Non-Tree Routing

Bernard A. McCoy and Gabriel Robins

Technical Report No. CS-93-16
April 12, 1993

Non-Tree Routing

Bernard A. McCoy and Gabriel Robins

Computer Science Department, University of Virginia, Charlottesville, VA 22903-2442

Abstract

An implicit premise of existing routing methods is that the routing topology must correspond to a tree (i.e., it does not contain cycles). In this paper we abandon this basic axiom and investigate the consequences of allowing routing topologies that correspond to arbitrary graphs (i.e., where cycles are allowed). We show that adding extra wires to an existing routing tree can often significantly improve signal propagation delay by exploiting a tradeoff between wire capacitance and resistance, and we propose several new routing algorithms based on this phenomenon. Using SPICE to determine the efficacy of our methods, we obtain dramatic results: for example, the addition of a single new wire to an existing minimum spanning tree (MST) routing reduces the average signal propagation delay by up to 24%, while the average interconnection cost increases by only 11%, depending on net size. The delay performance of our methods is competitive with the best existing routing tree constructions, while the average wirelength of our constructions is significantly better. Our basic formulation extends to several important routing regimes, including the Steiner case, critical-sink routing, and wire sizing.

1 Introduction

Recent advances in VLSI technology have steadily improved chip packing densities. As feature sizes decrease, device switching speeds tend to increase; however, thinner wires have higher resistance, causing signal propagation delay through the interconnect to increase [18]. Thus, interconnection delay has had a greater impact on circuit speed, being responsible for up to 70% of the clock cycle in the design of dense, high-performance circuits [20]. In light of this trend, performance-driven physical layout has become central to the design of leading-edge digital systems. Early work focused on performance-driven placement, with the usual objective being the close placement of cells in timing-critical paths [9] [14] [15].

While timing-driven placement has a large effect on layout performance, the lack of optimal-delay interconnection algorithms impedes designers in fully exploiting a high-quality placement. Once a module placement has been fixed, good timing-driven interconnection algorithms are key to enhancing the performance of the layout solution. For a given signal net, the typical objective has been to minimize the maximum signal delay from the source pin to any sink pin. Many approaches have appeared in the literature, e.g., Dunlop et al. [10] determine net priorities based on static timing analysis, and process higher priority nets earlier, using fewer feedthroughs; Jackson et al. [12] outline a hierarchical approach to timing-driven routing; and Prastjutrakul and Kubitz [17] use A* heuristic search and the Elmore delay formula [11] in their tree optimization; Cohoon and Randall [7] developed a critical net routing algorithm in order to reduce interconnect delay.

Cong et al. have proposed finding minimum spanning trees with bounded source-sink pathlength [8] by simultaneously minimizing both tree cost and the tree radius; another cost-radius tradeoff was achieved by Alpert et al. [1]. Boese et al. [5] have developed a "critical sink" routing approach which significantly reduces delay to specified sinks, thereby exploiting the critical-path information that is implicitly available during iterative timing-driven layout. Recently, Boese et al. [4] have identified and exploited a high-quality, algorithmically tractable model of interconnect delay, based on an upper bound [19] for Elmore delay.

An implicit premise of previous methods is that a routing topology must correspond to a tree (i.e., an acyclic topology). In retrospect, this seems a natural assumption, since a tree topology spans a net, thus achieving connectivity using a minimum number of edges. In this paper, we question this seemingly basic axiom, and investigate the consequences of removing the acyclic restriction. Thus, we formulate a routing problem where the interconnection topology may correspond to an arbitrary graph.

At this point, the reader may question the wisdom of adding extra wires to an existing routing tree: how can this possibly improve signal propagation delay? The answer lies in the tradeoff between the capacitance and resistance in a circuit. Clearly, adding extra wires to a routing tree increases the overall routing capacitance; however, the extra wires may significantly lower certain source-sink resistance values. It is possible for this decrease in resistance to more than compensate for the associated increase in capacitance, as can be seen in a simple example (Figure 1). Similarly, Figure 2 gives an example of a random net where adding a single extra edge to the minimum spanning tree creates a substantial delay improvement.

