Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596

ROUTING TREE CONSTRUCTIONS FOR CRITICAL PATH OPTIMIZATION

K. D. Boese
A. B. Kahng

September 1992
CSD-920039

Routing Tree Constructions for Critical Path Optimization *

Kenneth D. Boese and Andrew B. Kahng

Abstract

We address routing tree constructions for VLSI global routing which exploit available information about *critical paths* between primary inputs and primary outputs of the design. In this snapshot of our ongoing work, we describe five heuristic variants of our Critical-Sink Routing Tree (CSRT) approach and compare them with the 1-Steiner (1Stein) algorithm of Kahng and Robins [11]. Each of the five heuristics is a modification of 1Stein designed to minimize the delay from the source to a single sink in a net. We simulate the nets constructed by these methods on real-world IC and MCM parameters using the 2-pole simulator of Zhou et al. [23]. This simulator has been shown to be very accurate when tested against the circuit simulator SPICE. We also present a Global Slack Removal (GSR) method that reduces source-sink pathlengths in Steiner trees produced by 1Stein without increasing total wirelength. The results of our simulations indicate that the smallest delays are produced by our heuristic, called HBest, which connects non-critical sinks via the 1-Steiner algorithm and uses the 2-pole simulator to determine how to best connect the critical sink. Another successful heuristic simply connects the critical sink with an edge directly to the source. Future directions for our research include the investigation of methods that allow multiple critical nodes with differing priorities.

1 Introduction

This report gives a snapshot of ongoing research in routing tree design for high-performance integrated circuits and multi-chip modules. Our work is predicated on a simple observation: while there are many timing-driven module placement algorithms, there are no complementary global routing methods which can explicitly honor the timing-driven placement, or exploit the timing information that becomes available during the placement process. Thus, there is a fundamental mismatch between placement and routing in current "performance-driven" physical design methodologies.

When performance becomes a dominant system criterion, static timing analysis is iteratively used to determine necessary changes to the module placement and global routing: this

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

is the basic "performance-driven" approach to layout. We observe that critical-path information may become available in such an iterative process, and moreover that this information may be directly used in the the design of routing trees.

Existing performance-driven placement methods are essentially of two basic flavors.

1. *Net-dependent* algorithms typically use centroid-connected star cost [20], probabilistic estimates of Steiner tree cost [10], minimum spanning tree cost [5] or the bounding box half-perimeter [14] to estimate wire capacitance and signal delay for a multi-terminal net. Based on this timing analysis, module placements are updated to reduce these "net-based" objective functions for those signal nets which lie along the critical path.

2. *Path-dependent* methods are distinguished by their consideration of delay between the source and a particular *critical sink* of a multi-terminal net. The critical sink is determined via timing analysis using known module delays and estimated path delays. For example, Lin and Du [13] 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. [8] and Teig et al. [21].

For any net on a timing-critical path, the path-dependent approach affords an explicit routing constraint with respect to the delay to the net's critical sink. Arguably, net-dependent methods only provide implicit routing constraints, e.g., by limiting the bounding box half-perimeter. However, even with a net-dependent placement methodology it is easy to identify critical sinks after the timing analysis has been performed, or *a priori* by finding paths in the design that contain more module delays. Given a placement of net $N = \{n_0, n_1, ..., n_k\} \subset \Re^2$, with source n_0 and one or more identified critical sinks of possibly varying priorities, our goal is to construct a *critical-path dependent* routing tree $T(N)$ such that the weighted sum of the delays at the critical sinks is minimized. In this report, we concentrate on the special case when exactly one critical sink n_c is specified in the net N.

The remainder of this report is organized as follows. In Section 2, we survey existing works in the performance-driven global routing literature, and describe a few motivating observations concerning desired qualities of a "performance-driven" interconnection tree under the distributed RC delay model (specifically, the first-order moment of the impulse response, also known as *Elmore delay*). Section 3 then presents our critical-sink routing tree (CSRT) ap-

proach, embodied by five simple variations of minimum Steiner tree constructions; note that the traditional heuristic minimum Steiner tree approach for global routing remains central in our work by virtue of the observations of Section 2 concerning RC delay. Experimental results are presented in Section 4, comparing delays at critical sinks in our heuristic tree topologies with analogous delays obtained using the best-performing efficient Steiner tree heuristic [11]; these results show that the CSRT approach has considerable promise.

