Prim-Dijkstra Tradeoffs for Improved Performance-Driven Routing Tree Design*

C. J. Alpert, T. C. Hu†, J. H. Huang, A. B. Kahng and D. Karger‡

CS Dept., University of California at Los Angeles, Los Angeles, CA 90024-1596
† CSE Dept., University of California at San Diego, La Jolla, CA 92093-0114
‡ CS Dept., Stanford University, Stanford, CA 94305

Abstract

Analysis of Elmore delay in distributed RC tree structures shows the influence of both tree cost and tree radius on signal delay in VLSI interconnects. We give new and efficient interconnection tree constructions that smoothly combine the minimum cost and the minimum radius objectives, by combining respectively *optimal* algorithms due to Prim and Dijkstra. Previous "shallow-light" techniques [2, 3, 8, 13] are both less direct and less effective: in practice, our methods achieve uniformly superior cost-radius tradeoffs. Detailed timing simulations for a range of IC and MCM interconnect technologies show that our wirelength savings yield reduced signal delays when compared to shallow-light or standard minimum spanning tree and Steiner tree routing.

1 Introduction and Motivation

With the scaling of device technology and die size, interconnection delay now contributes up to 50% to 70% of the clock cycle in dense, high performance circuits [4]. Performance-driven layout design has therefore been studied actively in recent years. Initial research centered on timing-driven placement, where the objective is to place modules on critical paths close together. Given a module placement, timing-driven *routing* algorithms (e.g., [15, 17]) attempt to minimize average or maximum signal delay from the source terminal to the sink terminals of a signal net.

*Partial support for this work was provided by a Department of Defense Graduate Fellowship; 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 an NSF Graduate Fellowship, NSF CCR-9010517, and grants from Mitsubishi and OTL.

A *signal net* $V = \{v_0, v_1, \ldots, v_n\}$ is a set of $n+1$ terminals, with v_0 the *source* and the remaining terminals *sinks*. In the underlying routing graph $G = (V, E)$, each edge $e_{ij} \in E$ has cost d_{ij} equal to the v_i-v_j routing cost. The cost of the shortest v_0-v_i path in G is denoted by R_i, and $R = \max_{1 \le i \le n} R_i$ is the *radius* of G. A *routing tree* $T = (V, E')$ is a spanning subgraph of G with $|E'| = n$. Given a routing tree T, the cost of the unique v_0-v_i path in T is l_i, the *radius* of T is $r(T) = \max_{1 \le i \le n} l_i$, and the *cost* of T is $w(T) = \sum_{e_{ij} \in T} d_{ij}$. We are primarily concerned with the case where G is a complete graph with each e_{ij} having cost equal to the Manhattan distance d_{ij}.

For a given signal net, the proper objective for *efficiently* constructing a "high-performance routing tree" is not yet established. We can obtain valuable insight by considering the Elmore delay, i.e., the first moment of the impulse response for a distributed RC representation of the routing tree [11]. This approximation has been shown to have high *fidelity* with respect to SPICE-computed delays, e.g., [14] simulated critical-path delays over a suite of 209 ripple-carry adder implementations and found near-perfect rank correlation between SPICE-computed and Elmore delays, and [24] gave theoretical motivations for this phenomenon.

Elmore delay is defined as follows [20]. Let r_e and c_e denote respectively the resistance and capacitance of edge e, and let c_i denote the node capacitance of v_i. Given T, let T_e denote the subtree of T that is below edge e when T is rooted at v_0, and let $\mathbf{C}_e$ denote the sum of node and edge capacitances of T_e. Finally, let $\mathbf{C}_0$ indicate the total capacitance of T. The Elmore delay along edge e is $r_e \cdot (\frac{c_e}{2} + \mathbf{C}_e)$. If r_0 denotes the on-resistance of the output driver at the source node, then the Elmore delay $t(v_i)$ from source v_0 to sink v_i is given by

$$t(v_i) = r_0 \cdot \mathbf{C}_0 \quad + \sum_{e \in path(v_0, v_i)} r_e \cdot (\frac{c_e}{2} + \mathbf{C}_e). \tag{1}$$

If r_e and c_e are proportional to the length of e, then the $r_0 \cdot \mathbf{C}_0$ term in Eq. (1) implies that $t(v_i)$ has linear dependence on $w(T)$, while the summation term implies quadratic dependence on l_i. This is the essential "cost-radius conflict" of routing tree design: (i) when r_0 is relatively large, the $r_0 \cdot \mathbf{C}_0$ term dominates the summation term and suggests a minimum-cost routing solution, but (ii) when r_0 is relatively small, the quadratic dependence on source-sink pathlength dominates, and suggests a "star-like", shortest-path tree topology. Typical values of r_0 and $\frac{r_0}{\bar{r}}$

(where $\bar{r}$ is unit wire resistance; $\frac{r_0}{\bar{r}}$ is the "resistance ratio" discussed in [5, 9]) have generally decreased with the trend to submicron CMOS and MCM technologies (see Table 1), suggesting that minimum-cost routing is becoming less correlated with performance-driven routing.

Name	IC1	IC2	IC3	MCM
Technology	2.0 μm CMOS	1.2 μm CMOS	0.5 μm CMOS	MCM
r_0	164.0 Ω	212.1 Ω	270.0 Ω	25.0 Ω
$\bar{r}$ = unit wire resistance	0.033 $\Omega/\mu m$	0.073 $\Omega/\mu m$	0.112 $\Omega/\mu m$	0.008 $\Omega/\mu m$
$\bar{c}$ = unit wire capacitance	0.019 $fF/\mu m$	0.022 $fF/\mu m$	0.039 $fF/\mu m$	0.06 $fF/\mu m$
sink loading capacitance	5.7 fF	7.06 fF	1.0 fF	1000 fF
$\frac{r_0}{\bar{r}}$ (x $10^6 \mu m$)	0.0050	0.0029	0.0024	0.0031
chip size	1x1 cm^2	1x1 cm^2	1x1 cm^2	10x10 cm^2

Table 1: Interconnect parameters for three CMOS IC technologies and an MCM technology. Parasitics for the IC1 and IC2 technologies are provided by MOSIS; IC3 parasitics are courtesy of MCNC; MCM interconnect parasitics are courtesy of Professor Wayne W.-M. Dai of UC Santa Cruz and correspond to data provided by AT&T Microelectronics Division. The r_0 values are scaled driver resistances. Sink loading capacitances (c_i) are derived for minimum-size transistors.