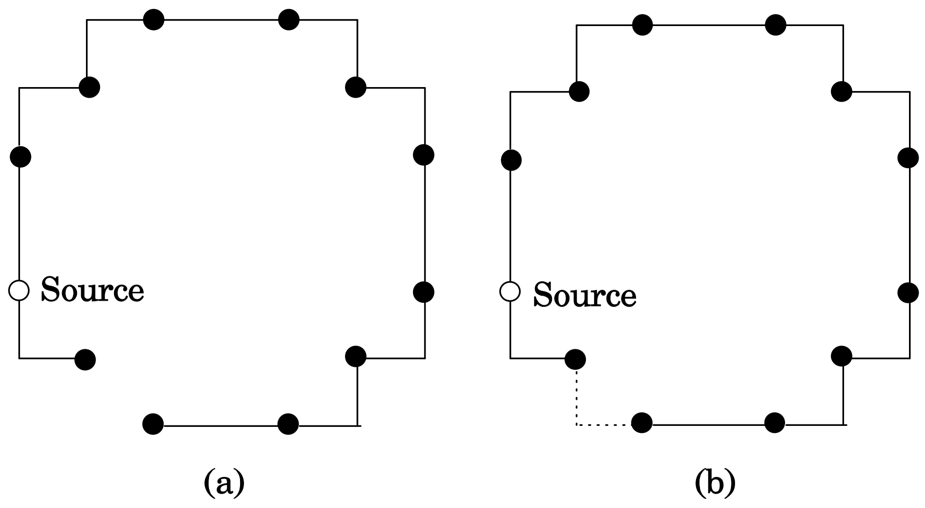

Figure 1: An example of how adding an extra edge to the minimum spanning tree on the left (a) can yield the routing topology with reduced interconnection delay on the right (b); in this example, routing topology (a) has maximum source-sink SPICE delay of 1.3 nanoseconds, while the topology on the right has a SPICE delay of 1.0 nanoseconds, representing a 23% delay improvement (at a total wirelength penalty of 9%). The interconnect parameters used are representative of a 0.8μ CMOS process (see discussion below for details).

Since we are highly concerned with obtaining realistic results, we use the SPICE circuit simulator [16] to determine the efficacy of our methods (in Section 2 we discuss in detail the SPICE parameters used). Our results are both surprising and dramatic: for example, even adding a single wire/edge to an existing minimum spanning tree routing reduces the average signal propagation delay by up to 24%, while the average interconnection cost increases by only 11%, depending on net size. The delay performance of our method is competitive with the best existing tree constructions [4], yet our average wirelength is superior as compared with such previous methods. Our algorithms are efficient, and our basic approaches are amenable to numerous extensions of the routing design problem, such as Steiner graph routing, critical sink routing, and wire sizing.

The rest of our paper is organized as follows. Section 2 gives basic definitions and formalizes the problem of constructing optimal-delay interconnection topologies. Section 2 also discusses the delay models, including the SPICE parameters used in the simulations. In Section 3 we

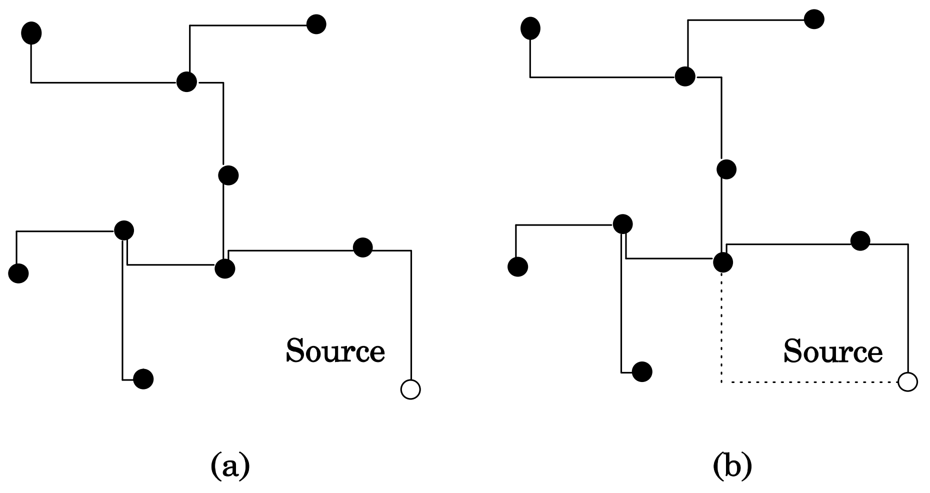

Figure 2: An example of a random net of 10 pins where the minimum spanning tree on the left (a) has a SPICE delay 5.4 nanoseconds; adding a single edge to the topology as shown on the right (b) creates a routing graph with SPICE delay of 3.6 nanoseconds, a 33.3% delay improvement. The total wirelength was increased by only 21.5% as compared to the MST cost.

present several basic heuristics that capture the intuitions behind choosing the extra wires to be added to a routing tree in order to optimize delay. In Section 4, we provide experimental results on the performance of our heuristics, and compare them with previous work. Section 5 concludes with extensions and directions for future research.

2 Problem Formulation

A *signal net* $N = \{n_0, n_1, ..., n_k\}$ is a fixed set of *pins* in the Manhattan plane to be connected by a *routing graph* $G = (N, E)$, where $E \subseteq N \times N$. Pin $n_0 \in N$ is a *source* (i.e., where the signal originates), and the remaining pins are *sinks* (i.e., where the signal propagates to). Each edge $e_{ij} \in E$ has an associated *edge cost*, d_{ij}, equal to the Manhattan distance between its two endpoints n_i and n_j; the *cost* of G is the sum of its edge costs. We use $t(n_i)$ to denote the signal propagation delay from the source to pin n_i. Our goal is to construct a routing which spans the net and which minimizes the maximum source-sink delay:

Optimal Routing Graph (ORG) Problem: Given a signal net $N = \{n_0, n_1, ..., n_k\}$ with source n_0, construct a routing graph $G = (N, E)$, $E \subseteq N \times N$, such that $t(G) = \max_{i=1}^{k} t(n_i)$ is minimized.

