New Performance-Driven FPGA Routing Algorithms*

Michael J. Alexander and Gabriel Robins

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 effective Steiner and arborescence FPGA routing algorithms. Our graph-based Steiner tree constructions have provably-good performance bounds and outperform the best known ones in practice, while our arborescence heuristics produce routing solutions with optimal source-sink pathlengths at a reasonably low wirelength penalty. We have incorporated our algorithms into an actual FPGA router which routed a number of industrial circuits using channel widths considerably smaller than was previously possible.

1 Introduction

Field-Programmable Gate Arrays (FPGAs) are flexible and reusable high-density circuits that can be (re)configured by the designer, enabling the VLSI design/validation/simulation cycle to be performed more quickly and cheaply [19]. The flexibility provided by FPGAs incurs a substantial performance penalty due to signal delay through the programmable routing resources, and this is currently a primary concern of FPGA designers and users [16]. In order to increase FPGA performance, partitioning and technology mapping have been used to minimize the length of critical paths [3]. On the other hand, less attention has been focused on the actual routing, which is surprising since circuit performance is limited by routing delays, rather than by combinational logic delays [11].

Routing affects the performance of FPGA-based systems in two major ways. First, a typical design must be partitioned and mapped onto several FPGAs. Because FPGA size is fixed, the ability to pack larger partitions onto a single FPGA can reduce the total number of partitions (and hence FPGAs) required to implement the 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 which minimize use of routing resources.

Second, since FPGA resource utilization typically does not exceed 80%, considerable flexibility remains onboard the FPGA for optimizing the routing. For example, we could reduce signal propagation delay through critical paths by using the most direct interconnections (i.e., shortest paths), where a secondary criterion is to minimize wirelength in order to reduce 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 class of algorithms for non-critical-net routing which can outperform the best known graph Steiner tree heuristics, i.e., those of Kou, Markowsky and Berman [12], and of Zelikovsky [20]. Our graph Steiner construction is based on an iterative template that uses any given Steiner tree heuristic H by greedily selecting Steiner nodes that induce maximum wirelength savings with respect to H. The theoretical performance bound is guaranteed to be no worse than that of H, and in practice the construction will tend to outperform H.

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 shortest-possible, and where total wirelength is optimized as a secondary objective. Our first graph Steiner arborescence heuristic is based on a *path-folding* strategy that overlaps and merges shortest paths in order to reduce the overall wirelength. Our second heuristic iteratively selects Steiner nodes which improve the total wirelength.

We have incorporated our algorithms into an actual FPGA router and successfully routed industry benchmark circuits using considerably smaller channel width than previous routers. Our routing benchmarks are currently the best among all published results. The total wirelength used by our arborescence constructions in unit-weighted grid graphs is on par with the best graph Steiner tree heuristics. This is interesting, since our arborescence solutions have optimal source-sink pathlengths, while Steiner tree heuristics are designed to only optimize wirelength.

2 Problem Formulation

An FPGA architecture consists of a set of user-configurable logic "blocks", and a set of programmable interconnection resources used for routing [3, 16] (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 technology mapping, partitioning, and placement have already been performed.

Previous work on FPGA routing concentrated on solution feasibility and resource-usage minimization. For example, the CGE [3] and SEGA [13] routers handle nets based on demand and assign critical nets a higher routing priority. Other papers studied FPGA routing

*G. Robins is supported by NSF Young Investigator Award MIP-9457412. Corresponding author is G. Robins, Department of Computer Science, Thornton Hall, University of Virginia, Charlottesville, VA 22903-2442, USA, Email: robins@cs.virginia.edu, phone: (804) 982-2207, FAX: (804) 982-2214, WWW URL http://uvacs.cs.virginia.edu/~robins/

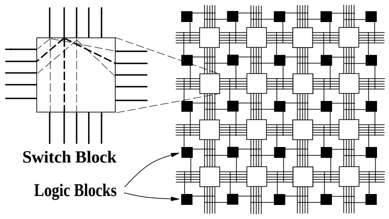

Figure 1: A symmetrical-array FPGA.

with switch blocks of limited flexibility [18], explored modified architectures [15], or computed lower bounds on routing rather than providing actual routes [4]. Recently [1, 2] developed a routing framework where mutually competing objectives (e.g., congestion, wirelength, jog minimization) are simultaneously optimized. However, these works do not directly minimize source-sink signal propagation delays, and while many approaches implicitly equate delay minimization with wirelength optimization, these two goals are not synonymous [11].