The remainder of this paper is organized as follows. Section 2 surveys the class of "shallow-light" constructions that have previously addressed cost-radius tradeoffs. Section 3 presents our two "Prim-Dijkstra" constructions, which directly trade off between algorithms that are respectively optimal for cost and radius. Section 4 compares both the cost-radius and delay performance of our heuristics to those of previous methods, and Section 5 concludes with possible directions for future research.

2 Related Work: Shallow-Light Trees

A minimum spanning tree (MST, or T_M) minimizes tree cost but may have radius an unbounded factor larger than optimal. Conversely, a shortest-path tree (SPT, or T_S) minimizes tree radius but may have cost an unbounded factor larger than optimal (see Figure 1). Several groups have proposed "shallow-light" tree constructions that guarantee to capture properties of both T_M and T_S *simultaneously* to within constant factors of optimal [2, 3, 8, 13]. Following [13], we adopt the following definition.

Definition: Given a signal net V and constant parameters $\alpha, \beta \geq 1$, an (α, β)-tree $T = (V, E')$

is a spanning tree over V that satisfies: (i) $l_i \leq \alpha \cdot R_i$, $1 \leq i \leq n$, and (ii) $w(T) \leq \beta \cdot w(T_M)$.

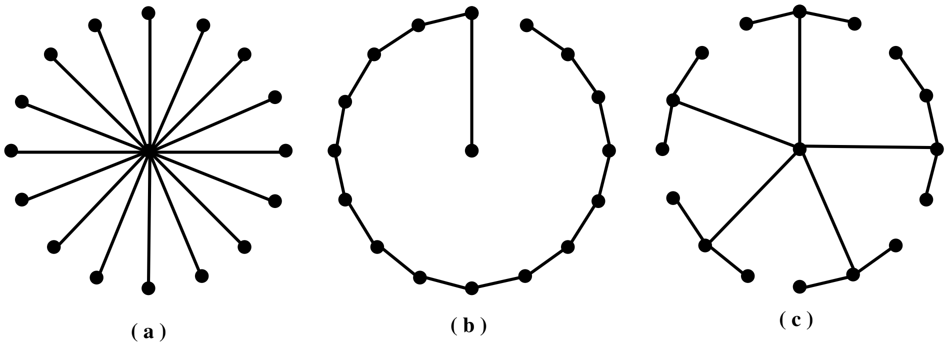


Figure 1: Three interconnection trees for the same signal net with v_0 at the center: (a) the shortest-path tree T_S; (b) the minimum spanning tree T_M; and (c) a "tradeoff" between the two constructions.

A spanning tree construction is *shallow-light* if for some constants α and β it always returns an (α, β)-tree. Since the cost of a spanning tree rises as its radius is constrained, we are typically interested in shallow-light constructions where α and β have an inverse functional relationship, i.e., α is a parameter of the construction and β decreases as α increases. The previous shallow-light constructions are all based on the general technique of Awerbuch et al. [2]: (i) construct T_M; (ii) visit the terminals of V in order of a depth-first traversal of T_M; (iii) whenever violations of the prescribed radius bound are observed, insert or delete edges as necessary; and (iv) return the shortest-path tree over the resulting graph. Cong et al. [8] proposed the BRBC algorithm for performance-driven global routing in VLSI and compared their results against Steiner routing using the linear delay model. In [3], Awerbuch et al. proposed an algorithm that is identical to BRBC and showed that it constructs a $(1 + 2\epsilon, 1 + \frac{2}{\epsilon})$-tree, for some parameter $\epsilon > 0$. Finally, Khuller et al. [13] obtained a $(1 + \epsilon, 1 + \frac{2}{\epsilon})$ shallow-light construction by "relaxing" edges, in contrast to earlier works which add complete source-sink shortest paths when violations of the radius bound occur.

A recent paper of Salowe et al. [21] presents a shortest-path "bottleneck tree" construction that is not shallow-light. [21] was motivated by the preliminary version of our present work [1] and is a powerful generalization of what we call the ALG2 construction in Section 3 (thus, the

ALG2 construction was independently developed by two groups; see Section 3.2 below). Finally, the topic of sparse graph spanners has been treated in the computational geometry literature (e.g., [7]). However, a graph spanner has bounded pathlengths between *all pairs* of nodes in a given graph, which is too strong a constraint for our (single-source routing) application.

3 The Prim-Dijkstra Tradeoff

The min-cost and min-radius objectives can be separately addressed by Prim's MST algorithm [18] and Dijkstra's SPT algorithm [10]. Tarjan [23] discusses the similarity between the Prim and Dijkstra algorithms: each is a variant of the "labeling method" that builds a spanning tree from a fixed source by adding the edge that minimizes an algorithm-specific "key". Our contribution stems from observing that the min-cost and min-radius objectives can be addressed *simultaneously* via direct combinations of the Prim and Dijkstra constructions. The combination of competing objectives, via a tradeoff of algorithms that are respectively optimal for these objectives, is unusual. Furthermore, our approach is more natural and symmetric than the shallow-light technique. While the constructions that we present do not yield (α, β)-trees, they are in practice more useful (see Section 4 below).

3.1 The ALG1 Tradeoff

Prim's algorithm begins initially with the tree consisting only of v_0. The algorithm iteratively adds edge e_{ij} and sink v_i to T, where v_i and v_j are chosen to minimize

$$d_{ij} \quad s.t. \quad v_j \in T,\ v_i \in V - T \tag{2}$$

Dijkstra's algorithm also begins with the tree consisting only of v_0. The algorithm iteratively adds edge e_{ij} and sink v_i to T, where v_i and v_j are chosen to minimize

$$l_j + d_{ij} \quad s.t. \quad v_j \in T,\ v_i \in V - T \tag{3}$$

Noticing the similarity between (2) and (3) leads to our ALG1 tradeoff, which iteratively adds edge e_{ij} and sink v_i to T, where v_i and v_j are chosen to minimize

$$(c \cdot l_j) + d_{ij} \quad s.t. \quad v_j \in T, \; v_i \in V - T \tag{4}$$

for some choice of $0 \leq c \leq 1$. When $c = 0$, ALG1 is identical to Prim's algorithm and constructs trees with minimum cost. As c increases, ALG1 constructs a tree with higher cost but lower radius, and when $c = 1$ ALG1 is identical to Dijkstra's algorithm. Sample executions of ALG1 for $c = \frac{1}{3}$ and $c = \frac{2}{3}$ are shown in Figure 2(a)-(b).