Note that the ORG problem generalizes the Optimal Routing Tree (ORT) problem of [4], which corresponds to the special case where G is a tree; this special case is studied extensively in [4] [5]. The case where certain sinks in the ORG are identified as critical is discussed in Section 5.1 below.

The specific routing graph G that solves the ORG problem will depend on the model used to estimate the delay $t(G)$. Ideally, we would like to compute and optimize delay according to the complete physical attributes of the circuit. To this end, we use the circuit simulator SPICE [16], which is generally regarded as the best available tool for obtaining a precise, complete measure of interconnect delay. These are representative of a typical 0.8μ CMOS process. Our SPICE delay model uses constant resistance and capacitance values per unit length of interconnect (i.e., both resistance and capacitance are proportional to wirelength). The root of the tree is driven by a resistor connected to the source pin. In addition, sink loading capacitances are used at all the pins to model loads driven by the interconnect. The SPICE parameters that we used in our simulations are given in Table 1.

Parameter	**Value**
driver resistance	$100\ \Omega$
wire resistance	$0.03\ \Omega/\mu m$
wire capacitance	$0.352\ fF/\mu m$
wire inductance	$492\ fH/\mu m$
sink loading capacitance	$15.3\ fF$
layout area	10^2 mm^2

Table 1: Parameter values for the CMOS interconnect technology used in our SPICE model.

Unfortunately, SPICE delay is too computationally prohibitive to evaluate during the routing phase of layout, and we are thus forced to seek other alternatives. Another delay model is the Elmore delay formula [11], which was shown in [4] to have both high accuracy and fidelity in comparison with SPICE. The Elmore delay is defined as follows. Given routing tree $T(N)$

rooted at n_0, let e_i denote the edge from pin n_i to its parent. The resistance and capacitance of edge e_i are denoted by r_{e_i} and c_{e_i}, respectively. Let T_i denote the subtree of T rooted at n_i, and let c_i denote the sink capacitance of n_i. We use C_i to denote the *tree capacitance* of T_i, namely the sum of sink and edge capacitances in T_i. Using this notation, the Elmore delay along edge e_i is equal to $r_{e_i}(c_{e_i}/2 + C_i)$. Let r_d denote the output driver resistance at the net's source. Then the Elmore delay $t_{ED}(n_i)$ from source n_0 to sink n_i is given by:

$$t_{ED}(n_i) = r_d C_{n_0} \quad + \sum_{e_j \in path(n_0,n_i)} r_{e_j}(c_{e_j}/2 + C_j). \tag{1}$$

We can extend the t_{ED} function to entire trees by defining $t_{ED}(T(N)) = \max_{i=1}^{k} t_{ED}(n_i)$. If r_{e_j} and c_{e_j} are proportional to the length of e_j, the delay $t_{ED}(n_i)$ is quadratic in the length of the n_0-n_i path. Because of its relatively simple form, Elmore delay can be calculated in $O(k)$ time, as noted by Rubinstein et al. [19]. Unfortunately, the Elmore delay model outlined above applies only to tree topologies, and in order to extend this formula to non-tree topologies, additional transformations are required [6]. We use the Elmore delay model in some of our heuristics to approximately solve the ORG problem.

3 Low Delay Routing Graph Heuristics

The ORG problem may be solved heuristically by starting with some reasonable tree topology such as the minimum spanning tree, and searching for some new edge to add, so that the delay in the resulting routing graph will be minimized. We add this edge into the routing graph, and iterate this process (i.e., we look for yet another good edge to add). We terminate when no further delay improvement is possible. An execution example of this method, called the Low Delay Routing Graph (LDRG) algorithm, is shown in Figure 3, while the LDRG algorithm is formalized in Figure 4.

If we use SPICE inside the LDRG method to determine circuit delay, such a method will be computationally prohibitive. Instead, we would like to efficiently determine which wires should be added in order to reduce delay by the greatest amount. We now propose several effective