In order to apply graph-based techniques, we 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 vice versa (Figure 2). Let $G = (V, E)$ denote such a graph, where each graph edge $e_{ij} \in E$ has a weight w_{ij} that corresponds to the wirelength of the associated FPGA routing wire segment (weights may also reflect 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 its edge weights.

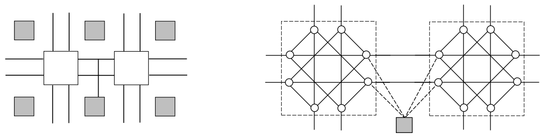

Figure 2: An FPGA routing graph model.

Prior to routing, nets may be classified as either *critical* or *non-critical* based on timing information that becomes available during iterative layout. When routing non-critical nets, we seek not to optimize delay but rather to maximize the likelihood of completely routing all nets on the given FPGA; this resource-usage minimization objective motivates the following graph Steiner minimal tree formulation:

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

Any node in $V - N$ may be used as a potential Steiner point in order to optimize the overall wirelength. The GSMT problem is known to be NP-complete and arises in numerous applications [9]. 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:

Graph Steiner Arborescence (GSA) Problem: Given weighted graph $G = (V, E)$, and 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 $E' \subseteq E$ 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 [7], and FPGA routing graphs are generally large, we must seek efficient heuristics for this problem. On the other hand, we can prove the following non-approximability result:

Theorem 2.1 *The GSA problem can not be approximated in polynomial time to within a performance bound of better than $O(\log N)$ times optimal (unless deterministic polylog space coincides with non-deterministic polylog space, which is a longstanding open problem).*

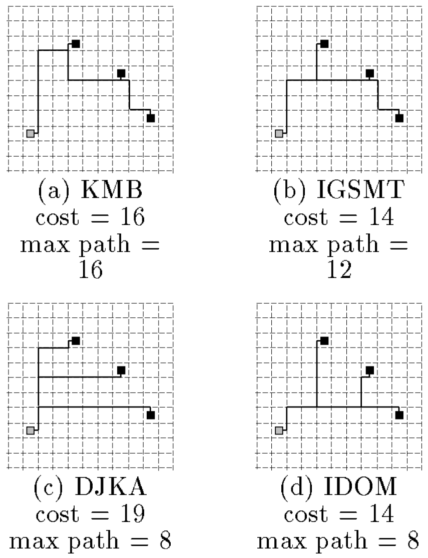

Figure 3: Four routing constructions.

Figure 3 illustrates routing solutions produced by the algorithms discussed below (the source is the lightly-shaded square, and the dark squares are sinks). The two solutions at the top depict Steiner trees, while the two lower constructions are Steiner arborescences: (a) the **KMB** heuristic of [12], (b) our **IGSMT** solution (which is also the optimal Steiner tree here), (c) **DJKA**, a variant of Dijkstra's algorithm [6], (d) our **IDOM** construction (which is also the optimal arborescence here).

3 A Graph Steiner Tree Heuristic

A number of heuristics were proposed over the years for the GSMT problem [9], two of which have performance bounds of a constant factor from optimal:

- **KMB** – the heuristic of Kou, Markowsky and Berman [12] with a performance bound of 2 times optimal; and
- **ZEL** – the more recent heuristic of Zelikovsky [20] with performance bound of $\frac{11}{6}$ times optimal.

We propose a method of *iterating* heuristics for the GSMT problem. Recall that an instance of the GSMT problem is a weighted graph $G = (V, E)$ and a net $N \subseteq V$, with the objective being finding 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, and let $cost(H(G, N))$ denote the cost of that solution. Our algorithm accepts as input an instance of the GSMT problem and any existing GSMT heuristic H. It then repeatedly finds Steiner node candidates that reduce the overall spanning 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 nodes $S \subseteq V - N$, the* cost savings *of S with respect to H is: $\Delta H(G, N, S) = cost(H(G, N)) - cost(H(G, N \cup S))$.*