2 Performance-Driven Routing Tree Design

2.1 Previous Approaches

This subsection briefly reviews several existing works in the performance-driven global routing literature. In 1984, Dunlop et al. [6] used the static timing analysis performed within the iterative place/route paradigm to directly yield net priorities; the assumption behind their work was that nets which are routed earlier will enjoy fewer detours and fewer feedthroughs, thus enhancing performance. Subsequently, Kuh, Jackson and Marek-Sadowska [12] gave an approach tuned to hierarchical building-block layouts; Prastjutrakul and Kubitz [16] used A* heuristic search [15] for a similar problem domain. A more general approach was given by Cong, Kahng, Robins, Sarrafzadeh and Wong [3] [4], wherein a parameter ϵ was used to trade off between the total wirelength of the routing tree (i.e., cost) and the longest source-sink path in the tree (i.e., radius). In [4], "provably good" method was proposed; this method, called the BRBC (bounded-radius, bounded-cost) algorithm, afforded both radius and cost simultaneously within constant factors of optimal (cf. the "shallow-light" construction of Awerbuch et al. [1]). The BRBC approach is interesting because its radius-cost tradeoff may also be viewed as one between (i) the shortest-path tree (SPT) construction wherein all source-sink paths are shortest-possible (i.e., monotone staircase routes in the Manhattan plane, yielding the notion of minimum radius) and (ii) the minimum spanning (Steiner) tree construction, which is the traditional routing tree construction (see [11] for a survey of the Steiner tree literature in VLSI CAD). With this in mind, we reproduce two algorithm templates: Figure 1 gives the BRBC algorithm [4] and Figure 2 gives the 1-Steiner algorithm [11].

2.1.1 The BRBC Algorithm

The BRBC algorithm [4] is stated in Figure 1. In the template, we use G to denote the complete graph of distances among the n_i, $i = 0, 1, \ldots, k$ that implicitly underlies the routing tree problem. We also use MST_G and SPT_G to respectively denote the minimum spanning tree over G and the shortest-paths tree over G. Terms such as *cost*, *minpath* and *dist* are defined in the obvious manner. Finally, ϵ is a user-specified parameter of the algorithm, where $\epsilon = \infty$ will allow BRBC to return a minimum spanning tree, and $\epsilon = 0$ in some sense allows BRBC to return a shortest-paths tree.[1]

```
compute MST_G and SPT_G
    /* e.g., using Prim's and Dijkstra's algorithms, respectively */
Q = MST_G
L = depth-first tour of MST_G
S = 0
for i = 1 to |L| - 1
    S = S + cost(L_i, L_{i+1})
    if S ≥ ε · dist_G(s, L_{i+1}) then
        Q = Q ∪ minpath_G(s, L_{i+1})
        S = 0
T = shortest-paths tree of Q
```

Figure 1: Computing a bounded-radius spanning tree T for $G = (N, E)$, with source $n_0 \in N$ and radius $R \equiv max_i\ dist(n_0, n_i)$, using parameter ϵ. T will have radius at most $(1 + \epsilon) \cdot R$, and cost at most $(1 + \frac{2}{\epsilon}) \cdot cost(MST_G)$.

2.1.2 The 1-Steiner Algorithm

The 1-Steiner algorithm of Kahng and Robins [11] is stated in Figure 2.

For a given point set P, a 1-*Steiner point* is any point x such that $cost(MST(P \cup \{x\}))$ is minimized, with $cost(MST(P \cup \{x\})) < cost(MST(P))$. 1-Steiner is the best-performing heuristic known for the NP-complete minimum rectilinear Steiner tree problem. The algorithm iteratively adds interior nodes that make the largest reduction to the wirelength of a Steiner tree spanning the net N. Because there are essentially only $|N|^2$ candidate interior

[1] It should be noted that BRBC extends to perform Steiner routing, e.g., in the Hanan grid [11] which will contain all distinct Steiner routing trees, modulo only displacements of either internal nodes (Steiner points) or wire edges which do not affect tree cost / delay performance. However, the extension to Steiner routing in [4] fails to exploit the ability to "overlap" tree edges to save wire in the Manhattan geometry, and thus post-processing methods such as that in [9] may be also needed to achieve a low-cost Steiner tree.