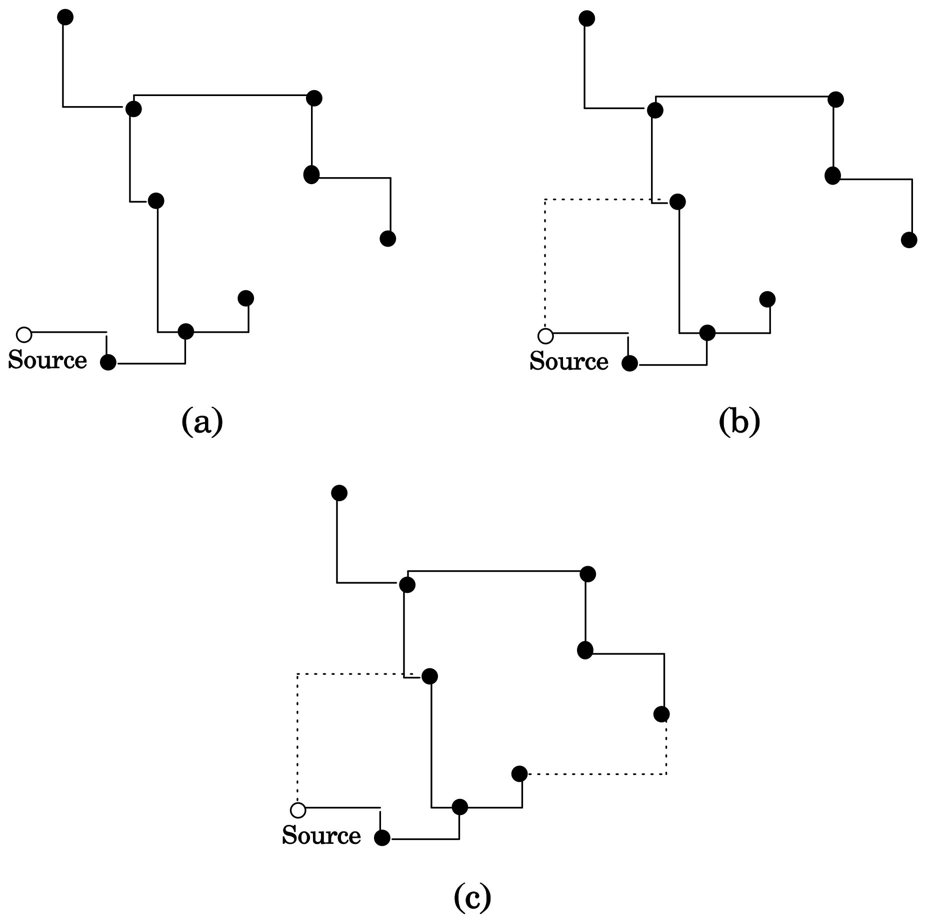

Figure 3: An execution of the LDRG algorithm on a random net of 10 pins. The MST shown at the top left (a) has SPICE delay of 4.4 nanoseconds, while the SLDRG tree at the top right (b) has SPICE delay of 4.1 nanoseconds; thus delay was improved by 7% (at a wirelength penalty of 25%). In the next iteration (c) a second new edge was added, bringing the overall delay reduction to 3.9 nanoseconds, or 11.4% reduction (at an overall wirelength penalty of 40%).

heuristics to for approximating solutions to the ORG problem. In Section 4 we compare the performance of these heuristics with the less efficient SPICE-based LDRG method discussed above. We propose three distinct additional heuristics; each heuristic starts with the MST topology, and then modifies it by connecting the source pin n_0 to some other pin in the topology, according to some fixed rule. These three heuristics (one corresponding to each fixed connection rule) are described as follows:

```
Low Delay Routing Graph (LDRG) Algorithm
Input: signal net $N$ with source $n_0 \in N$
Output: low-delay routing graph $G = (N, E)$
1.  $G = (N, E)$ where $E$ are the edges of the MST over N
2.  While $\exists\ e_{ij} \in N \times N$ such that $t((N, E \cup \{e_{ij}\})) < t(G)$
3.        Do $G = (N, E \cup \{e_{ij}\})$
4.  Output resulting routing topology $G$
```

Figure 4: The Low Delay Routing Graph heuristic: a greedy approximation of optimal routing graphs.

- **H1:** Connect n_0 to the pin with the longest SPICE delay;
- **H2:** Connect n_0 to the pin with the longest Elmore delay;
- **H3:** Connect n_0 to the pin with the largest value of (pathlength $\times$ Elmore) / length-of-new-edge.

The main edge selection step in heuristic H1 may be iterated until no further delay improvement is possible (the variants involving the Elmore delay formula can not be iterated, since Elmore delay is only defined for trees, not for arbitrary graphs). In practice, we observed that on average only two iterations occur before no further improvement is possible. The time complexity of H1 is dominated by a single call to spice; however, the time complexity of both H2 and H3 is linear if the MST is provided (otherwise, the time required by H2 and H3 is dominated by the MST computation); this sharply contrasts the much higher time complexities of comparable methods.

When vias (i.e., Steiner points) are allowed, the ORG problem can be generalized to allow Steiner points as junctures in the routing, in order to afford further opportunity for delay and wirelength optimization. The Steiner formulation of the ORG problem is as follows:

Steiner Optimal Routing Graph (SORG) Problem: Given a signal net $N = \{n_0, n_1, ..., n_k\}$ with source n_0, find a set S of Steiner points and construct a routing graph $G = (N \cup S, E)$, $E \subseteq (N \cup S) \times (N \cup S)$, such that $t(G) = \max_{i=1}^{k} t(n_i)$ is minimized.

