High-Performance Routing Trees
With Identified Critical Sinks

Kenneth D. Boese,
Andrew B. Kahng,
Gabriel Robins

Technical Report No. CS-92-37
November 1, 1992

High-Performance Routing Trees With Identified Critical Sinks*

Kenneth D. Boese, Andrew B. Kahng and Gabriel Robins†

CS Dept., University of California at Los Angeles, Los Angeles, CA 90024-1596
† CS Dept., University of Virginia, Charlottesville, VA 22903-2442

Abstract

We present *critical-sink routing tree* (CSRT) constructions which yield high-performance routing trees by exploiting the critical-path information that may be available during timing-driven layout. Motivated by analysis of the Elmore delay formula, we propose the CS-Steiner class of heuristics and a "Global Slack Removal" algorithm; these modify traditional Steiner tree constructions to optimize signal delay at identified critical sinks. Extensive timing simulations, using industry IC and MCM technology parameters and a fast simulator based on a 2-pole distributed RCL delay approximation [29], show that this simple approach affords very significant improvements over existing "performance-driven" routing tree constructions. Next, we observe that all existing routing tree objectives (e.g., minimum-cost Steiner [16] or bounded-radius [5]) are *heuristic* abstractions of the linear or Elmore delay models. We therefore propose a new class of efficient *Elmore routing tree* (ERT) constructions, which iteratively add tree edges that are *optimal* in terms of Elmore delay. For the CSRT problem, this direct optimization of Elmore delay yields trees that significantly improve (by averages of up to 69%) upon minimum Steiner routings in terms of delays to identified critical sinks. Moreover, ERTs serve as generic high-performance routing trees when no critical sink is specified: for 8-sink nets in 0.8μ CMOS IC technology, we improve average sink delay by 10% and maximum delay by 13% over the minimum Steiner routing. For a typical MCM technology, the corresponding improvements are 42% and 22%. The ERT approach represents a basic advance over existing performance-driven routing tree constructions, including such recent works as [1] [4] [5].

1 Introduction

Due to the scaling of VLSI technology, interconnection delay has become a dominant concern in the design of complex, high-performance circuits [7, 25]. As a consequence, performance-driven layout design has become an active area of research over the past several years. Early work in this field centered on performance-driven *placement*, where timing-critical paths are determined by static timing analysis, and modules in these paths are then placed close together (see, e.g., [7, 10, 17, 18, 12, 25]). Later, a number of performance-driven *interconnection* algorithms were developed wherein for a given signal net, the typical objective is to minimize the average (or maximum) signal delay from the source pin to the sink pins. Based on these results, the prevailing approach to performance-driven layout involves use of static timing analysis to iteratively "drive" changes within the module placement and the global routing phases.

*Partial support for this work was provided by a GTE Graduate Fellowship and by NSF MIP-9110696, NSF Young Investigator Award MIP-9257982, ARO DAAK-70-92-K-0001, and ARO DAAL-03-92-G-0050. We are also grateful for the support of Cadence Design Systems under the State of California MICRO program.

Existing performance-driven placement algorithms fall into two general classes:

1. *Net-dependent* placement algorithms typically use centroid-connected star cost [24], probabilistic estimates of Steiner tree cost [12], minimum spanning tree cost [7] or the bounding box semiperimeter [18] to estimate wire capacitance and signal delay for a multi-terminal net. From this information, critical timing paths between primary inputs and primary outputs may be computed, after which module placements are updated to reduce these "net-based" objective functions for signal nets which lie along the critical paths.

2. *Path-dependent* placement algorithms are distinguished by their consideration of delay between the source and a particular *critical sink* of a multi-terminal net. The critical sink is typically determined via timing analysis using known module delays and estimated path delays. For example, Lin and Du [17] use a linear delay approximation so that their method updates the module placement to reduce the rectilinear distance between sources and critical sinks. Other path-dependent placement methodologies include those due to Hauge et al. [10] and Teig et al. [26].

If a timing-critical path passes through a given net, the path-dependent approach will afford an explicit routing constraint which bounds delay at that net's critical sink. While the net-dependent approach arguably provides only implicit routing constraints, it is easy to identify critical sinks after the timing analysis has been performed, or even *a priori* by finding paths in the design that contain more module delays. However, despite the availability of such critical-path information during the iterative performance-driven layout process, current routing methods do not fully exploit this information, thus betraying what might be termed a "placement-routing mismatch" in existing methodologies. With this in mind, our paper offers two contributions. First, we present new algorithms which *directly* exploit available critical-path timing information to yield high-performance routing trees. Second, we propose a much more basic change to current routing tree approaches: we avoid the abstraction inherent in such previous objectives as "minimum cost" or "bounded radius", and instead give a class of greedy heuristics which *directly* optimize Elmore delay in the tree construction.

The remainder of this paper is organized as follows. Section 2 gives a formal definition of the *critical-sink routing tree* (CSRT) problem, and in this context discusses existing algorithms in the performance-driven global routing literature. Section 2 also presents several motivating observations vis-a-vis the desirable qualities of a "performance-driven" routing tree which optimizes the first-order moment of the impulse response in the distributed RC delay model, i.e., Elmore delay. Section 3 then presents two new classes of CSRT algorithms. We first describe the *CS-Steiner* method, which perturbs an existing Steiner tree construction to directly capture the presence of identified critical sinks. We then propose an efficient class of *Elmore routing tree* (ERT) constructions. These generalize to yield good CSRT solutions and are moreover the first methods to directly optimize Elmore delay, unfettered by heuristic abstractions inherent in previous

routing tree objectives. Our experimental results are presented in Section 4, where we compare delays at critical sinks in our heuristic tree topologies with analogous delays obtained using the best-performing efficient Steiner tree heuristic [13]. These results show that both of our methods effectively address the CSRT formulation, attaining up to 69% *expected* delay improvement to identified critical sinks. Moreover, we show that the ERT approach yields *generic* high-performance routing trees in the case where no critical sink has been identified: for 8-sink nets in 0.8μ CMOS IC technology, we improve average sink delay by 10% and maximum delay by 13% over the minimum Steiner routing. For a typical MCM technology, the corresponding improvements are 42% and 22%. This represents a significant advance over every existing performance-driven routing tree construction in the literature, including such recent works as [1] [4] [5] [20].

2 High-Performance Routing Tree Design

In our discussion, we say that a *signal net* N consists of a set of *pins* or *terminals* which are to be connected by a *routing tree* $T(N)$ in the Manhattan plane. Since module placements are fixed prior to the global routing stage of layout, we assume that the net N corresponds to a particular set of pin locations $N = \{n_0, n_1, ..., n_k\} \subset \Re^2$, where n_0 always refers to the location of an identified *source* pin, and the n_i $(1 \leq i \leq k)$ are the *sinks* of the net. We say that the *cost* of an edge e_{ij} in $T(N)$, denoted by d_{ij}, is the Manhattan distance between the endpoints n_i and n_j of the edge. The *cost* of $T(N)$ is simply the sum of its edge costs. In a given routing tree $T(N)$, the signal delay between two terminals n_i and n_j is denoted by $t(n_i, n_j)$; we use the shorthand notation $t(n_i)$ to indicate the delay from the source to a sink n_i. Finally, we allow each n_i to have an associated *criticality* α_i, reflecting the timing information obtained during the performance-driven placement phase. Our goal is to construct a routing tree $T(N)$ which minimizes the weighted sum of the sink delays:

Critical-Sink Routing Tree (CSRT) Problem: Given a signal net $N = \{n_0, n_1, ..., n_k\} \subset \Re^2$ with source n_0 and possibly varying sink criticalities $\alpha_i \geq 0$, $i = 1, \ldots, k$, construct a routing tree $T(N)$ such that $\sum_{i=1}^{k} \alpha_i \cdot t(n_i)$ is minimized.

Notice that the CSRT problem formulation is quite general. In particular, traditional performance criteria for routing trees can be easily captured: (i) we can minimize the *average delay* to all sinks by using all $\alpha_i =$ some positive constant, then taking the sum of the weighted delays using the L_1 norm (i.e., the "standard" sum of magnitudes); and (ii) we can minimize the *maximum delay* to any sink by using all $\alpha_i =$ some positive constant, then taking the sum of the weighted delays using the L_∞ norm (i.e., the "max" norm). In the discussion below, we will concentrate on the simple yet realistic case where *exactly one* critical sink, denoted by n_c, has been identified. In other words, we assume that $\alpha_c = 1$ and that all other $\alpha_i = 0$. Our methods may be generalized to the case where a small number of critical sinks is specified.

2.1 Previous Performance-Driven Routing Approaches

In this subsection, we review previous work with an eye to revealing the rationales for current performance-driven routing objectives. Many of these objectives can be viewed simply as approximations of the "true" signal delay $t(n_i)$ used in the CSRT formulation.