Iterated 1-Steiner: Steiner tree heuristic of [11]
Input: N, a placement of source n_0 and k sinks **Output:** Heuristic minimum rectilinear Steiner tree over N
$S = \emptyset$ **While** $\|S\| < k + 1$ and $\exists$ 1-Steiner point x **Do** $S = S \cup \{x\}$ **Output** MST($P \cup S$)

Figure 2: The iterated 1-Steiner algorithm.

nodes, the 1-Steiner algorithm is relatively efficient, and can be implemented to have worst-case time complexity $O(|S| \cdot |N|^2)$ where S is the set of added Steiner points. Below, we use the "batched" version of the 1-Steiner algorithm, in which several Steiner points may be added during a single $O(|N|^2 log|N|)$ phase, so long as these Steiner points are "noninterfering". Because the 1-Steiner algorithm reduces wirelength without attention to source-sink signal delays, it is, on the surface, not a specifically "performance-driven" algorithm. However, consideration of the Elmore model for distributed RC delay yields a slightly broader view of the continuing utility of Steiner tree constructions.

2.2 Intuitions From the Elmore Model

If we consider the Elmore delay formula [7] (the first-order moment of the impulse response when the routing tree is treated as a distributed RC tree), we may develop useful intuitions for routing tree construction with respect to critical sinks. Let e_v denote the edge from node v to its parent when we root the tree at n_0. The resistance and capacitance of e_v are respectively given by r_{e_v} and c_{e_v}. Let T_v denote the subtree of T rooted at v, and let c_v denote the node capacitance of v. The *tree capacitance* C_v of T_v is the sum of node and edge capacitances in T_v [7] [19]. The Elmore delay is then given by $t_{Elmore}(n_0, n_i) = \sum_{e_v \in path(n_0, n_i)} r_{e_v}(c_{e_v}/2 + C_v)$.

In Figure 3, we show a given net N with identified critical sink n_c, along with three routing trees: (a) the 1-Steiner tree, (b) a minimum cost, shortest-paths tree (SPT), and (c) an optimal-delay tree. We may make several observations.

1. The 1-Steiner solution has large delay to n_c due to the long source-sink path. Indeed, the Elmore formula suggests that optimal delays will be obtained when source-sink paths are as short as possible (i.e., monotone staircase routes).

2. On the other hand, the monotone paths to every sink that are present in the SPT tree (b) result in extra tree capacitance, again implying large delay at n_c.

3. The third construction (c) shows that identification of the critical node clearly affects the optimal routing topology.

4. The Elmore delay formula clearly shows the effect of *technology* in the choice of optimal critical-sink routing topologies. For example, note that the formula as stated seems to imply that the optimal-delay routing will be a *star*, i.e., a separate wire from every sink directly to the source. Consider the fact of non-zero driver resistance as being embodied by a wire from a new "virtual source" n_0' to the original source n_0, with the original routing tree topology still incident to n_0. In this scenario, delays from n_0' to the sinks will become more dependent on tree cost (and less dependent on directness of the source-sink connections) as we increase the length of this n_0'-n_0 wire, i.e., as we increase the driver resistance: this justifies the minimum Steiner routing used in present global routers. On the other hand, as we decrease the driver resistance (as in multi-chip module technologies; also consider the widely variable capacitance of IC and MCM interconnect technologies), monotonicity of paths becomes more important than overall tree cost. Indeed, the optimal topology of Figure 3(c) is evocative of *both* these concerns.[2] (With this in mind, our experiments reported below have been performed with respect to several distinct technology files.)

5. Along a similar vein, we may infer from the Elmore delay formula that the number of Steiner points in the n_0 - n_c path should in general be minimized, and the Steiner points should always 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 much less signal delay at n_c.)

Finally, note that the optimal tree in Figure 3(c) minimizes total tree cost (as in the 1-Steiner tree), *subject to the path from n_0 to n_c being monotone.* This simultaneous optimization of tree radius and tree cost recalls the motivations for the BRBC approaches, but here we optimize with respect to the critical sink n_c. With this in mind, our basic heuristic for producing critical path-dependent interconnection topologies is our *Critical-Sink Routing Tree* (CSRT) algorithm as shown in Figure 4.

[2]Recall the ability of the BRBC algorithm [4] to trade off between the conflicting goals of source-sink directness and overall tree cost; see also the Steiner formulation of [17].