Experimental results related to this formulation are discussed in Section 4. An execution example for a Steiner version of the LDRG algorithm (SLDRG) is shown in Figure 5, while the formal statement of this algorithm is given in Figure 6. To find the Steiner tree over the net (Step 1 of the SLDRG algorithm of Figure 6), an efficient implementation of the Iterated 1-Steiner algorithm of Kahng and Robins may be used [2] [3] [13].

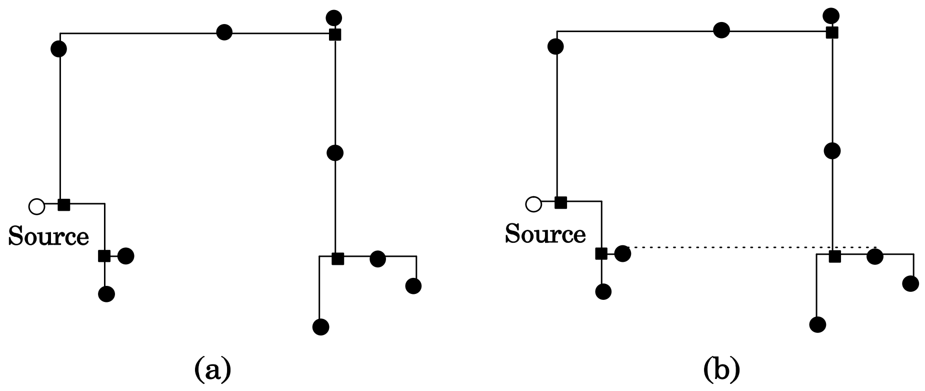

Figure 5: An execution of SLDRG algorithm (the Steiner version of the LDRG algorithm) on a random net of 10 pins. The Steiner tree shown on the left (a) has SPICE delay of 2.8 nanoseconds (Steiner points are depicted by the small squares), while the SLDRG routing on the right (b) has SPICE delay of 1.9 nanoseconds, corresponding to 32% improvement (the wirelength increase was 25%).

```
Steiner Low Delay Routing Graph (SLDRG) Algorithm
Input: signal net $N$ with source $n_0 \in N$
Output: low-delay routing Steiner graph $G = (\hat{N}, E)$, $E \subseteq \hat{N} \times \hat{N}$
1.  Compute a Steiner tree $G = (\hat{N}, E)$ over $\hat{N} = N \cup S$,
        where $S$ are Steiner points,
        and $E \subseteq \hat{N} \times \hat{N}$ is the set of Steiner tree edges
2.  While $\exists\ e_{ij} \in \hat{N} \times \hat{N}$ such that $t((\hat{N}, E \cup \{e_{ij}\})) < t(G)$
3.        Do $G = (\hat{N}, E \cup \{e_{ij}\})$
4.  Output resulting routing topology $G$
```

Figure 6: The Steiner Low Delay Routing Graph heuristic: a greedy approximation of optimal routing graphs when Steiner points may be introduced.

4 Experimental Results

We have implemented the LDRG algorithm, the SLDRG algorithm, as well as the three heuristics H1, H2 and H3 using C in the UNIX/Sun environment. We have run trials on sets of 50 nets for each of several net sizes; pin locations were randomly chosen from a uniform distribution

LDRG Algorithm Statistics						
	net	All Cases		Percent	Winners Only	
	size	Delay	Cost	Winners	Delay	Cost
LDRG	5	0.94	1.22	52	0.88	1.44
Iteration	10	0.84	1.23	90	0.82	1.25
One	20	0.81	1.16	100	0.81	1.16
	30	0.76	1.11	100	0.76	1.11
	net	All Cases		Percent	Winners Only	
	size	Delay	Cost	Winners	Delay	Cost
LDRG	5	NA	NA	NA	NA	NA
Iteration	10	0.98	1.04	10	0.79	1.40
Two	20	0.91	1.13	42	0.78	1.30
	30	0.83	1.53	68	0.75	1.23

Table 2: Average delay and cost values of the routing graph produced by the LDRG algorithm; all values are normalized to the corresponding MST values. The "All Cases" column reports the average values over all 50 instances, including the ones where the LDRG algorithm was not able to improve on the MST delay. The "Percent Winners" column reports the percentage of the cases where the LDRG algorithm was able to improve on the MST delay. The "Winners Only" column reports the average values only for those instances where the LDRG algorithm was able to improve on the MST delay.

in a square layout region. Our inputs correspond to the same CMOS interconnect technology discussed in Section 2.

Table 2 and Table 3 give the performance of the LDRG and SLDRG Algorithms with respect to all samples ("all cases"), as well as with respect to only those cases which the added edge will yield a routing graph with a better delay than that of the MST ("winners only"). For example, nets of 10 pins have 16% less delay of the MST on average, beating MST 90% of the time. Of those 90% that win, the LDRG routing graphs have on average 18% less delay, as compared to the MST. In Table 2 we see that for 30 pins the LDRG Algorithm always wins over MST, yielding an average delay improvement of 24% over MST, with wirelength penalty of only 11% over MST. For the SLDRG algorithm (Table 3) we observe a 23% delay improvement and 10% wirelength penalty over MST.