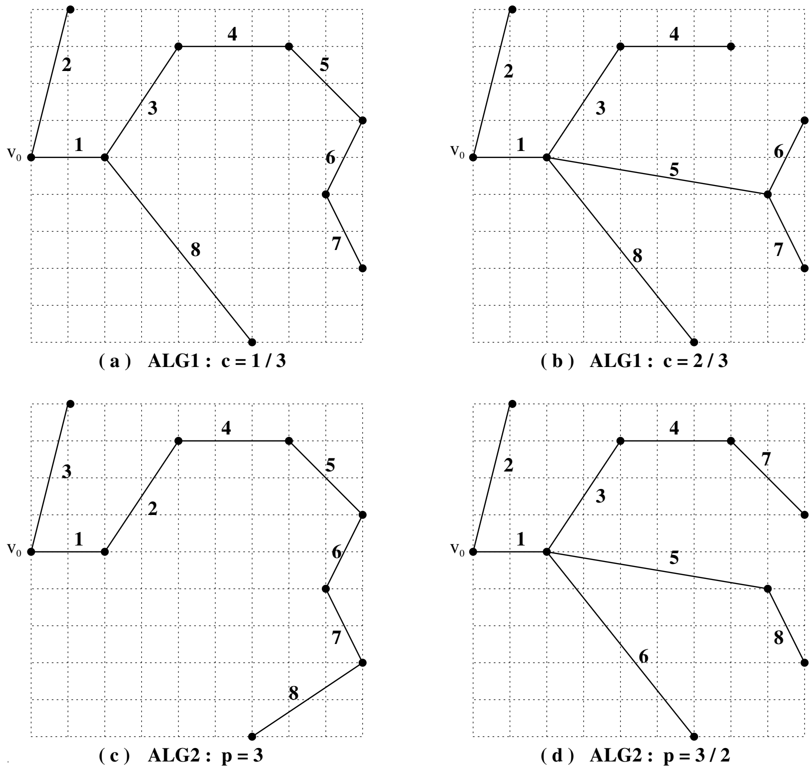


Figure 2: Sample executions for ALG1 and ALG2 for an 8-sink instance in the Euclidean plane. The edge labels give the order in which the algorithms add the edges into the tree. ALG1 is illustrated in (a) with $c = \frac{1}{3}$ (radius 15.91, cost 26.43) and (b) with $c = \frac{2}{3}$ (radius 10.32, cost 29.69). ALG2 is illustrated with "corresponding" parameterizations according to Table 2: (c) with $p = 3$ (radius 17.00, cost 23.63) and (d) with $p = \frac{3}{2}$ (radius 10.00, cost 30.28).

Observation 1: ALG1 constructs a tree T with $c \cdot l_i \leq R_i$ for all sinks v_i.

Proof: By strong induction; see Figure 3. Assume that for every ancestor v_j of v_i in T_S, we have $c \cdot l_j \leq R_j$. Consider a snapshot of T immediately before ALG1 adds sink v_i to T via edge v_{mi},

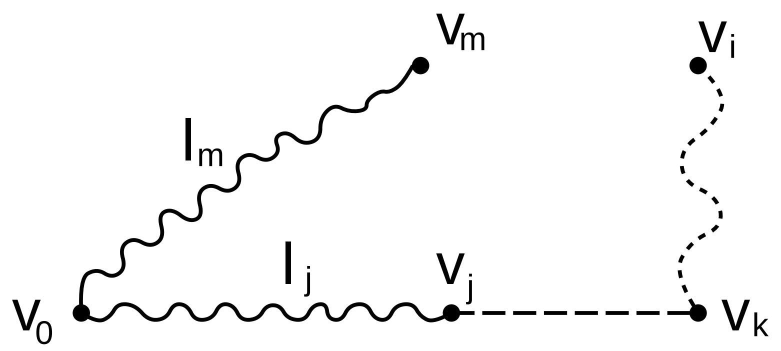


Figure 3: Illustration for proof of Observation 1. Snapshot of the ALG1 tree T (solid lines) just before edge v_{mi} is added. With respect to the v_0-v_i shortest path, v_j is the sink on the path that is furthest from v_0 and in T, while v_k lies just after v_j and is not in T. The observation follows from the fact that v_{mi} and not v_{jk} is the next edge added to T.

i.e., where v_m is the parent of v_i in T. Let v_j be the sink in T which is the closest ancestor to v_i in T_S (possibly $j = 0$), and let v_k be the sink lying immediately past v_j along the shortest v_0-v_i path (v_k is not yet in T, and possibly $k = i$). Since ALG1 adds v_i before v_k, $c \cdot l_m + d_{mi} \leq c \cdot l_j + d_{jk}$. By the inductive hypothesis, $c \cdot l_j \leq R_j$, and by the principle of optimality of shortest paths, $R_j + d_{jk} = R_k \leq R_i$. Since $c \cdot l_i \leq c \cdot l_m + d_{mi}$, combining these inequalities yields $c \cdot l_i \leq R_i$. $\square$

Observation 2: ALG1 is not shallow-light for general graph instances [1]. $\square$

Observation 3: For instances embedded in Euclidean space of any dimension d, ALG1 constructs a tree with cost within $\log n$ times a constant factor (dependent on d and c) of $w(T_M)$ [16]. $\square$

This last observation of Lenhof, Salowe and Wrege provides some encouragement regarding our conjecture [1] that ALG1 is actually shallow-light for geometric instances.