Much of the early work in performance-driven routing relied on the assumption that an optimal routing tree corresponds to the minimum Steiner tree over N. For example, the method of Dunlop et al. [8] uses static timing analysis to yield net priorities, so that the highest-priority nets may be routed by the minimum Steiner tree with no detours, while lower-priority nets may be routed by suboptimal Steiner trees due to possible routing blockages. Kuh, Jackson and Marek-Sadowska [15] have given an approach tuned to hierarchical building-block layouts, and Prastjutrakul and Kubitz [20] use A* heuristic search [19] for a similar problem domain. The latter work stands out for two reasons: (i) it uses the Elmore delay formula [9] in its tree optimization (but then implicitly relies on a crude abstraction of signal delay to constrain the construction, as we discuss below); and (ii) it was one of the first works to allow prescribed upper bounds on the individual $t(n_i)$, an increasingly well-studied formulation which is somewhat orthogonal to the CSRT problem above.[1]

In 1991, Cohoon and Randall [3] proposed a heuristic which tried to simultaneously reduce both the longest source-sink pathlength of the routing tree (which we call the tree *radius*) as well as the total edge length in the tree (i.e., the tree *cost*). A more general approach was given by Cong, Kahng, Robins, Sarrafzadeh and Wong [4], wherein a parameter ϵ is used to trade off between the radius and the cost of the routing tree: the longest source-sink distance is used as a lower bound R for the routing tree radius, and a low-cost routing tree with radius bounded by $(1 + \epsilon) \cdot R$ is produced. Later, Cong et al. [5] proposed the "provably good" BRBC (bounded-radius, bounded-cost) algorithm, which afforded both radius and cost simultaneously within *constant* factors of optimal. The BRBC method, along with the work of Awerbuch et al. [2] and Khuller et al. [14], belongs to the class of what have been called "shallow/light" tree constructions. These works all achieve a smooth tradeoff between competing minimum radius (i.e., "shallow") and minimum cost (i.e., "light") interconnection objectives via the same basic idea: make a depth-first traversal of the minimum spanning tree over N, and if the accumulated pathlength from the source to some sink becomes too large, modify the tree to reduce that particular source-sink pathlength.

The radius-cost tradeoff may also be viewed as one between (i) the shortest-path tree (SPT) construction [6], wherein all source-sink paths are as short as possible, and (ii) the minimum spanning tree (MST) or minimum Steiner tree construction, which has minimum cost. Using this perspective, Alpert et al. [1]

[1] Formulations which prescribe exact and/or relative delay bounds often require iterative tree constructions to satisfy these bounds, particularly when the bounds are ill-chosen (e.g., due to incomplete information about the layout) and must be revised. While our CSRT formulation can only heuristically enforce exact or relative sink delays $t(n_i)$, it is more conducive to the "one-shot" tree construction that may be desired in the global routing phase.

recently proposed the AHHK tree construction of Figure 2.1, which achieves a direct SPT-MST tradeoff.[2] They give simulation results showing that AHHK uniformly outperforms the BRBC method of Cong et al. [5] by averages of between 6.2% and 27.9% in terms of sink delay (this value depends on the net size, the interconnect technology, and whether average or maximum sink delay is considered). In view of the superior performance of the AHHK algorithm, we will compare our new methods against AHHK in Section 4 below.

AHHK Algorithm: MST-SPT tradeoff of [1]
Input: signal net N with source $n_0 \in N$, tradeoff parameter c, $0 \leq c \leq 1$
Output: routing tree T over N
1. $T = (V, E) = (n_0, \emptyset)$
2. **while** $\|V\| < \|N\|$ **do**
3. **select** $n_j \in V$ and $n_i \in N - V$ minimizing $(c \cdot l_j) + d_{ij}$
4. /* l_j is the pathlength in T from n_0 to n_j */
5. $V = V \cup \{n_i\}$; $E = E \cup \{e_{ij}\}$
6. **output** T

Figure 1: AHHK algorithm [1], which outputs a routing tree with MST-SPT tradeoff governed by parameter c, $0 \leq c \leq 1$. Choosing $c = 0$ yields an MST, while choosing $c = 1$ yields an SPT.

Since the minimum Steiner tree is also a bona fide "performance-driven" routing tree (e.g., within the net prioritization scheme of Dunlop et al. [8] noted above), and moreover still enjoys widespread use, Section 4 further compares our new routing heuristics against minimum Steiner tree constructions. To obtain this comparison, we have implemented the 1-Steiner algorithm of [13] (see Figure 2), which gives the best heuristic performance reported in the literature (i.e., routing tree cost an average of almost 11% less than MST cost). Analysis of the Elmore model for distributed RC delay indicates the continuing utility of Steiner tree constructions for certain technology regimes, because of a close relationship between tree cost and signal delay. However, the formula for Elmore delay also clearly points out regimes wherein "shallow/light" or AHHK-style routing will be superior to minimum Steiner tree approaches. We now describe several insights obtained from the Elmore delay model; these serve to motivate our *CS-Steiner* class of CSRT heuristics in Section 3.1 below.

2.2 Intuitions From the Elmore Model

For arbitrary signal nets N, the appropriate criterion to use in *efficiently* constructing "high-performance routing trees" has not yet been well-established. In this subsection, we develop intuitions regarding the

[2] The authors of [1] note the following: (1) Dijkstra's SPT algorithm [6] begins with the trivial tree consisting only of the source n_0, and then iteratively adds the edge e_{ij} and the node n_i to the growing T, where n_i and n_j are chosen to minimize $l_j + d_{ij}$ s.t. $n_j \in T$, $n_i \in N - T$ (here, l_j is the cost of the path from n_0 to n_j in T). (2) Prim's MST algorithm begins with the trivial tree consisting only of the source n_0, and then iteratively adds the edge e_{ij} and the node n_i to the growing T, where n_i and n_j are chosen to minimize d_{ij} s.t. $n_j \in T$, $n_i \in N - T$. (3) A direct combination of these constructions would begin with the trivial tree, then iteratively add the edge e_{ij} and the node n_i to T, where n_i and n_j are chosen to minimize $(c \cdot l_j) + d_{ij}$ s.t. $n_j \in T$, $n_i \in N - T$ (c is the user-chosen MST-SPT tradeoff parameter, with $0 \leq c \leq 1$).

Iterated 1-Steiner Algorithm: Steiner tree heuristic of [13]	
Input:	signal net N
Output:	heuristic minimum rectilinear Steiner tree T over N
1.	$S = \emptyset$
2.	**while** $\lvert S\rvert < \lvert N\rvert + 1$ and $\exists$ 1-Steiner point x **do**
3.	$S = S \cup \{x\}$
4.	**output** MST$(N \cup S)$

Figure 2: The iterated 1-Steiner algorithm of [13]. For a given point set P (consisting of N and a set S of Steiner points), a 1-*Steiner point* is any point x such that $cost(MST(P \cup \{x\}))$ is minimized, with $cost(MST(P \cup \{x\})) < cost(MST(P))$. Thus, the algorithm iteratively adds interior nodes (Steiner points) that afford the largest cost reduction in the spanning tree over $N \cup S$.

"correct" objectives for critical-sink routing trees, via Elmore's formula [9] for the first-order moment of the impulse response when the routing tree is treated as a distributed RC tree. Elmore delay [9] [23] is computed as follows. Given a routing tree T, we use e_i to denote the edge from node v_i to its parent when we root the tree at n_0.[3] The resistance and capacitance of edge e_i are respectively given by r_{e_i} and c_{e_i}. Let T_i denote the subtree of T rooted at v_i, and let c_i denote the node capacitance of v_i. The *tree capacitance* C_i of T_i is the sum of node and edge capacitances in T_i. The Elmore delay from the source n_0 to the sink n_i is then expressed as the sum of Elmore delays on the edges of the n_0-n_i path:[4]

$$t_{ED}(n_0, n_i) = \sum_{e_v \in path(n_0, n_i)} r_{e_v}(c_{e_v}/2 + C_v). \tag{1}$$

Since r_{e_v} and c_{e_v} are usually proportional to the length of edge e_v, we see that $t_{ED}(n_i)$ has quadratic dependence on the length of the n_0-n_i path, suggesting a min-radius criterion. However, the C_v term implies that Elmore delay is also linear in the total edge length of the tree which lies outside the n_0-n_i path, suggesting a min-cost criterion. It should also be noted that Equation (1) implicitly assumes zero resistance and capacitance at the source node n_0, which would imply that the optimal routing tree would simply connect each sink directly to the source. This unrealistic conclusion can be avoided by representing the output driver as a wire from a new "virtual source" n_0' to the original source n_0, with the entire routing tree topology still incident to n_0. Varying the length of this wire allows us to capture features of various interconnect technologies (in particular, the relation of output driver resistance to wire resistance).

In Figure 3, we show a signal net N with identified critical sink n_c, along with three routing trees: (a) the 1-Steiner tree, (b) a minimum-cost SPT, and (c) the optimal CSRT with respect to critical sink n_c. Based on this example, the example of Figure 3(d), and Equation (1), we make the following observations.

[3] We use v_i instead of n_i to accomodate the possibility that a "node" in a Steiner routing tree is a Steiner point rather than a sink.

[4] The recursive expression implies that Elmore delay can be evaluated at *all* sinks of T in *linear* time, using a depth-first traversal of the tree [23] [27]. This fact is enabling to our efficient "Elmore routing tree" methodology, which we propose in Section 3.2.

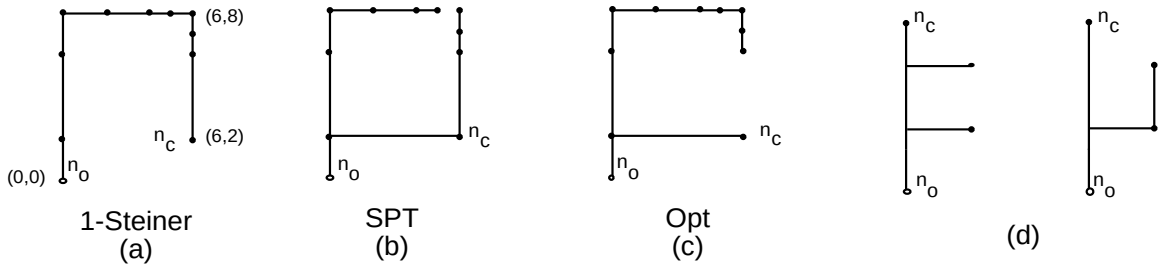


Figure 3: Parts (a)-(c): optimal Steiner tree (cost 2.0cm, delay(n_c) 5.90ns); minimum cost shortest-paths tree (cost 2.5cm, delay(n_c) 4.11ns); and optimal-delay tree (cost 2.2cm, delay(n_c) 3.07ns) for the same sink set. Coordinates shown are in mm, and 0.8μ CMOS technology parameters (see Section 4) were used for simulation purposes. Part (d): two distinct minimum-cost SPT solutions for a set of three sinks.