Tables 4 and 5 provide statistics for the H1, H2, and H3 heuristics discussed in Section 3. These heuristics attempt to minimize (or even remove altogether) the computationally expensive calls to SPICE. Heuristic H1 makes only one call to SPICE (as opposed to LDRG which makes

SLDRG Algorithm Statistics					
net	All Cases		Percent	Winners Only	
size	Delay	Cost	Winners	Delay	Cost
5	0.99	1.02	4	0.94	1.59
10	0.91	1.20	66	0.87	1.30
20	0.79	1.17	94	0.77	1.18
30	0.77	1.10	100	0.77	1.10

Table 3: Average delay and cost values of the routing graph produced by the SLDRG algorithm; all values are normalized to the corresponding Steiner tree values. The "All Cases" column reports the average values over all 50 instances, including the ones where the SLDRG algorithm was not able to improve on the MST delay. The "Percent Winners" column reports the percentage of the cases where the SLDRG algorithm was able to improve on the MST delay. The "Winners Only" column reports the average values only for those instances where the SLDRG algorithm was able to improve on the MST delay.

H1 Heuristic Statistics						
	net	All Cases		Percent	Winners Only	
	size	Delay	Cost	Winners	Delay	Cost
H1 Iteration One	5	0.98	1.10	20	0.90	1.49
	10	0.93	1.17	48	0.84	1.35
	20	0.88	1.16	68	0.82	1.24
	30	0.83	1.17	82	0.80	1.17
	net	All Cases		Percent	Winners Only	
	size	Delay	Cost	Winners	Delay	Cost
H1 Iteration Two	5	NA	NA	NA	NA	NA
	10	0.98	1.03	10	0.81	1.34
	20	0.99	1.02	6	0.87	1.26
	30	0.95	1.04	24	0.80	1.18

Table 4: Average delay and cost values of the routing graph produced by the H1 algorithm; all values are normalized to the corresponding MST values. The "All Cases" column reports the average values over all 50 instances, including the ones where the H1 heuristic was not able to improve on the MST delay. The "Percent Winners" column reports the percentage of the cases where the H1 heuristic was able to improve on the MST delay. The "Winners Only" column reports the average values only for those instances where the H1 heuristic was able to improve on the MST delay.

a quadratic number of calls to SPICE). We see from Table 4 that H1 is the heuristic with performance closest to that of the LDRG algorithm. For nets of size 20, H1 affords (in the first iterative stage) a 12% delay improvement over MST, with only a 17% wirelength penalty.

H2 Heuristic Statistics					
net	All Cases		Percent	Winners Only	
size	Delay	Cost	Winners	Delay	Cost
5	1.14	1.64	18	0.89	1.48
10	0.99	1.42	47	0.82	1.34
20	0.91	1.29	68	0.83	1.24
30	0.84	1.23	80	0.79	1.21

H3 Heuristic Statistics					
net	All Cases		Percent	Winners Only	
size	Delay	Cost	Winners	Delay	Cost
5	1.10	1.59	0	NA	NA
10	0.93	1.33	64	0.84	1.29
20	0.85	1.20	92	0.83	1.19
30	0.77	1.13	90	0.76	1.13

Table 5: Average delay and cost values of the routing graph produced by the H2 and H3 heuristics; all values are normalized to the corresponding MST values. The "All Cases" column reports the average values over all 50 instances, including the ones where the heuristics were not able to improve on the MST delay. The "Percent Winners" column reports the percentage of the cases where the heuristics were able to improve on the MST delay. The "Winners Only" column reports the average values only for those instances where the heuristics were able to improve on the MST delay.

Even if we limit ourselves to heuristic solutions that do not call SPICE *at all* (e.g., H2 and H3), significant delay reductions can still be realized. We observe from Table 5 that for 20 pins (over all cases), heuristic H3 offers a 15% delay improvement over MST. Furthermore, H3 improves upon the MST often than does H1 (for nets of 10, 20 and 30 pins).

We also investigated how LDRG routings fare against trees with best known delay characteristics, namely Elmore Routing Trees (ERTs), which were recently found to be near-optimal by Boese et al. [4]. Table 6 provides the relative performance of the ERT to the MST. Using Table 5 we can compare heuristics H2 and H3 to ERT. The data indicates that H3 beats ERT with respect to wirelength (for 20 pin nets H3 has 20% wirelength penalty over MST, where ERT's 26%).

We also ran the LDRG algorithm using an ERT as the starting point (rather than the MST). Table 7 provides data for such an ERT-based LDRG variant. For nets of size 20 over all

Elmore Routing Tree Statistics					
net	All Cases		Percent	Winners Only	
size	Delay	Cost	Winners	Delay	Cost
5	0.94	1.22	54	0.92	1.14
10	0.85	1.27	78	0.84	1.19
20	0.80	1.26	92	0.79	1.22
30	0.71	1.21	97	0.71	1.21

Table 6: Average delay and cost values of the routing tree produced by the ERT algorithm; all values are normalized to the corresponding MST values. The "All Cases" column reports the average values over all 50 instances, including the ones where the ERT algorithm was not able to improve on the MST delay. The "Percent Winners" column reports the percentage of the cases where the ERT algorithm was able to improve on the MST delay. The "Winners Only" column reports the average values only for those instances where the ERT algorithm was able to improve on the MST delay.