Starting with an initially empty set of Steiner nodes $S = \emptyset$, our heuristic finds a node $t \in V - N$ 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 decrease with each added node t, and the construction terminates when there is no $t \in (V - N) - S$ such that $\Delta H(G, N, S \cup \{t\}) > 0$. The output solution is $H(G, N \cup S)$. This general template, which we call the Iterated Graph Steiner Minimal Tree (IGSMT) approach, is formally described in Figure 4.

```
Iterated Graph Steiner Minimal Tree Algorithm
Input: A weighted graph $G = (V, E)$, a net $N \subseteq V$,
       and a GSMT 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 4: The IGSMT algorithm.

The performance bound of the IGSMT method is no worse than that of the heuristic H, since if no improving Steiner nodes can be found, the output of IGSMT will be identical to the output of H. For example, we may use the ZEL heuristic [20] as H inside the IGSMT template to yield the Iterated ZEL (IZEL) method, which inherits ZEL's performance bound of $\leq \frac{11}{6}$ times optimal. Note that IGSMT generalizes the Iterated 1-Steiner heuristic of Kahng and Robins [10] (where H is an ordinary rectilinear minimum spanning tree construction), as well as the algorithms of [1, 2] (where H is the KMB heuristic). Our experimental results indicate that iterating a heuristic H in this fashion yields significantly improved solutions as compared with the non-iterated version of H.

The time complexity of IGSMT depends on the particular heuristic H that is used. A naive implementation (which treats H as a "black box" subroutine) will have time complexity $O(|N| \cdot |V| \cdot t(H))$, where $t(H)$ is the time complexity of H. This time complexity may be substantially improved by (1) extracting out of H common computations (e.g., computing shortest-paths), to avoiding duplication of effort among multiple calls to H, and by (2) adding Steiner points in "batches" based on a non-interference criterion [8, 10].

4 Path-Folding Heuristic

Constructing an arborescence intuitively entails "folding" (i.e., overlapping) paths in a shortest-paths tree to yield the greatest possible wirelength savings while maintaining the shortest-paths property. For pointsets in the Manhattan plane, an effective arborescence heuristic is the construction of Rao et al. [14], which has a performance ratio of twice optimal, as well as good empirical performance (a variation of the method of [14] was given in [5]). However, these methods rely on the underlying geometry of the Manhattan metric. In order to handle 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 $minpath_G(n_0, p) = minpath_G(n_0, s) + minpath_G(s, p)$*

Thus, 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 generally not unique. We define $MaxDom(p, q)$ as a node $m \in V$ dominated by both p and q which maximizes $minpath_G(n_0, m)$. Selecting a $MaxDom$ as far away from the origin as possible maximizes the overlap (i.e., the wirelength savings) between the two paths.

These definitions enable our *Path-Folding Arborescence* (PFA) heuristic which generalizes the method of [14] to arbitrary weighted graphs. Starting with the set of nodes containing the source and all sinks, find a pair of nodes p and q such that $m = MaxDom(p, q)$ is farthest away from the source among all such pairs; then replace p and q by m and iterate until only the source remains. The graph Steiner arborescence solution is formed by using shortest paths in G to connect each $MaxDom(p, q)$ to both p and q (Figure 5).

```
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 $minpath_G(n_0, m)$ 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 5: Path-Folding Arborescence (PFA) heuristic.

Since there are at most $O(|N|)$ elements in set N, the time to compute all shortest-paths trees is bounded by $O(|N| \cdot |E|)$, and the total number of $MaxDom$ computations performed is at most $O(|V| \cdot |N|^2)$. Storing the results of the $MaxDom$ computations in a heap allows the next $MaxDom$ to be determined efficiently, and results in an overall time complexity for PFA of $O(|N| \cdot |E| + |V| \cdot |N|^2 \cdot \log |V|)$.

Our empirical results indicate that the PFA method is effective in producing shortest-paths trees at a relatively modest wirelength penalty. However, in considering the worst-case behavior of PFA, we found examples of graphs where PFA can perform as badly as $O(N)$ times optimal. Thus, the next section presents another heuristic for the graph arborescence problem which escapes such worst-case examples.

5 Iterated Dominance Heuristic

Our second heuristic for the GSA problem greedily iterates over a given *spanning* arborescence construction: we repeatedly find Steiner candidates that reduce the overall spanning arborescence cost, and include them into the growing set of Steiner nodes. The heuristic which we use for producing spanning arborescences is the Dominating (DOM) heuristic, described as follows:

- **DOM** – connect each sink to the closest sink or source that it dominates, and compute the shortest-paths tree over the union of these paths.