1. The 1-Steiner solution (a) has large delay to the critical sink n_c due to the long source-sink path. Recall that Equation (1) (assuming zero driver resistance) suggests that delays will be minimized when all source-sink paths are as short as possible.

2. However, requiring a monotone path to *every* sink, as in the SPT (b) (or in the AHHK tree with parameter $c \approx 1$) can result in too large a tree capacitance, again leading to large delay at n_c.

3. The optimal CSRT construction (c) not only demonstrates the dependence of routing topology on the choice of critical sink, but also shows hints of both star-like SPT and min-cost MST solutions.

4. If we model variations in technology by changing the n_0'-n_0 wirelength described above, we see that when the n_0'-n_0 wirelength is small, monotonicity of source-sink paths is more important than overall tree cost, so that star-like topologies give better performance. When we increase the length of the n_0'-n_0 wire (i.e., increase the output driver resistance), the Elmore delay depends more on total tree cost and a minimum-cost routing tree is preferable.[5]

5. Finally, we observe that according to Equation (1), the number of Steiner points in the n_0-n_c path should in general be minimized, and the Steiner points should be "shifted" toward the source n_0. (In Figure 3(d), note that even though the two trees shown are both shortest-path trees and minimum Steiner trees, the tree on the right has less signal delay at n_c.)

[5]The analysis for small driver resistance is increasingly appropriate, e.g., with multi-chip module interconnects; our intuition supports the use of such radius-cost tradeoffs as in [1] [3] [5]; also cf. the Steiner tree formulation of [21]. The analysis for large n_0'-n_0 wirelength reflects the previous generation of IC technologies and confirms the use of minimum Steiner and spanning tree constructions in existing global routers. Note that because the interconnect objective is so clearly technology-dependent, our experiments below are performed with respect to interconnect parameters for both IC and MCM technologies.

3 Two Classes of CSRT Heuristics

3.1 The CS-Steiner Approach

Given the observations above, we may characterize the optimal CSRT solution in Figure 3(c) as one which minimizes total tree cost, *subject to the path from n_0 to n_c being monotone.* This simultaneous consideration of radius and cost parameters recalls the motivations in [1] [3] [5], but here the tradeoff is formulated with respect to the critical sink n_c. We thus obtain a simple heuristic to address the CSRT problem. Our basic algorithm, which we call *CS-Steiner*, is shown in Figure 4.

CS-Steiner Algorithm
Input: signal net N; source $n_0 \in N$; identified critical sink $n_c \in N$
Output: heuristic CSRT solution T
1. **construct** heuristic minimum-cost tree T_0 over $N - n_c$. 2. **form** T by adding a *direct connection* from n_c to T_0, i.e., such that the n_0-n_c path in T is monotone.

Figure 4: CS-Steiner heuristic template.

The idea behind CS-Steiner is extremely simple: construct a minimum-cost Steiner routing tree as usual, but then "fix" the tree to reflect an identified critical sink.[6] Since the algorithm template in Figure 4 is quite general, we have examined a number of CS-Steiner variants. All of our variants use the 1-Steiner heuristic of Kahng and Robins [13] to construct the initial tree T_0 in Step 1. Section 4 reports results for the following three variants:[7]

1. **H0:** The direct connection in Step 2 consists of a single wire from n_c to n_0.

2. **H1:** The direct connection in Step 2 consists of the shortest possible wire that can join n_c to T_0, subject to the monotone path constraint.

[6] It is possible to coerce the CS-Steiner approach into the templates of previous methods, although this certainly does not reflect our motivations. In particular, our use of a minimum spanning tree or heuristic minimum Steiner tree in Step 1 recalls the BRBC algorithm of [5]. Our consideration of only the n_0-n_c pathlength is similar to invoking BRBC with $\epsilon = 0$ for n_c and $\epsilon = \infty$ for all other sinks. Indeed, [5] describes an extension which permits differing ϵ_i values for each sink n_i. However, the BRBC $(1 + \epsilon_i) \cdot R$ radius bound is maintained with respect to the *net radius* R, which is a function of *all* sink locations. Therefore, even by using $\epsilon_c = 0$ the BRBC extension cannot enforce a monotone path to the critical sink when $d_{0c} < max_i d_{0i} = R$. The Ph.D. thesis of Robins [22] describes how to enforce distinct ϵ_i values along with different R_i values at each sink n_i. In a sense, the CS-Steiner method resembles setting $R_c = d_{0c}$ and $\epsilon_c = 0$ in this latter scheme, but our construction greatly simplifies due to the assumption of $\epsilon_i \in \{0, \infty\}$ $\forall i$. It should be noted that the AHHK method [1] cannot be extended to address the CSRT formulation at all.

[7] We also studied two additional variants. Variant **H2** modifies Step 1 of CS-Steiner so that the initial heuristic tree T_0 is constructed over the entire net N. H2 then deletes the edge which lies directly above n_c when we root T_0 at n_0, and rejoins (the component containing) n_c to (the component containing) n_0 using a shortest possible wire from n_c, as in variant H1. Variant **H3** performs Steps 1 and 2 simultaneously by executing the 1-Steiner algorithm subject to a "maintaining monotone feasibility" constraint. In other words, we iteratively choose a Steiner point which minimizes the sum of the tree cost and the cost of any needed direct connection from n_c to n_0. The direct connection from n_c requires that there exist a monotone path through the "bounding boxes" of the edges in the path to n_0. Intuitively, this favors initial choice of Steiner nodes along some monotone path from n_0 and n_c, since such nodes will most rapidly reduce the marginal cost of adding the direct n_c-n_0 connection. We found that H2 and H3 gave much worse results than H0, H1 and HBest.

3. **HBest:** Repeatedly apply Step (2), trying all shortest connections from n_c to the edges in the heuristic tree T_0, as well as from n_c to n_0. Perform timing analysis on each of these routing trees, and return the routing tree which has the smallest delay at n_c.[8]

The complexity of these variants is dominated either by the construction of T_0 in Step 1, or possibly by the calls to timing analysis in HBest. Below, we present experimental results showing that each of these variants significantly improves critical-sink delays $t(n_c)$ over traditional routing methods. It should be noted that our experimental results reflect the incorporation of a linear-time postprocessing algorithm, called *Global Slack Removal* (GSR), which shifts edges in the 1-Steiner output to maximize the monotonicity of all source-sink paths (thus "removing U's") without increasing overall tree cost or the Elmore delay to any sink. This may be viewed as nearly equivalent to the shifting of Steiner points, which we motivated in observation (5) of Section 2.2.) For expository reasons, we defer formal description of GSR, along with its proofs of correctness, to the Appendix.

3.2 Elmore Routing Trees

As noted in Section 2.2, current routing objectives such as minimum tree cost, bounded tree radius, or prescribed cost-radius balance have all been *motivated* by consideration of the Elmore model. However, these objectives do not actually optimize Elmore delay. Rather, they only attempt to *heuristically* capture the true minimum-delay routing criterion (with the CS-Steiner method being no exception). Indeed, whether the current routing objectives are useful will depend on the prevailing technology, on the particular distribution of sink locations for a given signal net, and even on the user's ability to find some parameter value (e.g., ϵ in the BRBC algorithm, or c in the AHHK algorithm) which happens to yield a good solution for the particular input.

In this subsection, we completely avoid the abstraction inherent in "minimum cost" or "bounded radius" objectives, and propose a new class of greedy *Elmore routing tree* (ERT) algorithms which optimize Elmore delay *directly* as the routing tree is constructed. The ERT approach is efficient, since Elmore delay at all nodes of a routing tree can be evaluated in linear time (see Footnote 4 above). Based on the performance results in Section 4 for both the CSRT and "generic" performance-driven routing formulations, we believe that the ERT approach, along with its SERT and SERT-C extensions described below, offer a basic enhancement to current routing methods.

The Elmore routing tree (ERT) algorithm is stated as follows. Starting with the trivial tree T consisting of only the source n_0, we grow T by adding a single additional sink into the tree at each iteration. In each step, we seek terminals $u \in T$ and $v \notin T$, such that adding the new edge (u, v) to T will minimize the

[8] In some cases, variants H0 and HBest will produce solutions whose edges cross when embedded in the plane. If a non-self intersecting CSRT is required, these variants are easily modified: H0 can always make the closest connection to n_c that does not cross existing tree edges; similarly, HBest may consider only connections to edges that can be reached without intersecting other edges.

maximum Elmore delay at any leaf of the tree. The algorithm terminates when the tree spans N.[9] A formal description of the ERT algorithm is given in Figure 5.

ERT Algorithm
Input: signal net N with source $n_0 \in N$
Output: routing tree T over N
1. $T = (V, E) = (\{n_0\}, \emptyset)$
2. **while** $\|V\| < \|N\|$ **do**
3. **find** $u \in V$ and $v \notin V$ which minimize the maximum Elmore delay from n_0 to any leaf in the tree $(V \cup \{v\}, E \cup \{(u, v)\})$
4. $V = V \cup \{v\}$
5. $E = E \cup \{(u, v)\}$
6. **output** resulting spanning tree $T = (V, E)$

Figure 5: ERT template: direct incorporation of the Elmore delay formula into a (spanning) tree construction over N.