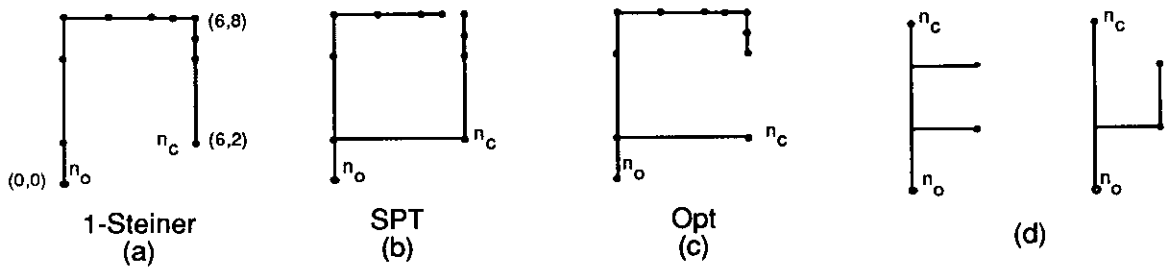


Figure 3: Parts (a)-(c): optimal Steiner tree (cost 2.0cm, delay(n_c) 5.90ns); minimum cost shortest-paths tree (cost 2.5cm, delay(n_c) 4.11ns); and optimal-delay tree (cost 2.2cm, delay(n_c) 3.07ns) for the same sink set. Coordinates are in mm's, and IC simulation parameters were used. Part (d): two distinct minimum cost shortest-paths trees for a set of three sinks.

Procedure Critical-Sink Routing Tree (CSRT) (template)
Input: Sink Locations N; identified critical sink $n_c \in N$
Output: Routing tree T over N
1. Construct heuristic (minimum-cost) tree T_0 over $N - n_c$. 2. Make a *direct* connection from n_c to T_0 using the least added wire such that the n_0-n_c path is monotone.

Figure 4: Critical-Sink Routing Tree template.

In all of our simulations we use the 1-Steiner tree heuristic to construct tree T_0 in Step (1). In Section 4 we report results for our basic CSRT heuristic (which we call H1) and for four simple variants:

1. **H0:** modify Step (2) in the CSRT template so that the critical node is connected by a single edge to the source.

2. **H1:** CSRT algorithm as stated in the Figure 4 template.

3. **H2:** modify Step (1) in the CSRT template to construct a heuristic tree over the entire net N. Then, delete the edge directly above node n_c in the tree rooted at n_0, and reconnect n_c using Step (2).

4. **H3:** perform Steps (1) and (2) simultaneously by executing the 1-Steiner algorithm subject to a "maintain monotone feasibility" constraint.[3]

[3]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. The direct connection from n_c requires that there exists a monotone path through the "bounding boxes" of the edges in the path to n_0. Intuitively, this favors Steiner nodes along a monotone path between n_0 and n_c, since such nodes will most rapidly reduce the marginal cost of adding a direct connection.

5. **HBest:** repeat Step (2) by making the closest connection of n_c to each edge in the heuristic tree T_0 plus to n_0. Run timing analysis separately on each of these possible routing trees, and then return the routing tree which has the smallest delay to n_c.[4]

The complexity of these heuristics is dominated by the complexity of the Step (1) tree construction.[5]

2.3 Global Slack Removal

To complement the CSRT approach, we propose a linear-time postprocessing algorithm, *Global Slack Removal* (GSR), which shifts edges in the 1-Steiner output to maximize the monotonicity of source-sink paths without increasing total wirelength. For the sake of brevity, we call any tree that can be produced by the 1-Steiner algorithm a *1-Steiner tree.* If we orient the 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 such that the v_1-v_4 pathlength in T is greater than the v_1-v_4 Manhattan distance (Figure 5(a)). The GSR algorithm takes as input a rooted (1-)Steiner tree T and removes U's as shown in Figure 5(b); the input tree is processed in top-down order.[6] Further details of the method are contained in the pseudo-code of Figure 6. Note that the algorithm 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 usually easiest to implement. Notice also that the current node in the traversal is checked to see if it is the *third* node in a U. This detail is important for ensuring that the output tree has no remaining U's. 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 because

[4] In some cases, heuristics H0 and HBest will produce solutions with crossing edges in T. If a non-self intersecting routing solution is required, these heuristics can be modified so that H0 makes the closest connection to n_c that does not cross edges; similarly, HBest may consider only connections to a subset of edges that can be reached without intersecting other edges.