3.2 The ALG2 Tradeoff

Note that Dijkstra's algorithm can be viewed as using a key (in the terminology of [23]) which is the L_1 sum of edge costs in the source-sink path (Although L_p usually denotes a vector norm, here we simply say that the L_p sum of quantities $x_1, x_2, \ldots, x_n$ has value $({x_1}^p + {x_2}^p + \ldots + {x_n}^p)^{1/p}$; we write this as $||x_1, x_2, \ldots, x_n||_p$.). In the following, we use l_i^p to denote the L_p sum of edge costs in the v_0-v_i path in T. We will also use $|l_i|$ to denote the largest edge cost in the v_0-v_i path. The observation regarding Dijkstra's algorithm suggests our ALG2 tradeoff: iteratively add edge e_{ij} and sink v_i to T, where v_i and v_j are chosen to minimize

$$||l_j{}^p\ ,\ d_{ij}||_p \quad s.t. \quad v_j \in T,\ v_i \in V - T \tag{5}$$

for some choice of $1 \leq p < \infty$. Sample executions of ALG2 for $p = 3$ and $p = \frac{3}{2}$ are shown in Figure 2(c)-(d).

The ALG2 tradeoff and some of its properties were discovered independently by Salowe, Richards and Wrege [21], via an approach that is considerably different from ours. Salowe et al. apply the general single-source shortest path labeling method developed by Tarjan [23] to the "bottleneck" shortest-path problem, i.e., they use the label $\max\{|l_j|, d_{ij}\}$ and then generalize this to the objective of Equation (5). Unique to our work is that ALG2 embodies a Prim-Dijkstra tradeoff, i.e., that ALG2 can return either T_M or T_S depending on the value of p.

Observation 4: When $p = 1$, ALG2 yields a shortest path tree. $\square$

When $p = \infty$, the ALG2 objective reduces to $\max\{|l_j|, d_{ij}\}$, which yields a "bottleneck" shortest-path tree, i.e., if the cost of a path in the tree is the cost of the largest edge in that path, then ALG2 constructs a shortest-path tree in this sense when $p = \infty$. The optimal "bottleneck" tree is not unique: once a bottleneck edge with large cost is present in some source-sink shortest path, a bottleneck shortest-path tree is maintained by appending *any* edge with less cost than the bottleneck edge. In order for ALG2 at $p = \infty$ to capture the limiting behavior from large finite values of p, we break ties by choosing the sink v_i according to (5) which also minimizes d_{ij}. Given this tie-breaking rule, we have

Observation 5: When $p = \infty$, ALG2 is identical to Prim's algorithm.

Proof: By induction on the current size of T. Both Prim's algorithm and ALG2 will add the same first edge to T. Assume that Prim's algorithm and ALG2 both add the same first k edges and assume toward a contradiction that they differ at the $(k + 1)^{st}$ edge, i.e., Prim's algorithm adds edge e_1 and ALG2 adds edge e_2 as in Figure 4. Because e_1 and e_2 are distinct, the Prim objective implies $e_1 < e_2$. Since ALG2 added e_2, $\max\{|l_2|, e_2\} \leq \max\{|l_1|, e_1\}$. Moreover, if $\max\{|l_2|, e_2\} = \max\{|l_1|, e_1\}$, the tie-breaking rule would force ALG2 to choose e_1, hence $\max\{|l_2|, e_2\} < \max\{|l_1|, e_1\}$. Having $\max\{|l_1|, e_1\} = e_1$ contradicts $e_1 < e_2$, so $\max\{|l_1|, e_1\} = |l_1|$ and hence $\max\{|l_2|, e_2\} < |l_1|$. Let e be the edge in l_1 with cost $|l_1|$. Consider the tree immediately before edge e was added. Since every edge in the $l_2 e_2$ path has less cost than e, Prim's algorithm could not have added e before completely adding the $l_2 e_2$ path, contradicting the inductive hypothesis. $\square$

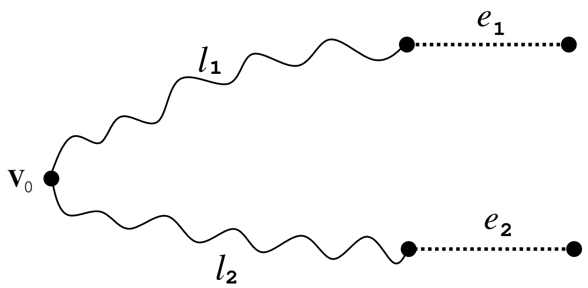


Figure 4: Illustration for proof of Observation 5. Prim's algorithm adds edge e_1; ALG2 adds edge e_2. Note that paths l_1 and l_2 may overlap.

Salowe et al. [21] have shown that ALG2 constructs a tree T with $l_i \leq R_i \cdot n^{1-1/p}$ for all sinks v_i, and that this bound is tight. It is easy to see that for any finite value of p, ALG2 may yield a tree with cost an unbounded factor greater than the MST cost, even in geometry.

4 Experimental Results

Both ALG1 and ALG2 have time complexity $O(n^2)$, since each is extendible from Dijkstra's algorithm. We tested our Prim-Dijkstra tradeoffs against the MST construction that is traditional in VLSI global routing, as well as against previous shallow-light algorithms (the KRY method of [13], and the BRBC method of [3, 8]). For a given problem instance, each cost-radius tradeoff generates a *family* of spanning trees corresponding to the range of parameter values; we study such families of output trees to determine the parameter values best suited to particular technology or area-performance requirements. In what follows, we compare the cost-radius tradeoffs over the

	ALG1	ALG2	BRBC	KRY
User parameter	c	p	ϵ	α
Yields T_M when	$c = 0$	$p = \infty$	$\epsilon = \infty$	$\alpha = \infty$
Yields T_S when	$c = 1$	$p = 1$	$\epsilon = 0$	$\alpha = 1$
Relation to c	c	$p = \frac{1}{c}$	$\epsilon = \frac{1-c}{c}$	$\alpha = \frac{1}{c}$

Table 2: Equivalences of algorithm parameters.

families of trees output by each algorithm, as well as delay simulation results over a range of IC and MCM technologies. We also compare cost-radius tradeoff and signal delay performance of the *Steiner* trees which are induced from the various spanning tree constructions.