The ERT algorithm generalizes to yield a *Steiner Elmore routing tree* (SERT) when we allow the new pin to connect to a tree edge, inducing a Steiner point as described in Figure 6. The ERT construction is a special case of the SERT construction where we require either $w = v$ or $w = v'$ in Line 4 of Figure 6.

SERT Algorithm
Input: signal net N with source $n_0 \in N$
Output: Steiner routing tree T over N
1. $T = (V, E) = (\{n_0\}, \emptyset)$
2. $M = N - \{n_0\}$
3. **while** $M \neq \emptyset$ **do**
4. **find** $u \in M$, $(v, v') \in E$, and a new point w which minimizes the maximum Elmore delay from n_0 to any leaf in the tree $(V \cup \{u, w\}, (E - \{(v, v')\}) \cup \{(v, w), (w, v'), (u, w)\})$
5. $V = V \cup \{u, w\}$
6. $E = (E - \{(v, v')\}) \cup \{(v, w), (w, v'), (u, w)\}$
7. $M = M - \{u\}$
8. **output** resulting Steiner tree $T = (V, E)$

Figure 6: SERT template: Steiner generalization of the ERT construction.

Finally, we address the CSRT problem by beginning with a tree containing the single edge (n_0, n_c) in

[9] Our approach should be distinguished from the method of Prasitjutrakul and Kubitz [20] described in Section 2.1 above, wherein A* heuristic search and the actual Elmore delay formula are used in a performance-driven routing tree construction. Like our method, [20] grows a routing tree over a net N starting from the source n_0; they perform A* search of a routing graph (e.g., in building-block design) to find the Elmore delay-optimal Steiner connection from the existing tree to a new sink. However, *the choice of this new sink is forced:* the algorithm always adds the sink that is closest (by Manhattan distance) to the existing tree, and thus falls into the standard pitfall of ignoring the underlying delay criterion. The effect of this difference is apparent in the ERT ordering of added nodes in Figure 4.2 of Section 4 below. Indeed, the method of [20] can yield Elmore delays that are at least twice as large as those of ERT: given a very tall, "hairpin"-like version of Figure 3a with many sinks very closely spaced along the entire hairpin path, [20] forces the sinks to be added into the tree according to the path order (starting from the source n_0 at the lower left), yielding an obviously poor solution. Finally, practical considerations also separate the two methods, e.g., [20] cannot be easily modified to address our CSRT formulation (in the context of the SERT-C discussion to follow, note that given an initial n_0-n_c edge in the tree, the method of [20] may be forced to choose arbitrarily among many ties for the "closest sink" which must be next added into the tree).

Line 1 of the Figure 6 template (i.e., $T = (V, E) = (\{n_0, n_c\}, \{(n_0, n_c)\})$), and then continuing according to the SERT algorithm template with the criterion in Line 3 modified to minimize the Elmore delay $t_{ED}(n_c)$. This yields the SERT-C ("SERT with identified Critical sink") algorithm.

Several points should be noted with regard to implementation of these algorithms:

1. To improve the time complexity of SERT and SERT-C, we consider for each $u \in M$ (Line 4 of Figure 6) only the *shortest* Steiner connection to each edge $(v, v') \in E$. In other words, we choose w so that d_{uw} is as short as possible with $d_{vw} + d_{wv'} = d_{vv'}$. (By the intuition of Figure 3d, ties should be broken to make this shortest connection as close to n_0 on the n_0-v' path, where v is the predecessor of v' in the rooted tree.) Note that we defer the embedding of each tree edge $(v, v') \in E$ for as long as possible, i.e., we maintain it as a bounding box and determine the shortest connection to $u \in M$ with respect to this flexibility.

2. The SERT-C algorithm can be implemented in $O(k^2)$ time, where k is the number of sinks in N. This is shown as follows. For any edge from $u \in M$ to $(v, v') \in E$ with v the predecessor of v' in the rooted tree, there is only one possible connecting point w that we consider. The effect of the (u, w) edge on the delay $t_{ED}(n_c)$ is an *additive constant* no matter when (u, w) might be added into the tree. This delay increment depends on d_{uw} and on either (i) the n_0-w pathlength, when (v, v') is on the n_0-n_c path, or (ii) the n_0-n_a pathlength (where n_a is the lowest common ancestor of n_c and v'), when (v, v') is not on the n_0-n_c path. We can compute the best connection from each sink in M to the initial tree (which is trivial). For each new sink added, at most three new edges will be included into the tree. We simply need to recalculate the effects of connections from nodes $u \in M$ to these new edges (all previously computed effects remain the same), and this requires linear time. We thus obtain the desired $O(k^2)$ complexity.

3. The ERT algorithm can be implemented in $O(k^3)$ time, assuming that unit resistance and capacitance of the interconnect are constant. This fact follows from a simple observation: if a new tree edge incident to sink $u \in V$ (Line 3 of Figure 5) minimizes the maximum Elmore delay $max_i t_{ED}(n_i)$, it must connect u to the sink $v \notin V$ that is closest to u. At each pass through the **while** loop, we update the shortest "outside connections" for every $u \in V$ (a total of $O(k \log k)$ time), and then simply add each of these $O(k)$ outside connections to T in turn, evaluating the Elmore delays to all sinks of the resulting trees in $O(k)$ time per tree (recall Footnote 4). We then choose the outside connection which resulted in the least increase in $max_i t_{ED}(n_i)$. Hence, each pass through the **while** loop requires $O(k^2)$ time, yielding the $O(k^3)$ complexity result. In *practice*, this complexity will be transparent to the user since k is small, and since even standard MST or Steiner global routing will use $\Omega(n^2)$ algorithm implementations. We leave reduction of the ERT complexity as an interesting open issue.

4. We do not know any implementation of the SERT algorithm that is faster than the naive $O(k^4)$ method. Intuitively, the difficulty seems to be that (i) we must always consider $\Theta(k^2)$ Steiner connections from all sinks in M to all edges of T, and (ii) that the connection which minimizes $max_i t_{ED}(n_i)$ may not be the best one from the "perspective" of any individual sink in N or edge T. Thus, we currently have a rather interesting situation where the CSRT problem formulation leads to an algorithm (SERT-C) that enjoys quadratic speedup over the generic Steiner computation (SERT), and where moreover a Steiner tree computation is faster than a spanning tree computation. We believe that the time complexities of both SERT and ERT can each be improved, probably by a factor of k; we leave these as open research directions.

While CS-Steiner began with a minimum-cost Steiner tree and perturbed it to heuristically improve $t(n_c)$, our SERT-C algorithm takes a virtually opposite approach: it starts with the required n_0-n_c connection and grows the routing tree while keeping $t_{ED}(n_c)$ as small as possible. In some sense, SERT-C is the obvious "greedy" approach to the original CSRT formulation of Section 2, when $t(n_i)$ is replaced by $t_{ED}(n_i)$.

We again observe that the promise of the SERT-C approach lies in its consistent, *direct* incorporation of Elmore delay within the construction. Again, this contrasts with heuristics whose objectives or strategies may be only motivated by Elmore delay and whose outputs can therefore remain sensitive to technology, choice of parameters, and input instance.

4 Experimental Results

4.1 CS-Steiner Trees

We implemented each of the CS-Steiner variants H0, H1 and HBest along with the 1-Steiner algorithm [13] using C in the UNIX Sun environment, and ran the algorithms on random 4-, 8- and 16-sink inputs. We also applied our GSR post-processing algorithm (denoted as +U) to 1-Steiner and each of the CS-Steiner variants. Our inputs correspond to two distinct technologies: (i) *IC* is a representative 0.8μ CMOS process, and (ii) *MCM* is typical of current MCM technologies, with lower driver resistance and unit wire resistance.[10]

Table 1 gives delay and tree cost (WL) results and comparisons. The delays at all sink nodes were estimated using the two-pole circuit simulator developed by Zhou and coworkers in [28] [29]. This simulator is a computationally efficient method which has produced very accurate results (less than 10% discrepancy) when tested against the commonly used circuit simulator SPICE over a range of technology and clocking

[10] Specifics of the technology files (IC,MCM): driver resistance = (100,25) Ω; wire resistance = (0.03,0.008) $\Omega/\mu m$; wire inductance = (492,380) $fH/\mu m$; sink loading capacitance = (15.3,1000) fF; wire capacitance = (0.352,0.06) $fF/\mu m$; layout area = $(10^2, 100^2)$ mm^2.

regimes. Each entry in Table 1 represents an average taken over every sink node in 50 random point sets. We emphasize that the 1-Steiner algorithm (or the BRBC, AHHK, etc. methods), being net-oriented, will return the same tree for a given sink set no matter which sink happens to be critical; the delays at the sinks n_i are in some sense "generic". In contrast, each of the three CS-Steiner variants can return a different tree for each choice of critical sink in the same net. Thus, for each variant we report the delay at n_i in the *specific* tree corresponding to the identification of n_i as the critical sink.

