 ACM/SIGDA Physical Design Workshop, and he also serves on the technical program committees of several other leading conferences. He is a member of ACM, IEEE, MAA, and SIAM.

Michael Alexander received his Ph.D. degree in 1996 from the Department of Computer Science at the University of Virginia in Charlottesville, where he won a Teaching Assistant Extraordinaire award. He is now Assistant Professor in the School of Electrical Engineering and Computer Science at Washington State University. His research interests include VLSI physical design, with emphasis on high-performance routing and FPGA architectures. Dr. Alexander has served on the program committees of the IEEE International Symposium on Circuits and Systems, the Canadian Workshop on Field-Programmable Devices, the ACM/SIGDA Physical Design Workshop, and the ACM/SIGDA International Symposium on Physical Design. He is a member of Tau Beta Pi, ACM, and IEEE.