4.1 Comparing Cost and Radius

For each signal net, we generated a "family" of 51 output trees for ALG1 with c ranging from 0 to 1 at intervals of 0.02. To generate corresponding families of trees for the other algorithms, we used input parameters that matched the ALG1 parameter values according to relationships inferred from the algorithms' limiting behaviors (see Table 2). We found that use of these relationships led to a good sampling of the families of trees generated by ALG2 and KRY. However, since BRBC tends to generate trees virtually identical to T_M for $\epsilon \geq 1.5$, we study the family of 51 trees generated by BRBC with the parameter ϵ ranging from 0 to 1.5 at intervals of 0.03.

We ran each algorithm over its family of parameter values, for signal nets of 16 sinks chosen randomly from a uniform distribution in a 1cm by 1cm Manhattan square; each point in Figure 5(a) represents an average over 250 such instances: All four algorithms "smoothly" trade off between cost and radius, with ALG1 being clearly superior, i.e., for any desired cost/radius tradoff, ALG1 performs uniformly better than the other algorithms. The superiority of ALG1 is especially clear for the tradeoff region that is of likely practical interest, i.e., when we wish to reduce tree radius without sacrificing more than 10% or 20% extra tree cost. ALG2 does not do as well as ALG1, but does provide superior cost/radius tradeoffs over the previous methods.

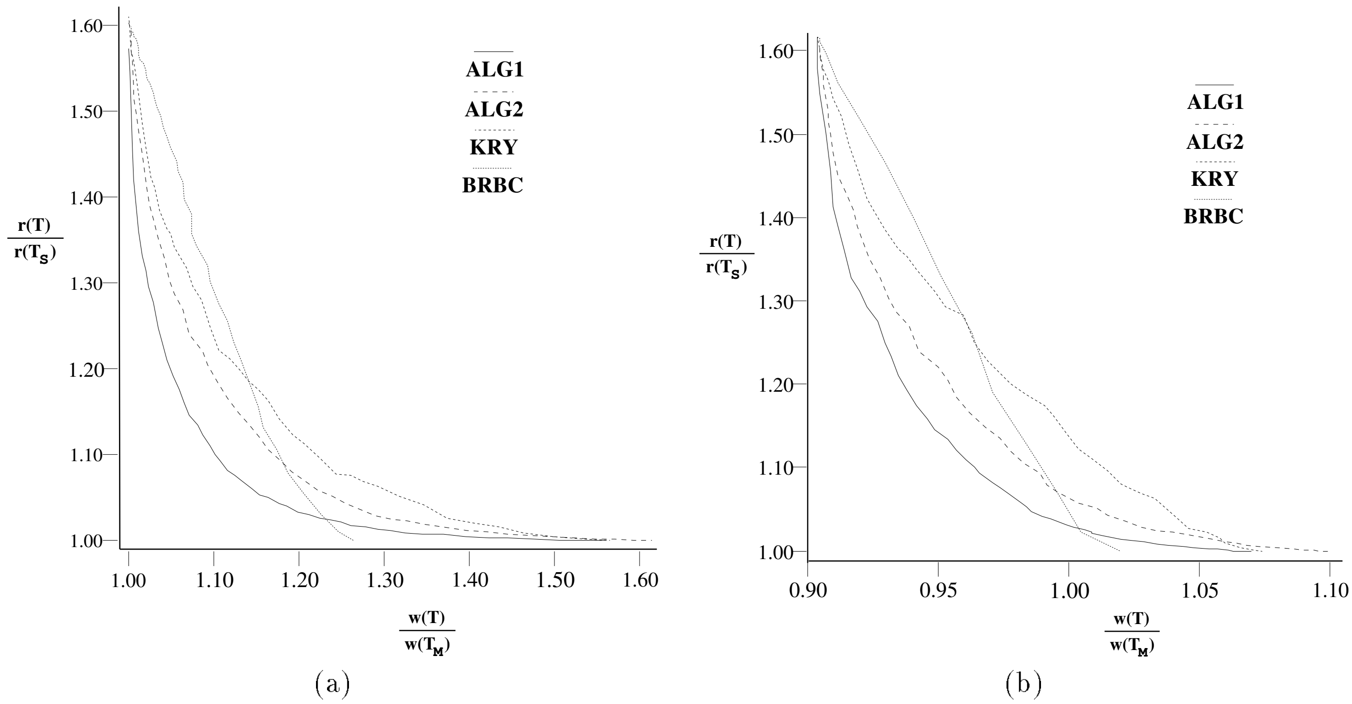


Figure 5: Graph of radius ratio ($\frac{r(T)}{r(T_S)}$) versus cost ratio ($\frac{w(T)}{w(T_M)}$) for ALG1, ALG2, KRY[13] and BRBC[3,8] for uniformly random instances of 16 sinks in the Manhattan unit square. Each point indicates the algorithm performance for a specific parameter value, averaged over 250 instances (a), and the same experiments with the edges overlapped to induce a Steiner topology (b).

4.2 Delay Simulations

We also compared the various tree constructions for uniformly random signal nets of 4, 8 and 16 sinks. Delays at all sink nodes were computed using the Two-Pole circuit simulator developed by Zhou et al. [25], for each of the four interconnect technologies listed in Table 1 of the Introduction. The Two-Pole simulator is a moment-matching distributed RCL delay code which produces very accurate results (within a few percent) when tested against SPICE3e [25]. We recorded both average delay (over all sinks) and maximum delay (i.e., the latest arrival time of the signal to any sink), with all results normalized to the corresponding values for the MST routing. (Delay was measured as the rise time to a stable value of 0.9 times the reference voltage of 5.0V, given a step input function.) For each instance, we ran each algorithm over each of the 51 user parameters described above, and recorded the lowest delay value of any tree in the family.