		IC			MCM		
		$\|N\| = 5$	$\|N\| = 9$	$\|N\| = 17$	$\|N\| = 5$	$\|N\| = 9$	$\|N\| = 17$
Ave Delay (ns)	1Stein	2.44	3.48	5.26	10.52	15.18	25.77
	1Stein+U	2.26	3.30	4.97	9.43	14.11	24.27
	H0+U	2.37	2.92	3.57	7.26	7.38	7.40
	H1+U	2.20	3.02	3.93	8.90	11.22	13.81
	HBest+U	2.12	2.77	3.47	7.02	7.31	7.35
Ave Delay Ratios	H0+U/1St+U	1.05	.88	.71	.77	.52	.30
	H1+U/1St+U	.97	.92	.79	.94	.80	.57
	HBest+U/1St+U	.94	.84	.70	.74	.52	.30
Ave WL (cm)	1Stein+U	1.51	2.22	3.13	15.65	21.91	31.29
	H0+U	1.95	2.74	3.70	20.35	27.32	36.78
	H1+U	1.58	2.39	3.41	16.20	23.33	33.65
	HBest+U	1.67	2.54	3.58	19.51	26.95	36.59
Wins (%)	HBest+U	51.0	75.5	90.6	81.5	95.5	98.3
	1Stein+U	35.5	12.5	2.3	3.5	0.5	0.1
Nodewise (HBest)/(1St+U) Delay Ratio	min	0.85	0.68	0.53	0.46	0.29	0.16
	ave	0.94	0.85	0.72	0.69	0.50	0.33
	max	1.01	1.00	0.95	0.96	0.87	0.76

Table 1: Routing tree simulation results using IC and MCM technology parameters. Notes: (i) all source and sink locations are chosen randomly in a layout region with grid resolution $25\mu m$; (ii) HBest+U and 1-Steiner+U wins compare only these two heuristics and do not necessarily add up to 100% because of ties; (iii) min (max) Nodewise Delay Ratio gives the geometric mean of, for each net N, the min (max) HBest+U, 1Steiner+U delay ratio over all sinks in N; and (iv) each value in a given column represents an average over the same 50 random signal nets.

Variants H0 and HBest significantly reduce delay to the critical sink, particularly in nets with a large number of sinks and in the MCM technology where output driver and wire resistances are low. In other words, the simple strategy of connecting the critical node via a path with low branching factor is very successful for these cases. Of course, this strategy will produce larger net cost.[11] It is clear that the degree of success achieved by CS-Steiner versus the generic 1-Steiner routing depends on the choice of critical node n_i. For example, in each IC signal net with $|N| = 17$, one expects to find at least one n_i for which the HBest+U delay will be almost 50% less than the 1-Steiner tree delay, while for another n_j, one expects the difference to be only about 5%. With this in mind, we also report the percentage of "wins" between these two algorithms on a node-by-node basis.

The distinct "domains of expertise" of the 1-Steiner and CS-Steiner approaches seem to be correlated with the distance of the critical sink from the source. (We have also observed this phenomenon in all

[11] The "star" strategy can possibly introduce other difficulties such as crossing wires and nodes of degree > 4.

		IC			MCM		
		$\|N\| = 5$	$\|N\| = 9$	$\|N\| = 17$	$\|N\| = 5$	$\|N\| = 9$	$\|N\| = 17$
Rank of Sink	1	0.90	0.84	0.83	0.51	0.44	0.45
	2	0.91	0.78	0.76	0.62	0.37	0.34
	3	0.96	0.80	0.68	0.75	0.44	0.27
	4	0.99	0.80	0.68	0.95	0.44	0.28
	5		0.82	0.66		0.47	0.30
	6		0.87	0.68		0.54	0.26
	7		0.91	0.67		0.63	0.28
	8		0.96	0.67		0.78	0.28
	9			0.67			0.27
	10			0.70			0.29
	11			0.71			0.31
	12			0.72			0.34
	13			0.72			0.34
	14			0.77			0.38
	15			0.80			0.43
	16			0.86			0.50

Table 2: Geometric mean ratio of HBest+U to 1Stein+U delay to sinks, sorted by their distance to the source. (Sink 1 is closest to the source.)

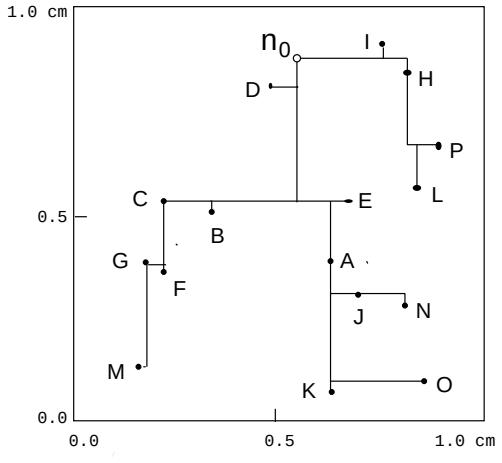
other comparisons of CS-Steiner against other global routing algorithms, e.g., the BRBC method.) Table 2 illustrates this correlation between the HBest+U and 1Stein+U methods. In the table, the ratio between HBest+U and 1Stein+U delays is reported on a sink-by-sink basis, where sinks are sorted by distance from the source. Critical-sink routing is most successful for critical sinks closer to the source, except if n_c is the very closest sink to the source. CS-Steiner also provides larger wins in the MCM technology.[12] Figure 4.1 provides a nodewise delay comparison between H1+U and a min-cost SPT (as might be produced by the AHHK algorithm with $c = 0.999$ or by the methods of [21]) for a single net with 16 sinks. This comparison supports the results of Table 2: in general, the H1+U tree offers a significant delay improvement for critical sinks located close to the source, but if the critical sink is far away, the SPT seems a better choice.

4.2 Elmore Routing Trees

We constructed Elmore routing trees for the same sets of random inputs used in the CS-Steiner experiments. Delay simulation results are presented in Table 3. For purposes of comparison, the table includes data from the minimum spanning tree, AHHK tree (quoted from [1], and 1-Steiner tree constructions.

Our results show that even as generic net-dependent routers, the ERT methods we propose are highly effective, beyond their relative efficiency and ease of implementation. For nets with 16 sinks, the spanning tree ERT construction reduces average sink delay versus the MST construction by 33% in the IC technology and by 69% in the MCM technology. The ERT algorithm also improves upon AHHK, with reductions of 10% (IC) and 43% (MCM). These results are particularly impressive because the implementation of AHHK

[12] Recall the intuition above, namely that critical-path routing will be useful if the technology favors star-like, "direct" connections to the n_i. Note also that timing-driven placement algorithms may tend to place the critical sink n_c close to the source.



Sink	(H1+U)/(SPT) Delay		Ratio
A	3.39/4.49	=	0.679
B	3.34/4.88	=	0.684
C	3.63/5.07	=	0.716
D	2.31/2.87	=	0.805
E	3.70/4.57	=	0.810
F	4.44/5.28	=	0.841
G	4.51/5.31	=	0.849
H	2.54/2.88	=	0.882
I	2.53/2.77	=	0.913
J	4.84/5.19	=	0.932
K	5.27/5.46	=	0.965
L	3.08/3.11	=	0.990
M	5.84/5.50	=	1.062
N	5.88/5.24	=	1.122
O	7.69/5.61	=	1.371
P	5.40/3.10	=	1.742

Figure 7: Random 16-sink IC example (1cm by 1cm layout region), showing nodewise delay ratios of H1+U output versus a heuristic minimum-cost SPT; this SPT is shown in the figure.

in [1] simulates delays for output trees for 21 different values of the c parameter, and then chooses the best tree found for each signal net instance.

The Steiner tree version of our ERT method also performs well as a generic high-performance router. For signal nets with 16 sinks, SERT improves average sink delay versus the 1-Steiner routing by 21% and 62% for the IC and MCM technologies, respectively. For 8-sink nets, average delays are reduced by 10% for IC and 42% for MCM. The percentage reductions in maximum delay are even greater. It should be noted that for the MCM technology, the ERT and SERT constructions tend to be star-like, producing tree costs much higher than those of the 1-Steiner construction. In practice, when delay is not an overriding concern, wirelength can be "recaptured" by increasing the length of the virtual edge between n_0' and n_0 in the ERT and SERT algorithms (i.e., by simulating a larger output driver resistance), if such a heuristic seems desirable.

Finally, even more significant reductions in delay can be achieved when a critical sink has been identified, per our original CSRT formulation. The SERT-C algorithm improves over the SERT results by an *additional* reduction in delay at the critical sink of 10% for IC's and 7% for MCM's. Identification of a critical sink has clear advantages in terms of tree cost, particularly for MCM routing: the SERT-C trees have much less cost than the SERT outputs, while still improving the delay to the critical sink. Finally, we note that the SERT-C router compares very favorably to the HBest variant of CS-Steiner discussed in the subsection above. SERT-C produces slightly better delay and tree cost values for the IC technology, and only slightly worse results for the MCM technology. Note that SERT-C is more practical than HBest, in that it does not use the circuit simulator during construction of the tree.[13] Node-by-node comparisons (Table 4) of SERT-C

[13] With respect to practicality: the average times in CPU seconds for SERT-C and the distributed RCL delay simulator on a Sun Sparc IPC were respectively .0012 and .031 seconds for $|N| = 5$; .017 and .049 for $|N| = 9$; .31 and .089 for $|N| = 17$. This reflects the "all-purpose" implementation of our package, which has $\Theta(k^4)$ complexity for each of ERT, SERT and SERT-C. We stress that SERT-C can be implemented to run in $O(k^2)$ time, as described above.