Definition 5.1 *Given a set of Steiner candidate node $S \subseteq V - N$, we define the* cost savings *of S with respect to DOM as* $\Delta\text{DOM}(G, N, S) = cost(\text{DOM}(G, N)) - cost(\text{DOM}(G, N \cup S))$.

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 for DOM to span $N \cup S$ will decrease with each added node t, and the construction terminates when there is no $t \in (V - N) - S$ such that $\Delta\text{DOM}(G, N, S \cup \{t\}) > 0$, with the final solution being $\text{DOM}(G, N \cup S)$. This Iterated Dominance (IDOM) approach is formally described in Figure 6. The IDOM heuristic can be implemented within time $O(|N| \cdot |E| + |V| \cdot |N|^3)$.

```
Iterated Dominance (IDOM) Algorithm.
Input: A weighted graph $G = (V,E)$, a net $N \subseteq V$
Output: 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 6: The Iterated Dominance algorithm.

6 Experimental Results

We have implemented the IGSMT, PFA and IDOM algorithms using C++ in the SUN Unix environment. Our code is available upon request, and all of our benchmarks and routing solutions are available on the World Wide Web at URL http://uvacs.cs.virginia.edu/~robins/. We have also implemented KMB and ZEL, and used each of these as H inside the inner loop of IGSMT, yielding the IKMB and IZEL constructions, respectively. For comparison, we have implemented DOM, as well as the following adaptation of Dijkstra's shortest-paths tree algorithm [6] to the GSA problem:

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.

- **DJKA** – compute Dijkstra's shortest-paths tree rooted at the source, and then delete edges that are not contained in any source-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 uniformly distributed random nets in 20×20 weighted grid graphs, where the edge weights modeled congestion induced by previously-routed nets. Congestion was created 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, resulting in a higher average routing-graph edge weight $\bar{w} > 1.00$. 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 (6 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)

Xilinx 3000-Series Circuits			Maximum required channel width for a **complete** routing	
Name	FPGA size	#nets	CGE	**Ours**
busc	12×13	151	10	**7**
dma	16×18	213	10	**9**
bnre	21×22	352	12	**9**
dfsm	22×23	420	10	**9**
z03	26×27	608	13	**11**
Totals:		1744	55	**45**
Ratios:			1.22	**1.00**

Table 2: Complete routing of benchmark circuits on a Xilinx 3000-type part, with switch-block flexibility of 6 and 60% connectivity on the channel edges.

Xilinx 4000-Series Circuits			Maximum required channel width for a **complete** routing		
Name	FPGA size	#nets	SEGA	GPB	**Ours**
alu4	19×17	255	15	14	**11**
apex7	12×10	115	13	11	**10**
term1	10×9	88	10	10	**8**
example2	14×12	205	17	13	**11**
too_large	14×14	186	12	12	**10**
k2	22×20	404	17	17	**15**
vda	17×16	225	13	13	**12**
9symml	11×10	79	10	9	**8**
alu2	15×13	153	11	11	**9**
Totals:		1710	118	110	**94**
Ratios:			1.26	1.17	**1.00**

Table 3: Complete routing of benchmark circuits on a Xilinx 3000-type part, with switch-block flexibility of 3 and 100% connectivity on the channel edges.

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 using the various algorithms.

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.

with respect to KMB (resp. optimal), while a negative number represents a decrease (i.e., improvement).

Among the four Steiner heuristics (KMB, ZEL, IKMB, IZEL), IZEL has superior performance. The ranking IZEL<IKMB<ZEL<KMB is highly 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.

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 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 Iterated 1-Steiner [10] yield identical solutions for geometric instances (when using the Hanan grid as the underlying graph).