Table 3 gives maximum and average signal delays, averaged over 250 instances. Because practitioners might not wish to compute the best of 51 distinct trees, we also show the average value for each algorithm's best parameterization under both the max sink delay objective and the average sink delay objective. This indicates how the ideal cost-radius tradeoff parameter is correlated with technology and net size (for example, the best ALG1 c parameter for 16 sinks is is 0.23 for IC1 and 0.73 for MCM). If only one spanning tree construction is allowed, we believe the "best" parameter will generally yield a tree with low delay (see Footnote 2 below for further discussion of this issue). We find that ALG1 is the best algorithm of those tested, yielding delays that are better than or equal to those of its nearest competitor, KRY, in 27 of 30 comparisons. The Prim-Dijkstra tradeoffs achieve particularly substantial delay reductions over the minimum spanning tree routing for MCM, reinforcing our intuition that minimum-cost tree constructions are becoming less useful for newer interconnect technologies.

The KRY delays are surprisingly good in view of the algorithm's inferior cost/radius tradeoff. While ALG1 seems to yield a more "natural" tree (e.g., KRY trees are commonly self-intersecting, while ALG1 trees rarely are – see Figure 6), we believe that KRY benefits from its tendency to branch early from v_0, causing relatively little off-path tree weight for any given source-sink path. While our Prim-Dijkstra methods offer clear advantages over previous (performance-driven) routing constructions, the success of KRY underscores the continuing need for better routing tree analysis and design techniques.

4.3 Steiner Routing

Many global routing approaches require rectilinear Steiner tree constructions. A popular approach converts a spanning tree to a Steiner tree by overlapping the embeddings of tree edges. This method preserves the tree radius of the initial spanning topology within the eventual Steiner tree output. Ho et al. [12] have given a linear time construction that optimally converts a spanning tree to a Steiner tree by edge overlapping, but there are several reasons why their code is

Spanning Trees		Max sink delay vs. MST (best parameter)			
#sinks	ALG.	IC1	IC2	IC3	MCM1
4	ALG1	0.896 (0.13)	0.859 (0.28)	0.849 (0.30)	0.759 (0.39)
	ALG2	0.900 (23.62)	0.861 (19.25)	0.851 (18.29)	0.759 (14.84)
	KRY	0.897 (22.88)	0.859 (19.17)	0.850 (18.08)	0.759 (14.64)
	BRBC	0.906 (0.09)	0.872 (0.06)	0.863 (0.07)	0.786 (0.04)
8	ALG1	0.808 (0.19)	0.746 (0.45)	0.732 (0.46)	0.584 (0.62)
	ALG2	0.823 (11.24)	0.760 (7.66)	0.745 (6.80)	0.590 (3.46)
	KRY	0.815 (9.83)	0.752 (6.62)	0.736 (6.47)	0.584 (2.93)
	BRBC	0.850 (0.13)	0.796 (0.08)	0.784 (0.09)	0.657 (0.06)
16	ALG1	0.742 (0.23)	0.666 (0.49)	0.648 (0.52)	0.458 (0.73)
	ALG2	0.772 (4.78)	0.696 (3.39)	0 678 (3.09)	0.484 (1.40)
	KRY	0.752 (3.52)	0.671 (2.38)	0.648 (2.07)	0.456 (1.29)
	BRBC	0.831 (0.17)	0.772 (0.11)	0.758 (0.12)	0.615 (0.07)

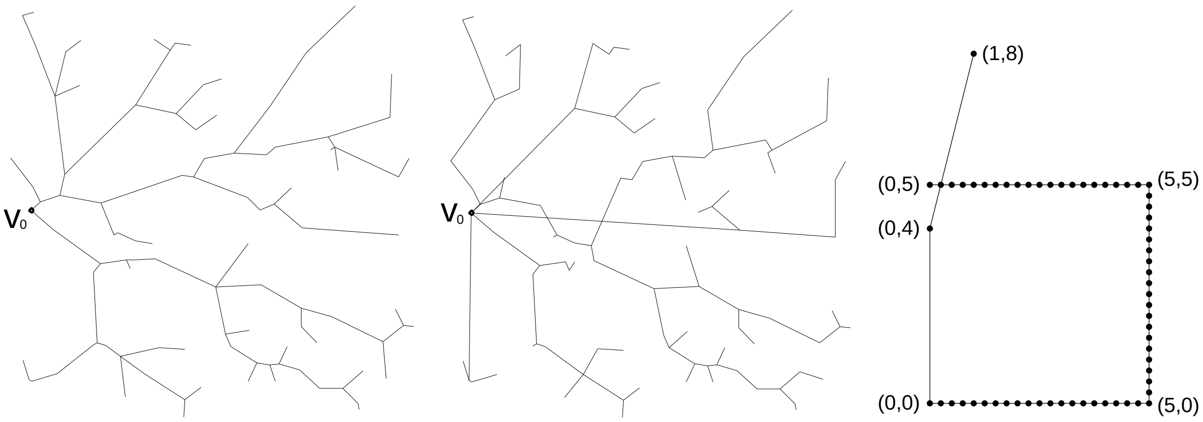
Spanning Trees		Avg sink delay vs. MST (best parameter)			
#sinks	ALG.	IC1	IC2	IC3	MCM1
4	ALG1	0.911 (0.10)	0.866 (0.32)	0.854 (0.34)	0.712 (0.55)
	ALG2	0.916 (21.29)	0.871 (17.28)	0.858 (16.27)	0.714 (8.59)
	KRY	0.912 (19.56)	0.866 (16.22)	0.854 (15.83)	0.712 (8.10)
	BRBC	0.928 (0.09)	0.891 (0.10)	0.880 (0.10)	0.768 (0.11)
8	ALG1	0.808 (0.15)	0.778 (0.47)	0.759 (0.49)	0.540 (0.75)
	ALG2	0.861 (11.45)	0.794 (6.31)	0.774 (5.85)	0.551 (2.00)
	KRY	0.850 (9.42)	0.781 (4.83)	0.760 (3.96)	0.540 (1.79)
	BRBC	0.899 (0.08)	0.848 (0.06)	0.834 (0.05)	0.678 (0.04)
16	ALG1	0.800 (0.20)	0.720 (0.48)	0.697 (0.50)	0.429 (0.82)
	ALG2	0.829 (5.22)	0.749 (2.90)	0.726 (2.58)	0.452 (1.24)
	KRY	0.808 (3.57)	0.723 (1.99)	0.696 (1.87)	0.424 (1.19)
	BRBC	0.893 (0.12)	0.839 (0.13)	0.824 (0.13)	0.648 (0.12)