v		IC			MCM		
		$\|N\| = 5$	$\|N\| = 9$	$\|N\| = 17$	$\|N\| = 5$	$\|N\| = 9$	$\|N\| = 17$
Ave Delay (ns)	MST	2.92	4.35	6.46	12.38	18.72	29.57
	AHHK	2.64	3.53	4.77	9.94	12.39	16.26
	ERT	2.43	3.24	4.31	7.49	8.16	9.29
	1Stein+U	2.26	3.30	4.97	9.43	14.11	24.27
	SERT	2.22	2.97	3.91	7.49	8.16	9.29
	SERT-C	2.12	2.70	3.43	7.26	7.39	7.41
Ave Delay Ratios	ERT/MST	.83	.74	.67	.60	.44	.31
	ERT/AHHK	.92	.92	.90	.75	.66	.57
	SERT/1St+U	.98	.90	.79	.79	.58	.38
	SERT-C/1St+U	.94	.82	.69	.77	.52	.31
Max Delay (ns)	MST	3.75	5.69	8.75	17.28	27.75	45.83
	AHHK	3.30	4.48	6.04	13.88	18.70	24.75
	ERT	3.14	4.07	5.40	13.06	16.18	18.80
	1Stein+U	2.89	4.28	6.54	14.93	20.81	37.38
	SERT	2.83	3.71	4.90	13.06	16.18	18.79
Max Delay Ratios	ERT/MST	.84	.72	.62	.75	.58	.41
	ERT/AHHK	.95	.91	.89	.94	.87	.76
	SERT/1St+U	.98	.87	.75	.87	.78	.50
Ave WL (cm)	MST	1.69	2.47	3.49	17.07	24.43	34.88
	AHHK	1.89	2.87	4.10	20.02	29.43	44.22
	ERT	2.00	3.04	4.41	28.21	54.80	103.67
	1Stein+U	1.51	2.22	3.13	15.65	21.91	31.29
	SERT	1.67	2.61	3.70	28.21	54.80	103.67
	SERT-C	1.71	2.48	3.45	20.35	27.52	37.07
Wins (%)	CERT-C	52.5	84.0	90.0	80.0	93.3	98.0
	1Stein+U	21.0	11.0	7.0	12.5	3.7	1.3
Nodewise CERT-C/(1St+U) Delay Ratio	min	0.84	0.68	0.54	0.42	0.30	0.16
	ave	0.92	0.82	0.72	0.64	0.51	0.33
	max	0.99	0.95	0.97	0.93	0.89	0.78

Table 3: Delay simulation results for Elmore routing tree outputs, compared with the 1-Steiner heuristic using IC and MCM technology parameters (see notes contained in Table 1 caption). Note that data for AHHK are quoted from [1] and are computed using values relative to MST for the same technology parameters: a small error is possible since the respective works did not use the exact same sets of 50 random nets.

versus 1-Steiner are qualitatively the same as for the HBest algorithm in Table 2.

Figures 4.2 and 4.2 illustrate the SERT and SERT-C algorithms for one of the random signal nets used in our simulations, with the IC technology parameters. Figure 4.2 shows the progressive growth of the SERT construction. Figure 4.2 contains the trees produced by SERT-C for each choice of critical node. Note that there are redundancies: the tree constructed when node 3 or node 7 is critical is also the 1-Steiner tree, and the tree constructed when node 8 is critical is the same as the generic SERT output. The critical delays and tree costs of these constructions are shown in Table 5. Note that the sink delays produced by SERT-C improve over the SERT result for all but three of the sinks, and are worse only for sink 3. For the two sinks furthest from the source, SERT and SERT-C produce identical delays. These relative differences in delays corroborate the results of Tables 2 and 4, where path-dependent global routing performs best compared to net-dependent routing when the critical sinks are at a medium distance from the source, or are close (but not too close) to the source.

		IC			MCM		
		$\|N\| = 5$	$\|N\| = 9$	$\|N\| = 17$	$\|N\| = 5$	$\|N\| = 9$	$\|N\| = 17$
Rank of Sink	1	0.91	0.89	0.89	0.49	0.48	0.48
	2	0.88	0.81	0.82	0.55	0.39	0.34
	3	0.92	0.79	0.72	0.68	0.44	0.28
	4	0.97	0.77	0.73	0.90	0.44	0.29
	5		0.77	0.68		0.48	0.31
	6		0.80	0.69		0.55	0.26
	7		0.84	0.69		0.63	0.29
	8		0.88	0.65		0.79	0.28
	9			0.67			0.28
	10			0.69			0.29
	11			0.69			0.31
	12			0.70			0.34
	13			0.67			0.34
	14			0.72			0.38
	15			0.74			0.44
	16			0.78			0.50

Table 4: Geometric mean ratio of Critical SEST to 1Stein+U delays, sorted by distance of the sink to the source. (Sink 1 is closest to the source.)

Critical Node	SERT Delay	SERT-C Delay	SERT-C Cost
2	1.81	1.66	2.37
3	1.89	1.97	2.27
4	1.96	1.68	2.37
5	2.30	2.16	2.80
6	2.18	1.95	2.55
7	2.40	2.27	2.27
8	2.42	2.42	2.47
9	2.82	2.82	2.46

Table 5: Delays and costs for trees constructed by the SERT-C algorithm and compared with the SERT algorithm for the 9-pin signal net in Figures 8 and 9. (Delays are in nanoseconds and costs are in centimeters. Sinks labels are sorted according to distance from the source.)

5 Conclusions

We have addressed a *critical-sink routing tree* (CSRT) formulation which arises when critical-path information becomes available during the timing-driven layout proces. The CSRT formulation is orthogonal to previous formulations that required prescribed delay bounds at the sinks of a signal net. Thus, the CSRT problem facilitates simple, fast solution methods which avoid, for example, iterative determination of appropriate delay bounds.

Two new classes of CSRT constructions are proposed: (i) the CS-Steiner method, which perturbs a minimum Steiner tree to accomodate an identified critical sink, and (ii) the SERT-C method, which begins with a connection from the source to the critical sink and then grows a tree so as to minimize the increase in Elmore delay to the critical sink. Each of these algorithms is efficient, and offers very significant performance

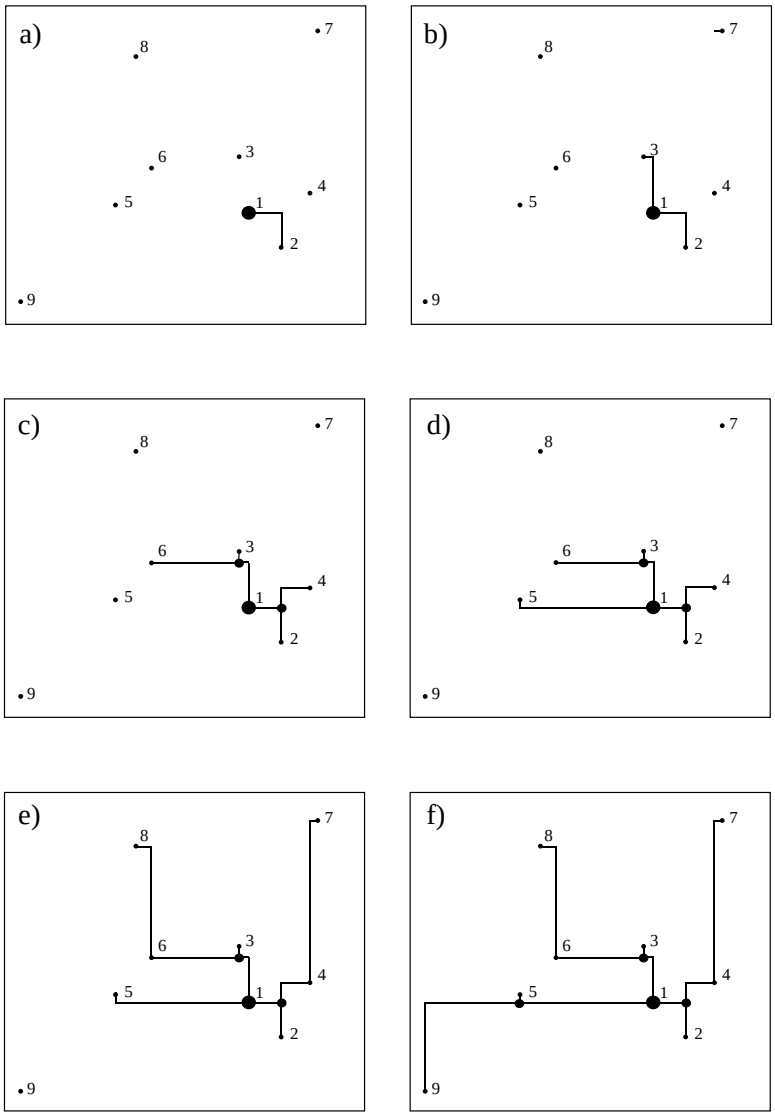


Figure 8: Example of the progressive construction of a 9-terminal net by the SERT algorithm on one of the 50 random examples used in our simulations using IC parameters. The pin labelled 1 is the source, and sinks are numbered in order of distance from the source.

improvements over existing performance-driven routing tree constructions. We note that the greedy "Elmore routing tree" (ERT) approach underlying the SERT-C algorithm seems quite powerful. In particular, ERT generalizes to a "generic" SERT Steiner router which outperforms all previous performance-driven routing algorithms. The ERT approach is also the first to *consistently*, and directly, optimize the Elmore delay formula itself, rather than an objective which heuristically abstracts Elmore delay. Since Elmore routing trees are efficiently computed, our approaches may lead to basic new utilities that can be integrated within performance-driven global routing packages.