[5] It should be clear to the reader that using a minimum spanning tree or heuristic minimum Steiner tree in Step (1) is evocative of the BRBC method [4] in the sense of using $\epsilon = 0$ for n_c and $\epsilon = \infty$ for all other sinks. Indeed, the authors of [4] describe an extension which permits differing ϵ_i values for each sink n_i. However, the $(1 + \epsilon_i) \cdot R$ radius bound is with respect to the *net radius*, R, which is a function of *all* sink locations; the BRBC extension thus cannot enforce a monotone path to the critical sink. The Ph.D. thesis of G. Robins [18] describes how to enforce distinct ϵ_i values with respect to a different R_i value at each sink n_i. Our construction simplifies due to the restriction $\epsilon_i \in \{0, \infty\}$.

[6] The GSR algorithm can actually be run on any Steiner tree input to reduce source-sink pathlengths without increasing wirelength. GSR is guaranteed to return a tree with no remaining U's only in the case that no single Steiner node can be added to the input tree so as to reduce wirelength.

(a) more U's can be found if all low-degree Steiner nodes are removed at the beginning of the algorithm, and (b) each removal of a U can introduce additional low-degree Steiner nodes, so low degree Steiner nodes must again be eliminated at the end of the algorithm.

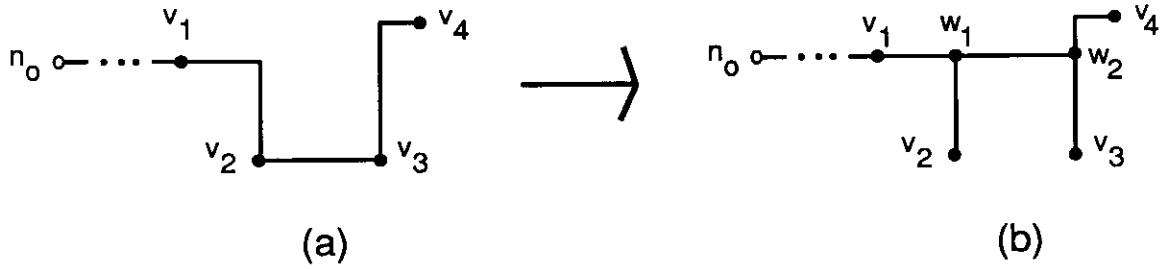


Figure 5: Removing a single "U" in the GSR Algorithm.

GSR Algorithm

```
Procedure GSR
    remove all Steiner nodes of degree ≤ 2;
    Q ← Enqueue(n0);
    while Q not empty
        v3 ← Dequeue(Q);
        v2 ← pred(v3);
        v1 ← pred(v2);
        for each node v4 ∈ succ(v3) do
            Q ← Enqueue(v4);
            if path v1v2v3v4 is a U
                remove the U as in Figure 5;
                insert Steiner nodes w1 and w2 into the tree;
            end if;
        end for;
    end while;
    remove all Steiner nodes of degree ≤ 2;
end procedure
```

Figure 6: Pseudo-code description of the GSR Algorithm.

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

Theorem 1: GSR will return a tree T' such that (i) T' has the same or less total wirelength 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 pathlength in T; and (iii) the Elmore delay at each n_i in T' is less than or equal to the Elmore delay in T.

Proof: (i) In Figure 5, 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 5(b), then T' will have less total wirelength than T after all low-degree Steiner nodes are removed.
(ii) In Figure 5, 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 5 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). □

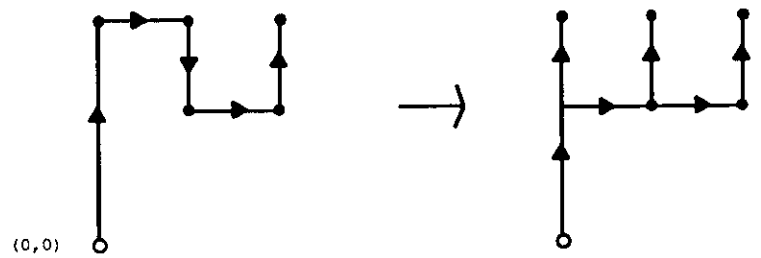
We note that the order in which U's are removed from the tree is important. If the U's are processed in a bottom-up order instead of a top-down order, then new U's can be introduced and the resulting tree may not have all of its U's removed. An example of this is seen in Figure 7. Furthermore, two different 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 8.

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 its wirelength.