Table 3: Maximum source-sink delay and average source-sink delay in the best tree for each algorithm. Values are given as a ratio to corresponding MST delay values, averaged over 250 random instances. Numbers in parentheses give average best parameter value for each algorithm.

not applicable to our spanning trees (e.g., our spanning trees can have high-degree nodes, and do not always satisfy the *separability* requirement of [12]). Thus, for simplicity we adopt a greedy edge-overlapping algorithm.[1]

Figure 5(b) and Table 4 show that the utility of our Prim-Dijkstra spanning tree constructions is preserved when the trees are converted to Steiner trees in this manner. The performance-driven spanning tree constructions with lowest delay still have lowest delay when Steiner points are

[1] Our greedy edge-overlapping method simply examines the bounding boxes of every pair of adjacent edges in the tree, and calculates the cost reduction achievable by optimally overlapping these edges (i.e., inducing a Steiner point). The Steiner point which yields the maximum cost savings is added, until no additional cost reduction is possible. While this heuristic is not guaranteed to be optimal, its output is nearly identical to that of the optimal edge-overlapping algorithm of Ho et al. (called S-RST in [12]). For random 10-node instances, our heuristic averages 8.8% cost reduction from an input minimum spanning tree, while S-RST is reported to average 9.0% reduction. For random 25-node instances, our heuristic averages 9.3% percent cost reduction over the minimum spanning tree, while S-RST is reported to average 9.5% reduction. Thus, we believe that the greedy heuristic is adequate for our study.



(a) (b) (c)

Figure 6: Execution of ALG1 with $c = 0.5$ (a) and KRY [13] with $\alpha = 1.5$ (b), on a 100-sink example using Euclidean distance. ALG1 trees may also be self-intersecting, e.g. (c), with $c = 0.22$ and source at $(0, 0)$, though this is rare in practice.

incorporated. The average best values of the input parameters shift to more star-like spanning topologies when the Steiner conversion is employed: since edge-overlapping decreases cost without affecting radius, we can afford spanning trees that use additional tree cost to further reduce the radius. (Anomalies may result since the overlapping process *diminishes* the star-like nature of the tree topology. Thus, a Steiner tree can have greater sink delay than its spanning tree precursor.)

Finally, we observe that our delay results are substantially better than the leading "fixed" methods, i.e., tree constructions which cannot be parameterized to track interconnect technology. For example, we average over 25% reduction in average sink delay when compared with the results reported in [9]; this is not surprising since even ALG1 or ALG1-Steiner (i.e., ALG1 followed by greedy edge-overlapping to create a Steiner tree) with *fixed* $c = 1.00$ already constitutes a reasonably good heuristic Steiner arborescence, or "A-tree", construction.[2] We may

[2] The reference [9] presents numerical results for one set of interconnect technology parameters, namely, MCM; the parameters and simulation methodology are identical to those we use here. By normalizing the reported A-tree results to those of BRBC with parameter $\epsilon = 1.0$ – i.e., the comparison made in [9] – we obtain the following ratios for average sink delay: (i) A-tree / BRBC-1.0 = 1.024 (4 sinks), 0.846 (8 sinks), 0.645 (16 sinks). When we use a fixed, "best" c value for ALG1 as given in Tables 3 and 4, we obtain the following results: (ii) ALG1 / BRBC-1.0 = 0.805 (4 sinks), 0.598 (8 sinks), 0.463 (16 sinks); and (iii) ALG1-Steiner / BRBC-1.0 = 0.896 (4 sinks), 0.728 (8 sinks), 0.577 (16 sinks). Note that (i) and (iii) both represent Steiner routing constructions. If we are allowed to consider the entire family of ALG1 trees, the results improve to (ii') ALG1 / BRBC-1.0 = 0.702 (4 sinks), 0.547 (8 sinks), 0.431 (16 sinks); and (iii') ALG1-Steiner / BRBC-1.0 = 0.738 (4 sinks), 0.598 (8 sinks), 0.502 (16 sinks).

Steiner Trees		Max sink delay vs. MST (best parameter)			
#sinks	ALG.	IC1	IC2	IC3	MCM1
4	ALG1	0.812 (0.17)	0.789 (0.21)	0.780 (0.24)	0.748 (0.30)
	ALG2	0.837 (32.56)	0.813 (30.46)	0.804 (29.14)	0.778 (25.33)
	KRY	0.812 (31.77)	0.789 (28.85)	0.780 (27.86)	0.747 (23.79)
	BRBC	0.816 (0.03)	0.794 (0.05)	0.785 (0.06)	0.758 (0.04)
8	ALG1	0.727 (0.34)	0.684 (0.46)	0.672 (0.49)	0.587 (0.57)
	ALG2	0.797 (16.53)	0.755 (9.81)	0.742 (9.25)	0.671 (6.77)
	KRY	0.728 (14.45)	0.684 (8.97)	0.672 (8.72)	0.586 (6.15)
	BRBC	0.744 (0.10)	0.707 (0.10)	0.694 (0.10)	0.628 (0.10)
16	ALG1	0.665 (0.44)	0.606 (0.52)	0.590 (0.54)	0.463 (0.68)
	ALG2	0.758 (11.52)	0.699 (5.63)	0.682 (3.72)	0.567 (1.93)
	KRY	0.671 (5.66)	0.611 (3.73)	0.596 (3.85)	0.466 (1.76)
	BRBC	0.713 (0.16)	0.664 (0.16)	0.651 (0.15)	0.556 (0.12)
Steiner Trees		Avg sink delay vs. MST (best parameter)			
#sinks	ALG.	IC1	IC2	IC3	MCM1
4	ALG1	0.807 (0.25)	0.775 (0.27)	0.763 (0.28)	0.694 (0.32)
	ALG2	0.821 (27.45)	0.787 (26.24)	0.773 (27.04)	0.706 (22.88)
	KRY	0.807 (26.12)	0.776 (24.37)	0.763 (24.56)	0.694 (21.85)
	BRBC	0.817 (0.06)	0.787 (0.05)	0.775 (0.06)	0.716 (0.06)
8	ALG1	0.749 (0.50)	0.696 (0.55)	0.680 (0.56)	0.551 (0.64)
	ALG2	0.808 (11.63)	0.754 (8.93)	0.737 (7.34)	0.617 (4.27)
	KRY	0.751 (9.86)	0.698 (6.15)	0.682 (5.55)	0.550 (3.38)
	BRBC	0.777 (0.18)	0.735 (0.15)	0.720 (0.17)	0.625 (0.13)
16	ALG1	0.711 (0.52)	0.644 (0.60)	0.624 (0.62)	0.443 (0.75)
	ALG2	0.791 (12.83)	0.719 (5.03)	0.697 (3.38)	0.526 (1.56)
	KRY	0.715 (3.99)	0.647 (2.48)	0.628 (2.28)	0.445 (1.29)
	BRBC	0.765 (0.24)	0.719 (0.25)	0.702 (0.25)	0.579 (0.29)