Current work addresses extensions of the critical path-dependent routing tree design to the general case of multiple critical sinks with varying criticalities. If a subset of the sinks are designated as critical, the

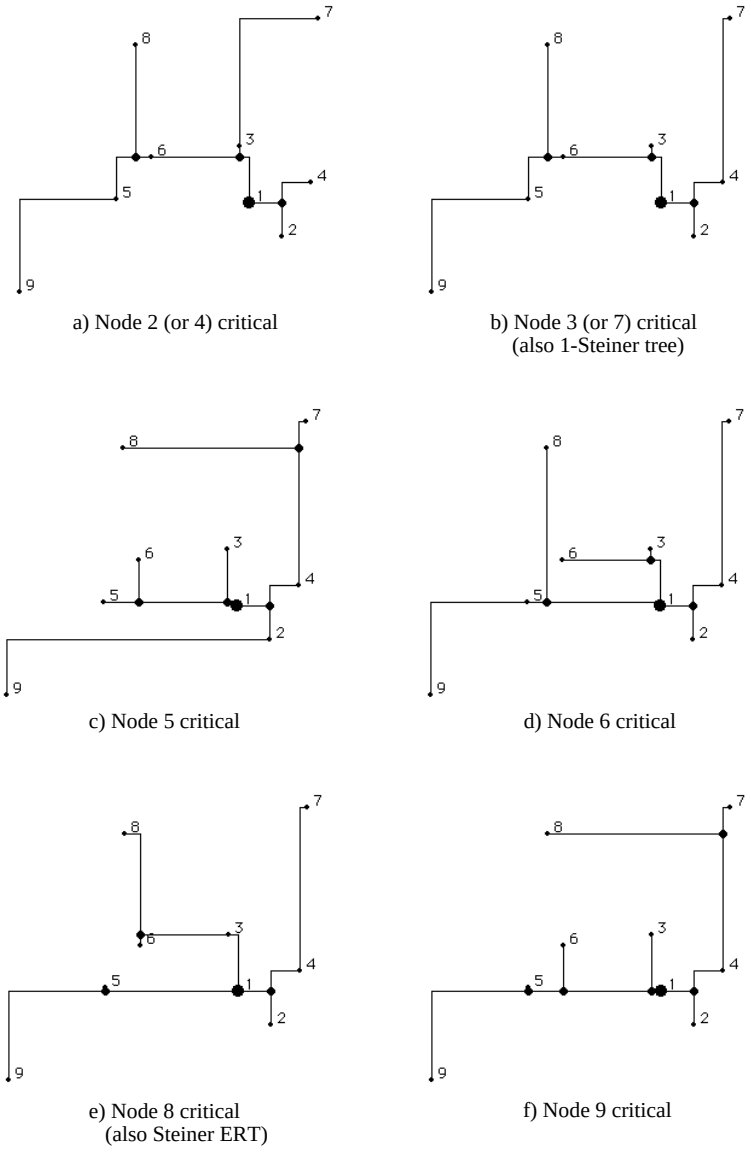


Figure 9: SERT-C tree constructions for a single 9-pin net.

SERT-C algorithm can be extended by first routing the critical sinks under the min-max delay objective of SERT, then connecting non-critical sinks as in SERT-C to minimize the weighted sum of the delays at the critical sinks. There are also interesting extensions of the CS-Steiner and ERT algorithms to general-cell layout with arbitrary routing region costs. Finally, we leave as an open problem the reduction in time complexity of the ERT constructions.

6 Acknowledgements

We are grateful to the authors of [1] for providing an early version of their manuscript, as well as to the authors of [28] [29] for use of their simulator code.

Appendix: Global Slack Removal

Recall from Section 3.1 that *Global Slack Removal* (GSR) is a linear-time postprocessing enhancement to the CS-Steiner approach. GSR shifts edges in the 1-Steiner output to maximize the monotonicity of all source-sink paths without any increase in total tree cost or Elmore delay to any sink. In what follows, we use the term *1-Steiner tree* to refer to any tree that can be output by the 1-Steiner algorithm. If we orient a 1-Steiner tree T by rooting it at the source n_0, a U is defined to consist of three consecutive edges v_1v_2, v_2v_3 and v_3v_4 on a root-leaf path in T such that the v_1-v_4 pathlength is greater than the Manhattan distance d_{14} (Figure 10(a)). The GSR algorithm (Figure 11) takes as input a rooted (1-)Steiner tree T and removes U's as shown in Figure 10(b); the input tree is processed in top-down order.[14] Three clarifying points should be noted.

1. GSR utilizes a queue which can be implemented arbitrarily as long as each node in the tree is processed before its children. In practice, a depth-first ordering is simplest.

2. The current node in the traversal is checked to see if it is the *third* node in a U. This detail is necessary to ensure that the output tree has no remaining U's.

3. Finally, notice that all *low-degree* Steiner nodes (i.e., of degree ≤ 2; these are clearly superfluous) are removed at two separate points in the algorithm. This is for two reasons: (a) more U's can be found if all low-degree Steiner nodes are removed at the outset, and (b) each removal of a U can introduce additional low-degree Steiner nodes, so low-degree Steiner nodes should again be eliminated at the end of the algorithm.

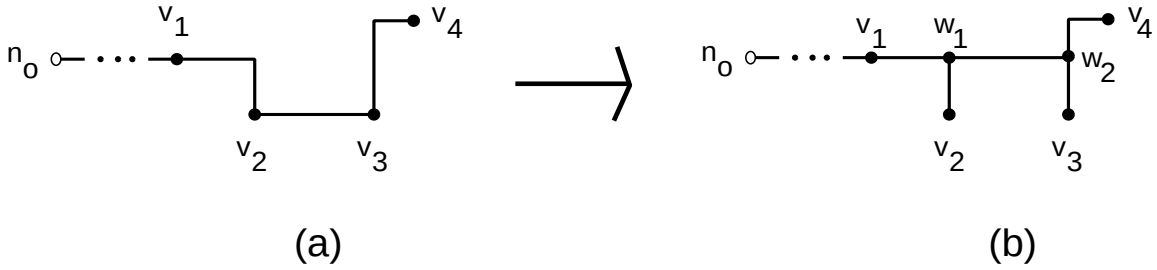


Figure 10: Removing a single "U" in the GSR Algorithm.

We now prove two results showing that the GSR algorithm returns a tree with no U's in linear time, and that this output tree dominates the input Steiner tree in each of three parameters: total tree cost, pathlength from the source to each sink, and Elmore delay at each sink.

Theorem 1: Given a 1-Steiner tree T as input, GSR will return a tree T' such that (i) T' has the same or less total cost as T; (ii) $\forall n_i$, $i = 1, \ldots, |k|$, the n_0-n_i pathlength in T' is less than or equal to the n_0-n_i

[14] The GSR algorithm can actually be used on any Steiner tree to reduce source-sink pathlengths without increasing wirelength. However, GSR is guaranteed to return a tree with no remaining U's only when no single Steiner node can be added to the input tree so as to reduce wirelength, i.e., when the input is a 1-Steiner tree.

GSR Algorithm
Input: 1-Steiner tree T with source n_0
Output: Steiner tree T with all U's removed
1. **remove** all Steiner nodes of degree ≤ 2 from T;
2. $Q \leftarrow \{n_0\}$;
3. **while** $Q \neq \emptyset$
4. $v_3 \leftarrow Dequeue(Q)$;
5. $v_2 \leftarrow pred(v_3)$;
6. $v_1 \leftarrow pred(v_2)$;
7. **for each** node $v_4 \in succ(v_3)$ **do**
8. $Q \leftarrow Enqueue(v_4)$;
9. **if** path $v_1v_2v_3v_4$ is a U
10. **remove** the U as in Figure 10;
11. **insert** Steiner nodes w_1 and w_2 into T;
12. **remove** all Steiner nodes of degree ≤ 2 from T;

Figure 11: Pseudo-code for the GSR algorithm. Local variables include the queue Q and nodes v_1 - v_4. Here, $pred(v)$ denotes the predecessor of node v when the tree T is rooted at n_0. Similarly, the set $succ(v)$ denotes the set of all nodes with predecessor v in the rooted tree.

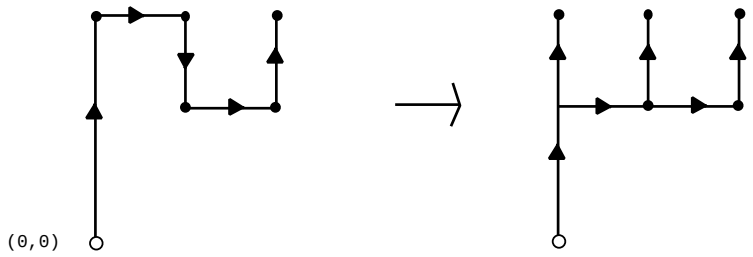
pathlength in T; and (iii) the Elmore delay $t_{ED}(n_0, n_i)$ at each n_i in T' is less than or equal to the Elmore delay $t_{ED}(n_0, n_i)$ in T.

Proof: (i) In Figure 10, the only change between (a) and (b) in terms of wirelength is the deletion of the edge v_2v_3 and the insertion of edge w_1w_2, both of which have the same length. Consequently, removing the U does not immediately change the wirelength of the tree. If, however, either v_2 or v_3 becomes a Steiner node of degree 1 in Figure 10(b), then T' will have less total wirelength than T after all low-degree Steiner nodes are removed.

(ii) In Figure 10, it is obvious that a single U removal can only affect the pathlengths from the source to sinks in the tree rooted at v_1. In fact, since the U removal does not affect the source-sink pathlengths for v_2 and v_3, and moreover reduces the pathlength for v_4, the only pathlengths changed are those for sinks in the tree rooted at v_4 (and these are reduced).