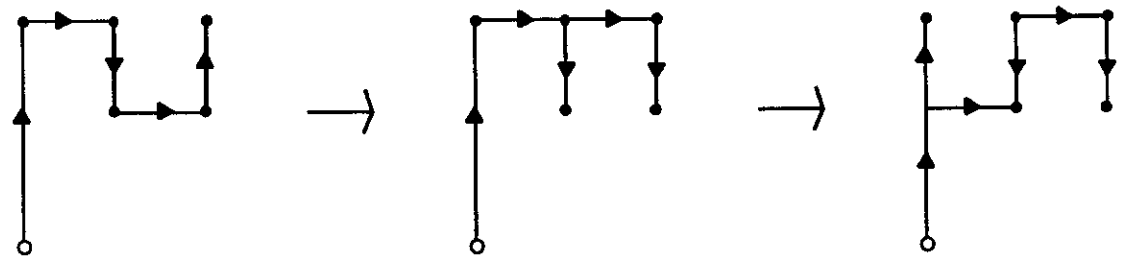
Lemma 2: In any 1-Steiner tree, the middle edge of a U must be either horizontal or vertical (i.e., not "L"-shaped).

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



(a)

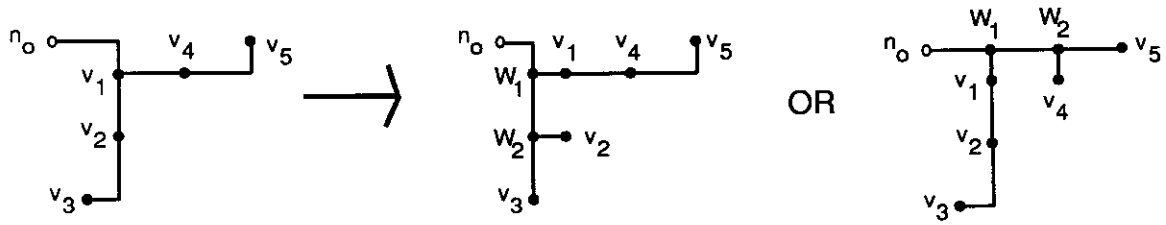
Removal of U's in
top-down order



(b)

Removal of U's in
bottom-up order

Figure 7: An example for which (a) processing U's in a top-down order returns a tree with no remaining U's and (b) processing U's in a bottom-up order returns a tree with one remaining U.



(a) (b) (c)

Figure 8: Example in which the GSR algorithm with input (a) can produce either tree (b) or (c) depending on the order that the U's are processed.

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 node in the input tree, and testing for and removing each U requires constant time. Hence, GSR runs in linear time.

We ask whether the removal of the U $v_1v_2v_3v_4$ as in Figure 5 can create any new U's which are not be inspected by the algorithm later on. Consider which nodes could be the third node in such a 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. □

3 Experimental Results and Discussion

We ran our five heuristics along with the 1-Steiner algorithm (1Stein) [11] on random 4-, 8- and 16-sink inputs. We also applied GSR (denoted as +U) to 1Stein and each of our heuristics except H3 (note that GSR has no effect on the monotone n_0-n_c path returned by the modified 1-Steiner heuristic used in H3). 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.[7]

Table 1 gives delay and wirelength results and comparisons. The delays at all sink nodes were measured using the two-pole circuit simulator proposed by Zhou et al. [23] and discussed in [2]. This simulator is a computationally efficient method which has produced very accurate results when tested against the commonly used circuit simulator SPICE. Each entry in Table 1 represents an average taken over every sink node in 50 random point sets. We emphasize that the 1Stein algorithm (or the BRBC method, etc.), 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 critical sinks n_i are in some sense "generic". In contrast, each our five heuristics can return a different tree for each choice of critical sink in the same net. Thus, for each of our CSRT variants, we report the delay at n_i in the *specific* tree corresponding to the identification of n_i as the critical sink.

[7] 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.