Table 4: Maximum source-sink delay and average source-sink delay in the best tree for each algorithm, after edge-overlapping is used to induce a Steiner routing. Values are given as a ratio to corresponding MST delay values, averaged over 250 random instances. Numbers in parentheses give average best parameter value for each algorithm.

similarly compare our constructions against the standard minimum Steiner tree heuristic of edge-overlapping a minimum spanning tree (this is identical to ALG1-Steiner with fixed $c = 0$). For this standard technique using MCM parameters, the ratios of average sink delays to corresponding MST delays are 0.802 (4 sinks), 0.808 (8 sinks), 0.818 (16 sinks). By contrast, a single ALG1 execution using the fixed, best c value from Table 3 will attain ratios of 0.788, 0.590, and 0.461 respectively. When edge-flipping is added in ALG1-Steiner and we use the best c values from Table 4, the average-delay ratios are 0.756, 0.586, and 0.468 respectively. (Delay ratios obtained by considering the family of ALG1 trees can be read off directly from Tables 3 and 4.)

5 Conclusions

Analysis of Elmore delay in RC trees suggests that low-delay routing trees should trade off cost and radius according to net size and interconnect technology. Previous approaches [2, 3, 8, 13] begin with a depth-first traversal of T_M and insert shortest paths as needed to maintain a prescribed radius bound. In contrast, our ALG1 (and ALG2) *directly* combine the recurrences for Prim's MST algorithm and Dijkstra's SPT algorithm. This more natural tradeoff significantly improves over the cost-radius performance of BRBC [3, 8] and KRY [13]. Simulation results show that ALG1 yields routing trees with less maximum and average delay than ALG2, KRY or BRBC in both IC and MCM interconnect technologies; improvements over the "provably good" BRBC approach are particularly substantial. Our delay reductions over fixed constructions are also substantial (such constructions include standard MST routing and heuristic minimum Steiner tree routing as well as the recent Steiner arborescence approach of [9]). It is therefore of interest to pursue integration of ALG1 within existing performance-driven global routers.

6 Acknowledgements

We thank Professor Jeff Salowe, Professor Samir Khuller, and Dr. Neal Young for illuminating discussions of the shallow-light literature and their works in this area. We also thank the authors of [25] for use of their simulator code, and Professor Wayne W.-M. Dai for releasing MCM simulation parameters for our study. The anonymous referees gave many helpful comments on the original draft of this work.

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", *Proc. IEEE Intl. Symp. on Circuits and Systems*, Chicago, May 1993, pp. 1869-1872 (also issued as UCLA CS Dept. TR-920051, Fall 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] B. Awerbuch, A. Baratz and D. Peleg, "Efficient Broadcast and Light-Weight Spanners", *manuscript*, November 1991.

[4] H. B. Bakoglu, *Circuits, Interconnections and Packaging for VLSI*, Addison-Wesley, 1990, pp. 81-133.

[5] K. D. Boese, J. Cong, A. B. Kahng, K. S. Leung and D. Zhou, "On High-Speed VLSI Interconnects: Analysis and Design", *Proc. Asia-Pacific Conf. on Circuits and Systems*, (1992), pp. 35-40.

[6] K. D. Boese, A. B. Kahng, B. A. McCoy and G. Robins, "Fidelity and Near-Optimality of Elmore-Based Routing Constructions", in *Proc. IEEE Intl. Conf. on Computer Design*, Boston, October 1993, pp. 81-84.

[7] B. Chandra, G. Das, G. Narasimhan and J. Soares, "New Sparseness Results on Graph Spanners", *Proc. 8th Ann. Symp. Computational Geometry* June 1992, pp. 192-201.

[8] 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.

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

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

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

[12] J. Ho, G. Vijayan, and C. K. Wong, "New Algorithms for the Rectilinear Steiner Tree Problem", *IEEE Trans. on CAD* 9(2), Feb 1992, pp. 185-193.

[13] S. Khuller, B. Raghavachari and N. Young, "Balancing Minimum Spanning and Shortest Path Trees", *Proc. ACM/SIAM Symp. on Discrete Algorithms*, January 1993, pp. 243-250.

[14] S. Kim, R. M. Owens, and M. J. Irwin "Experiments with a Performance Driven Module Generator", *Proc. ACM/IEEE Design Automation Conf.*, 1992, pp. 687-690.

[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] H.-P. Lenhof, J. S. Salowe and D. E. Wrege, "New Methods to Mix Shortest-Path and Minimum Spanning Trees", *manuscript*, 1993.

[17] 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.

[18] R. C. Prim, "Shortest Connecting Networks and Some Generalizations", *Bell System Tech. J.* 36 (1957), pp. 1389-1401.

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

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

[21] J. S. Salowe, D. S. Richards and D. Wrege, "Mixed Spanning Trees", *Proc. Great Lakes Symp. on VLSI*, March 1993, pp. 62-66.

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

[23] R. E. Tarjan, *Data Structures and Network Algorithms*, 1983.

[24] J. Vlach, J. A. Barby, A. Vannelli, T. Talkhan and C. J. Shi, "Group Delay as an Estimate of Delay in Logic", *IEEE Trans. on CAD* 10(7) (1991), pp. 949-953.

[25] D. Zhou, S. Su, F. Tsui, D. S. Gao and J. Cong, "Analysis of Trees of Transmission Lines", *technical report* UCLA CSD-920010, March 1992.