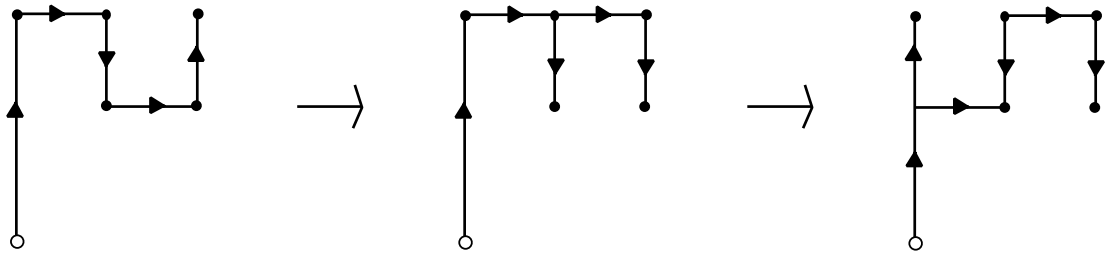
(iii) Note first that Elmore delay along a path depends on the total capacitance (i.e. wirelength) along the path as well as the wirelength of any subtree branching off from the path. If a subtree is moved so that it has the same total wirelength and meets the path closer to the source, then it will reduce the capacitance along a part of the path, and thereby reduce the total delay between the end points of the path. With this in mind, we see in Figure 10 that the delay to node v_2 is reduced because the tree capacitance that met the n_0-v_2 path at v_2 in (a) now meets the same path at w_1 in (b). For v_3, the capacitance that met the n_0-v_3 path at v_3 now meets the path at w_1 and w_2. The argument is essentially the same for node v_4, except that additionally the n_0-v_4 pathlength is reduced in (b). □

We note that the order in which U's are removed from the tree is important. If the U's are processed in a bottom-up order instead of a top-down order, then new U's can be introduced and the resulting tree



(a)

Removal of U's in
top-down order



(b)

Removal of U's in
bottom-up order

Figure 12: An example for which processing U's in a bottom-up order (b) returns a tree with one remaining U. Processing U's in a top-down order (a) is guaranteed to remove all U's.

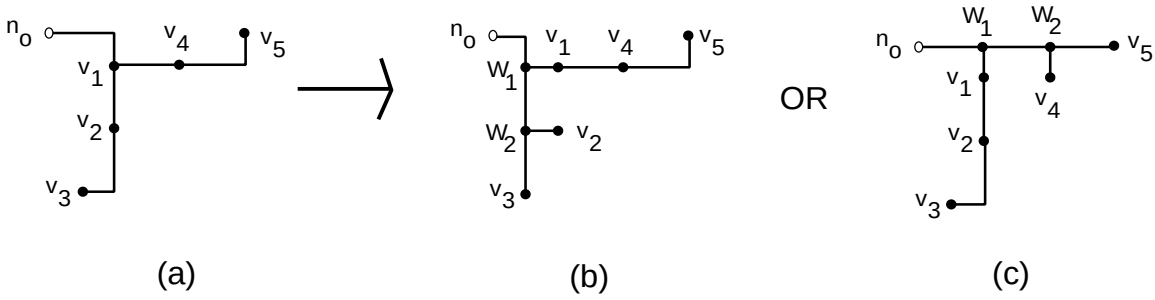


Figure 13: The GSR algorithm with input (a) can produce either tree (b) or tree (c), depending on the order in which the U's are processed.

may not have all of its U's removed. An example of this is seen in Figure 12. Furthermore, two different top-down orderings can produce different output trees (although both will have no remaining U's). Two different trees that could be produced by the GSR algorithm from the same tree are illustrated in Figure 13.

The following three lemmas are useful in showing the correctness of GSR in the sense of producing a tree with no remaining U's. Lemma 1 is a basic property of a 1-Steiner tree, while Lemmas 2 and 3 follow easily from Lemma 1.

Lemma 1: Adding a single Steiner node to a 1-Steiner tree cannot reduce the cost of the tree.

Lemma 2: In any 1-Steiner tree, the middle edge of a U must be either horizontal or vertical (i.e., not "L"-shaped).

Lemma 3: Each edge in a 1-Steiner tree is the middle edge in at most one U.

We now show:

Theorem 2: GSR runs in time linear in the size of T and returns a tree T' which contains no U's.

Proof: The algorithm checks for the existence of a U at most once at each edge in the input tree (corresponding to edge v_3v_4 in Figure 10), and testing for and removing each U requires constant time. Hence, GSR runs in linear time. To see that T' contains no U's, we ask whether the removal of the U $v_1v_2v_3v_4$ as in Figure 10 can create any new U's which are not inspected by the algorithm later on. Consider which nodes could be the third node in such a newly introduced U. Obviously, no node outside the subtree rooted at v_1 could be such a node, because the paths to these nodes are not affected by the single U removal. Other cases of nodes which could be such a third node include v_1, w_1, w_2, v_2, and nodes in the subtree rooted at v_2. Lemmas 1, 2 and 3 suffice to show that none of these nodes can be the third node of a new U. All other nodes (i.e., those in the original subtree rooted at v_3 and v_4) are yet to be processed. Hence, the Theorem follows by induction on the depth of v_3. □

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", *technical report* CSD-920051, UCLA Department of Computer Science, 1992.

[2] B. Awerbuch, A. Baratz and D. Peleg, "Cost-Sensitive Analysis of Communication Protocols", *Proc. ACM Symp. on Principles of Distributed Computing*, 1990, pp. 177-187.

[3] J. P. Cohoon and L. J. Randall, "Critical Net Routing", *Proc. IEEE Intl. Conf. on Computer Design*, 1991, pp. 174-177.

[4] J. Cong, A. B. Kahng, G. Robins, M. Sarrafzadeh, and C. K. Wong, "Performance-driven global routing for cell based IC's", *Proc. Intl. Conf. on Computer Design*, pp. 170-173, 1991.

[5] J. Cong, A. B. Kahng, G. Robins, M. Sarrafzadeh, and C. K. Wong, "Provably Good Performance-Driven Global Routing", *IEEE Trans. on CAD* 11(6), June 1992, pp. 739-752.

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

[7] 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", *Proc. ACM/IEEE Design Automation Conf.*, 1990, pp. 84-89.

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

[9] W. C. Elmore, "The Transient Response of Damped Linear Network with Particular Regard to Wideband Amplifiers", *J. Applied Physics* 19 (1948), pp. 55-63.

[10] P. S. Hauge, R. Nair and E. J. Yoffa, "Circuit Placement for Predictable Performance", *Proc. IEEE Intl. Conf. on Computer-Aided Design*, 1987, pp. 88-91.

[11] J.-M. Ho, G. Vijayan and C. K. Wong, "New Algorithms for the Rectilinear Steiner Tree Problem", *IEEE Transactions on Computer-Aided Design*, 9(2), 1990, pp. 185-193.

[12] M. A. B. Jackson and E. S. Kuh, "Estimating and Optimizing RC Interconnect Delay During Physical Design", *Proc. IEEE Intl. Conf. on Circuits and Systems*, 1990, pp. 869-871.

[13] A. B. Kahng and G. Robins, "A New Class of Iterative Steiner Tree Heuristics with Good Performance", *IEEE Transactions on CAD* 11(7), July 1992, pp. 893-902.

[14] S. Khuller, B. Raghavachari and N. Young, "Balancing Minimum Spanning and Shortest Path Trees", *Proc. ACM/SIAM Symp. on Discrete Algorithms*, January 1993, to appear.

[15] E. Kuh, M. A. B. Jackson and M. Marek-Sadowska, "Timing-Driven Routing for Building Block Layout", *Proc. IEEE International Symposium on Circuits and Systems*, pp. 518-519, 1987.

[16] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*, Berlin, Wiley-Teubner, 1990.

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

[18] M. Marek-Sadowska and S. Lin, "Timing Driven Placement", *Proc. IEEE Intl. Conf. on Computer-Aided Design*, 1989, pp. 94-97.

[19] J. Pearl, *Heuristics: Intelligent Search Strategies for Computer Problem Solving*, Reading, MA: Addison-Wesley, 1984.

[20] S. Prasitjutrakul and W. J. Kubitz, "A Timing-Driven Global Router for Custom Chip Design", *Proc. IEEE Intl. Conf. on Computer-Aided Design*, 1990, pp. 48-51.

[21] S. K. Rao, P. Sadayappan, F. K. Hwang and P. W. Shor, "The Rectilinear Steiner Arborescence Problem", *Algorithmica* 7 (1992), pp. 277-288.

[22] G. Robins, "On Optimal Interconnections", Ph.D. thesis (*technical report* CSD TR-920024), CS Department, University of California, Los Angeles, June 1992.

[23] J. Rubinstein, P. Penfield, and M. A. Horowitz, "Signal Delay in RC Tree Networks", *IEEE Trans. on CAD* 2(3) (1983), pp. 202-211.

[24] A. Srinivasan, K. Chaudhary and E. S. Kuh, "RITUAL: A Performance Driven Placement Algorithm for Small-Cell ICs", *Proc. IEEE Intl. Conf. on Computer-Aided Design*, 1991, pp. 48-51.

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

[26] S. Teig, R. L. Smith and J. Seaton, "Timing Driven Layout of Cell-Based ICs", *VLSI Systems Design*, May 1986, pp. 63-73.

[27] R. S. Tsay, "Exact Zero Skew", *Proc. IEEE Intl. Conference on Computer-Aided Design*, 1991, pp. 336-339.

[28] D. Zhou, F. P. Preparata and S. M. Kang, "Interconnection Delay in Very High-speed VLSI", *IEEE Trans. on Circuits and Systems* 38(7), 1991.

[29] D. Zhou, S. Su, F. Tsui, D. S. Gao and J. Cong, "Analysis of Trees of Transmission Lines", *technical report* UCLA CSD-920010.