		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
	H2+U	2.24	3.13	4.14	9.24	12.11	15.72
	H3	2.33	3.23	4.39	9.50	12.97	17.91
	HBest+U	2.12	2.77	3.47	7.02	7.31	7.35
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
	H2+U	1.53	2.32	3.37	15.78	22.81	33.34
	H3	1.61	2.33	3.28	16.41	23.09	32.66
	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	35.5	12.5	2.3	3.5	0.5	0.1
Nodewise (HBest)/(1St) 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 results for modern IC and MCM technologies. Notes: (i) all source and sink locations are chosen randomly in layout region with grid resolution $25\mu m$; (ii) HBest+U and 1Stein wins compare only these two heuristics and do not necessarily add up to 100% because of ties; (iii) Min (Max) Nodewise Delay Ratio is geometric average of, for each sink set N, the min (max) HBest+U, 1Stein delay ratio over all sinks in N; and (iv) Meta result for each n_i in each sink set is the tree (1Stein or HBest+U tree) which gives smaller delay to n_i.

A number of interesting conclusions may be drawn from these results. The variants H0 and HBest are very successful in reducing delay to the critical node, particularly in nets with a large number of sinks and in the MCM technology where driver and wire resistances are low. In other words, a strategy of connecting the critical node along a path with a low branching factor is very successful for these cases. Of course, this strategy often produces an undesirable side effect of larger net wirelength.[8]

It is clear that the success of critical-path routing compared to generic 1-Steiner routing depends on the choice of critical node n_i. For example, in each IC sink set with $|N| = 17$, one expects to find at least one n_i for which the H1+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.

[8]This "star" strategy may introduce other technical problems such as crossing wires and nodes of degree > 4.

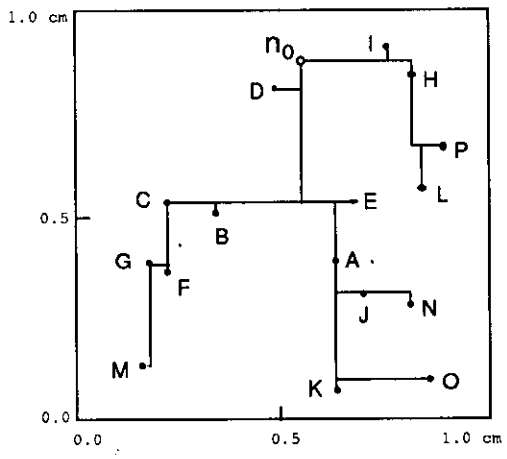
		IC			MCM		
		$\|N\| = 5$	$\|N\| = 9$	$\|N\| = 17$	$\|N\| = 5$	$\|N\| = 9$	$\|N\| = 17$
Rank of Sink	1	0.90	0.84	0.83	0.51	0.44	0.45
	2	0.91	0.78	0.76	0.62	0.37	0.34
	3	0.96	0.80	0.68	0.75	0.44	0.27
	4	0.99	0.80	0.68	0.95	0.44	0.28
	5		0.82	0.66		0.47	0.30
	6		0.87	0.68		0.54	0.26
	7		0.91	0.67		0.63	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.)

The distinct "domains of expertise" for the 1-Steiner and CSRT approaches are correlated with the distance of the critical sink from the source. (We have also observed this phenomenon in all other comparisons of CSRT against other global routing variants, e.g., the BRBC method.) Table 2 illustrates this correlation between the HBest+U and 1Stein+U methods. In the table the ratio of delays between HBest+U and 1Stein+U is compared on a sink-by-sink bases, comparing sinks in sorted order of distance from the source. Generally, critical-path routing is most successful for sinks closer to the source, except for the one very closest to the source.

The comparison between H1+U and a shortest-path tree of low wirelength shown in Figure 3 is also revealing: in general, for critical sinks that are placed close to the source, the H1+U tree can offer a significant delay improvement, but if the critical sink is far away, the SPT is usually a better choice.[9]

[9]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 9: Random 16-sink IC example (1cm by 1cm layout region), showing nodewise ratios of H1+U delays to a heuristic minimum cost shortest-paths tree, which is shown in the figure.

4 Future Work

We are considering extensions of critical path-dependent routing tree design to the general case of multiple critical sinks with prescribed delay bounds.

References

[1] 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.

[2] K. D. Boese, J. Cong, A. B. Kahng, K. S. Leung and D. Zhou, "On High-Speed VLSI Interconnects: Analysis and Design" for Critical Path Optimization", *Proc. Asia-Pacific Conf. on Circuits and Systems*, Dec. 1992, to appear.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

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

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

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

[16] S. Prastjutrakul 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.

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

[18] G. Robins, "On Optimal Interconnections", Ph.D. Thesis, CS Dept., University of California, Los Angeles, June 1992.

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

[20] 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.

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

[22] 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.

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