ERT-Based LDRG Algorithm Statistics					
net	All Cases		Percent	Winners Only	
size	Delay	Cost	Winners	Delay	Cost
5	0.99	1.38	8	0.92	1.31
10	0.99	1.22	22	0.96	1.21
20	0.98	1.13	44	0.96	1.12
30	0.97	1.12	56	0.96	1.12

Table 7: Average delay and cost values of the routing graph produced by the LDRG algorithm where an ERT is used as an initial tree instead of an MST; all values are normalized to the corresponding ERT values. The "All Cases" column reports the average values over all 50 instances, including the ones where the LDRG algorithm was not able to improve on the ERT delay. The "Percent Winners" column reports the percentage of the cases where the LDRG algorithm was able to improve on the ERT delay. The "Winners Only" column reports the average values only for those instances where the LDRG algorithm was able to improve on the ERT delay. These results indicate that even the highly delay-optimized ERT can be improved upon by the LDRG method.

cases, the ERT-based LDRG algorithm affords an average of 2% improvement over ERT delay. When considering only the cases when ERT-based LDRG wins over ERT, we can realize routing graphs with as much as 4% improvement over ERTs on average. This implies that there exist non-tree routings which can beat the optimal tree routing (since is was shown by Boese et al. [4] that the average delay of ERTs is only 2% away from optimal).

5 Conclusions and Future Directions

In this paper we have explored the consequences of abandoning an implicit restriction common to previous routing formulations, namely the insistence on a strictly acyclic (tree) routing topology. Instead, we reformulated the routing problem as one of constructing a routing *graph* with low delay. We have shown that adding extra wires can often improve signal propagation delay by exploiting the tradeoff between the capacitance and resistance in a circuit. While adding extra wires to a routing tree increases the overall routing capacitance, such extra wires can cause the resistance between the source and some pins to be dramatically reduced, thus enabling a routing with superior delay characteristics.

In order to obtain realistic results, we used SPICE to determine the efficacy of our methods. Our results are very encouraging: the addition of a single new wire/edge to an existing minimum spanning tree routing can improve the average signal propagation delay by up to 24%, while the average interconnection cost increases by only 11%, depending on net size. The delay performance of our method is competitive with the best existing tree constructions [4], yet our average wirelength is significantly better. Our methods are highly efficient, and extend to various routing regimes. Future work entails addressing several natural related formulations, including Steiner graph routing, a critical sink variant, and a wire-sized version of the ORG problem, as discussed below.

5.1 Critical Sink Routing

The ORG problem minimizes individual net delay but ignores the interdependence of nets in the overall circuit. In other words, the ORG problem concentrates on net-dependent objectives, rather than path-dependent objectives based on pre-defined critical paths. Following the formulation of [5], a path-dependent variant of the ORG problem can be defined as follows. For each sink n_i in N we can associate a *criticality* α_i, reflecting the timing information obtained during the performance-driven placement phase. Our goal is to construct a routing graph $G = (N, E)$ which minimizes the weighted sum of the sink delays:

The Critical-Sink Optimal Routing graph (CSORG) Problem: Given a signal net

$N = \{n_0, n_1, ..., n_k\}$ with source n_0 and possibly varying sink criticalities $\alpha_i \geq 0$, $i = 1, \ldots, k$, construct a routing graph $G = (N, E)$ such that $\sum_{i=1}^{k} \alpha_i \cdot t(n_i)$ is minimized.

The CSORG problem formulation is quite general. For example, it captures *traditional* performance criteria for routing trees: (i) we can minimize *average* delay to all sinks by using all $\alpha_i \equiv$ some positive constant; and (ii) another variation can be used to solve the simple, but practical case where exactly one critical sink n_{CS} has been identified, i.e., $\alpha_{CS} = 1$ and all other $\alpha_i = 0$. The CSORG problem for the special case where G is a tree is studied in [4].

5.2 Wire Sizing

Having two separate parallel wires of width w running between two pins is equivalent to having a single wire of width $2w$. Some of the wires added by our algorithms may therefore be "merged" with adjacent wires to yield wider wires. Our ORG problem may thus be further generalized to model wire widths as follows:

Wire-Sized Optimal Routing Graph (WSORG) Problem: Given a signal net $N = \{n_0, n_1, ..., n_k\}$ with source n_0, construct a **weighted** routing graph $G = (N, E)$, $E \subseteq N \times N$, with edge width function $w : E \rightarrow \Re$, such that $t(G) = \max_{i=1}^{k} t(n_i)$ is minimized.