We built an actual FPGA router based on these algorithms, and *completely*[1] routed 14 pre-placed industrial benchmark circuits, containing up to 608 nets each. Our constructions easily adapt to a variety of architectures; in particular, we modeled two distinct FPGA architectures, the first corresponding to Xilinx 3000-series parts [19] (Table 2), and the second corresponding to 4000-series parts [19] (Table 3) - these architectures are identical to those used by the CGE router [3], and the SEGA [13] and GPB [18] routers, respectively. The 3000-series FPGAs used to route the circuits in Table 2 have switch-block flexibility of 6 and 60% channel-edge connectivity, while the 4000-series FPGAs in Table 3 have switch-block flexibility of 3 and 100% channel-edge connectivity. We did not alter the fixed benchmark placements. CPU times to completely route the circuits on a Sun SparcServer 10/514 workstation varied from several minutes for the smallest circuit to several hours for the largest.

We route the nets one at a time, updating the routing-graph edge weights as we proceed to reflect congestion. 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 phases, thereby increasing the probability of a successful routing of all nets. Only a few such passes are required to completely route each benchmark.

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 [3] using the 3000-series architecture (Table 2), as well as to the best reported values for SEGA [13] and GPB [18] using the 4000-series architecture (Table 3). For both types of architectures we are able to route all of the benchmark circuits using significantly smaller channel width than CGE, SEGA and GPB (with these other routers requiring an average of 22%, 26%, and 17% more channel width, respectively, than our router).

[1] Incomplete routes or global routing only are not useful in practice, since there can be an arbitrarily large increase in channel width in obtaining complete detailed routes from these [17].

To illustrate how minimizing maximum pathlength affects wirelength (and thus 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. As expected, both PFA and IDOM require larger channel width than IKMB. However, neither PFA nor IDOM require larger channel width than SEGA or GPB (which do not directly minimize maximum source-sink pathlength). Thus, PFA and IDOM simultaneously minimize wirelength and maximum pathlength quite effectively.

Table 5 shows the average increase in wirelength vs. the decrease in maximum pathlength for IKMB, PFA and IDOM 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). The increase in wirelength for PFA and IDOM (18.2% and 12.8%, respectively) corresponds to the increase in channel width observed in Table 4. Both PFA and IDOM effectively reduce the maximum pathlength (by 9.5% and 10.2% on average, respectively). Figure 7 illustrates our router's solution for the smallest 4000-series benchmark circuit.

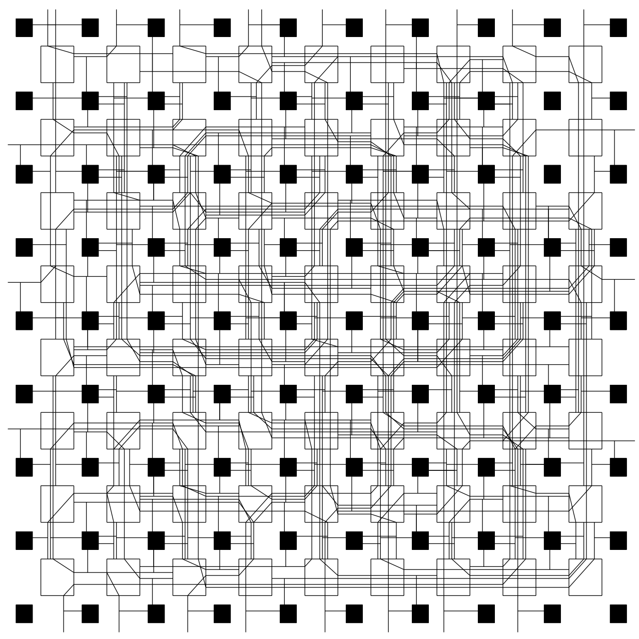
Figure 7: IKMB solution for the term1 circuit.

7 Acknowledgments

We are grateful to Dr. Bob Grafton of NSF for his support and encouragement. We thank Jonathan Rose and Stephen Brown for their benchmark circuits.

References

[1] 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.

[2] 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.

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

[4] Y.-W. Chang, S. Thakur, K. Zhu, and D. F. Wong, *A New Global Routing Algorithm for FPGAs*, in Proc. IEEE Intl. Conf. Computer-Aided Design, San Jose, CA, November 1994.

[5] 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.

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

[7] J. L. Ganley, *private communication*, April, 1994.

[8] 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.

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

[10] 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.

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

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

[13] 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.

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

[15] 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.

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

[17] Y.-L. Wu and D. Chang, *On the NP-Completeness of Regular 2-D FPGA Routing Architectures and a Novel Solution*, in Proc. IEEE Intl. Conf. Computer-Aided Design, San Jose, CA, November 1994, pp. 362–366.

[18] 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.

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

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