In this formulation the wire width function w determines the width of each edge in the routing graph G, and the delay function $t(G)$ also depends on the wire widths. In most practical applications a discrete grid is used for layout, and thus the range of w may be restricted to the integers. Intuitively, wider wires near the source pin would tend to reduce overall signal propagation delay. Future work would investigate the WSORG problem, especially the case where G is a tree.

5.3 Combinations

Naturally, various extensions to the basic ORG problem may be combined, to yield general multi-objective routing formulations, such as:

Hybrid Optimal Routing Graph (HORG) Problem: Given a signal net $N = \{n_0, n_1, ..., n_k\}$ with source n_0, and sink criticalities $\alpha_i \geq 0$, $i = 1, \ldots, k$, find a set S of Steiner points and construct a weighted routing graph $G = (N \cup S, E)$, $E \subseteq (N \cup S) \times (N \cup S)$, with edge width function $w : E \longrightarrow \Re$, such that $\sum_{i=1}^{k} \alpha_i \cdot t(n_i)$ is minimized.

Note that the HORG problem subsumes all the other formulations stated above. On the other hand, such a general formulation as the HORG problem will be correspondingly more difficult to address effectively.

6 Acknowledgments

We wish to thank Mary Lou Jurgens for her help with proofreading. Many thanks go to Kevin McLaughlin, and Professors Andy Schwab, Hugh Landes, and Michael Shur of the University of Virginia Electrical Engineering Department for their help with SPICE.

References

[1] C. J. Alpert, T. C. Hu, J. H. Huang, and A. B. Kahng, *A Direct Combination of the Prim and Dijkstra Constructions for Improved Performance-Driven Global Routing*, in Proc. IEEE Intl. Symp. on Circuits and Systems (to appear), Chicago, IL, May 1993.

[2] T. Barrera, J. Griffith, , G. Robins, and T. Zhang, *Narrowing the Gap: Near-Optimal Steiner Trees in Polynomial Time*, in Proc. IEEE Intl. ASIC Conf. (to appear), Rochester, NY, September 1993.

[3] T. Barrera, J. Griffith, S. A. McKee, G. Robins, and T. Zhang, *Toward a Steiner Engine: Enhanced Serial and Parallel Implementations of the Iterated 1-Steiner Algorithm*, in Great Lakes Symposium on VLSI, Kalamazoo, MI, March 1993, pp. 90–94.

[4] K. D. Boese, A. B. Kahng, B. A. McCoy, and G. Robins, *Towards Optimal Routing Trees*, in to appear in ACM/SIGDA Physical Design Workshop, Lake Arrowhead, CA, April 1993.

[5] K. D. Boese, A. B. Kahng, and G. Robins, *High-Performance Routing Trees With Identified Critical Sinks*, in Proc. ACM/IEEE Design Automation Conf. (to appear), Dallas, June 1993.

[6] P. K. Chan and K. Karplus, *Computing Signal Delay in General RC Networks by Tree/Link Partitioning*, IEEE Trans. on Computer-Aided Design, 9 (1990), pp. 898–902.

[7] J. Cohoon and J. Randall, *Critical Net Routing*, in Proc. IEEE Intl. Conf. on Computer Design, Cambridge, MA, October 1991, pp. 174–177.

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

[9] W. E. Donath, R. J. Norman, B. K. Agrawal, S. E. Bello, S. Y. Han, J. M. Kurtzberg, P. Lowy, and R. I. McMillan, *Timing Driven Placement Using Complete Path Delays*, in Proc. ACM/IEEE Design Automation Conf., 1990, pp. 84–89.

[10] A. E. Dunlop, V. D. Agrawal, D. Deutsch, M. F. Jukl, P. Kozak, and M. Wiesel, *Chip Layout Optimization Using Critical Path Weighting*, in Proc. ACM/IEEE Design Automation Conf., 1984, pp. 133–136.

[11] W. C. Elmore, *The Transient Response of Damped Linear Networks with Particular Regard to Wide-Band Amplifiers*, J. Appl. Phys., 19 (1948), pp. 55–63.

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

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

[14] I. Lin and D. H. C. Du, *Performance-Driven Constructive Placement*, in Proc. ACM/IEEE Design Automation Conf., 1990, pp. 103–106.

[15] M. Marek-Sadowska and S. P. Lin, *Timing Driven Placement*, in Proc. IEEE Intl. Conf. on Computer-Aided Design, Santa Clara, CA, November 1989, pp. 94–97.

[16] L. Nagel, *SPICE2: A Computer Program to Simulate Semiconductor Circuits*, May 1975.

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

[18] B. T. Preas and M. J. Lorenzetti, *Physical Design Automation of VLSI Systems*, Benjamin/Cummings, Menlo Park, CA, 1988.

[19] J. Rubinstein, P. Penfield, and M. A. Horowitz, *Signal Delay in RC Tree Networks*, IEEE Trans. on Computer-Aided Design, 2 (1983), pp. 202–211.

[20] S. Sutanthavibul and E. Shragowitz, *An Adaptive Timing-Driven Layout for High Speed VLSI*, in Proc. ACM/IEEE Design Automation Conf., 1990, pp. 90–95.