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

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. 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.

I. INTRODUCTION AND MOTIVATION

With the scaling of device technology and die size, interconnection delay now contributes up to 50-70% 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.

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 $r_{ij}$ routing cost. The cost of the shortest $v_0$-$v_i$ path in $G$ is denoted by $R_i$. and $R = \max_{1 \leq i \leq n} R_i$ 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 $r(T) = \max_{1 \leq i \leq n} l_i$, and the cost of $T$ is $w(T) = \sum_{e_{ij} \in E'} d_{ij}$. 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 $c_0$ denote the node capacitance of $v_0$. Given $T$, let $T_i$ denote the subtree of $T$ that is below edge $e$ when $T$ is rooted at $v_0$, and let $C_e$ denote the sum of node and edge capacitances of $T_i$. Finally, let $C_0$ indicate the total capacitance of $T$. The Elmore delay along edge $e$ is $r_e \cdot (\frac{C_e}{2} + C_0)$. 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

$$t(v_i) = r_0 \cdot C_0 + \sum_{e \in \text{path}(v_0, v_i)} r_e \cdot \left( \frac{C_e}{2} + C_0 \right).$$

If $r_e$ and $c_e$ are proportional to the length of $e$, then the $r_0 \cdot C_0$ term in (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 $r_0 \cdot C_0$ 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 $\frac{C_e}{2}$ (where $l$ is unit wire resistance; $\frac{C_e}{2}$ is the "resistance ratio" discussed in [5] and [9]) have generally decreased with the trend to submicron CMOS and MCM technologies (see Table I), suggesting that minimum-cost routing is becoming less correlated with performance-driven routing.

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

II. 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 Fig. 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 $(\alpha, \beta)$-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) $w(T) \leq \beta \cdot w(T_M)$.

A spanning tree construction is shallow-light if for some constants $\alpha$ and $\beta$ it always returns an $(\alpha, \beta)$-tree. Since the cost of a spanning tree rises as its radius is constrained, we are typically interested in
TABLE I
INTERCONNECT PARAMETERS FOR THREE CMOS IC TECHNOLOGIES AND AN MCM TECHNOLOGY

<table>
<thead>
<tr>
<th>Name</th>
<th>IC1</th>
<th>IC2</th>
<th>IC3</th>
<th>MCM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technology</td>
<td>2 μm CMOS</td>
<td>1.2 μm CMOS</td>
<td>0.5 μm CMOS</td>
<td>MCM</td>
</tr>
<tr>
<td>( r_0 )</td>
<td>164.0 ( \Omega )</td>
<td>212.1 ( \Omega )</td>
<td>270.0 ( \Omega )</td>
<td>25.0 ( \Omega )</td>
</tr>
<tr>
<td>( r = \text{unit wire resistance} )</td>
<td>0.033 ( \Omega/\mu )</td>
<td>0.073 ( \Omega/\mu )</td>
<td>0.112 ( \Omega/\mu )</td>
<td>0.008 ( \Omega/\mu )</td>
</tr>
<tr>
<td>( \epsilon = \text{unit wire capacitance} )</td>
<td>0.019 ( \text{fF}/\mu )</td>
<td>0.028 ( \text{fF}/\mu )</td>
<td>0.039 ( \text{fF}/\mu )</td>
<td>0.06 ( \text{fF}/\mu )</td>
</tr>
<tr>
<td>( \gamma ) ( \times 10^5 ) ( \mu )</td>
<td>5.7 ( \text{fF} )</td>
<td>7.06 ( \text{fF} )</td>
<td>1.0 ( \text{fF} )</td>
<td>1000 ( \text{fF} )</td>
</tr>
<tr>
<td>( z_{\text{chip size}} )</td>
<td>1x1 ( \text{cm}^2 )</td>
<td>1x1 ( \text{cm}^2 )</td>
<td>1x1 ( \text{cm}^2 )</td>
<td>10x10 ( \text{cm}^2 )</td>
</tr>
</tbody>
</table>

Parasitics for the IC1 and IC2 technologies are provided by MOSIS; IC3 parasitics are courtesy of MCNC; MCM interconnect parasitics are courtesy of Professor W. W.-M. Dai of the University of California, Santa Cruz and correspond to data provided by AT&T Microelectronics Division. The \( r_0 \) values are scaled drive resistances. Sink loading capacitances \( (c_{\text{v}}) \) are derived for minimum-size transistors.

A. The ALG1 Tradeoff

Primer'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

\[
d_{ij} \text{ s.t. } v_j, v_i \in V - T. \tag{2}
\]

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

\[
l_j + d_{ij} \text{ s.t. } v_i, v_j \in V - T. \tag{3}
\]

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

\[
(c \cdot l_j) + d_{ij} \text{ s.t. } v_i, v_j \in V - T \tag{4}
\]

for some choice of \( 0 \leq c \leq 1 \). When \( c = 0 \), ALG1 is identical to Primer'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 \( c = \frac{1}{2} \) and \( c = \frac{3}{2} \) are shown in Figs. 2(a)-(c).

Observation 1: ALG1 constructs a tree \( T \) with \( c \cdot l_j \leq R \) for all sinks \( v_i \).

Proof: By strong induction; see Fig. 3. Assume that for every ancestor \( v_j \) of \( v_i \) in \( T_S \), we have \( c \cdot l_j \leq R \). Consider a snapshot of \( T \) immediately before ALG1 adds sink \( v_i \) to \( T \) via edge \( e_m \), 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 \( v_k \), \( c \cdot l_m + d_{m,j} \leq c \cdot l_j + d_{j,k} \). By the inductive hypothesis, \( c \cdot l_j \leq R_j \), and by the principle of optimality of shortest paths, \( R_i + d_{i,j} = R_k \leq R \). Since \( c \cdot l_i \leq c \cdot l_m + d_{m,j} \), combining these inequalities yields \( c \cdot l_i \leq R \).

Observation 2: ALG1 is not shallow-light for general graph instances [1].
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 \( \max\{|v_1|, d_{v_1}\} \) and then generalize this to the objective of (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.

When \( p = \infty \), the ALG2 objective reduces to \( \max\{|v_1|, d_{v_1}\} \), 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 \( v_j \) according to (5) which also minimizes \( d_{v_j} \). 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 \( k \) 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 Fig. 4. Because \( e_1 \) and \( e_2 \) are distinct, the Prim objective implies \( e_1 < e_2 \). Since ALG2 added \( e_2 \), \( \max\{|v_1|, d_{v_1}\} \leq \max\{|l_1|, e_1| \}. Moreover, if \( \max\{|l_1|, e_1| \neq \max\{|l_2|, e_2| \}, the tie-breaking rule would force ALG2 to choose \( e_1 \), hence \( \max\{|l_1|, e_1| < \max\{|l_2|, e_2| \}, 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 \( l_2 \) path has less cost than \( e \), Prim’s algorithm could not have added \( e \) before completely adding the \( l_2 \) path, contradicting the inductive hypothesis.

Salowe et al. [21] have shown that ALG2 constructs a tree \( T \) with \( l_i \leq R_i n^{-1/p} \) for all sinks \( v_j \), and that this bound is tight. It is easy for some choice of \( 1 \leq p < \infty \). Sample executions of ALG2 for \( p = 3 \) and \( p = \frac{3}{2} \) are shown in Fig. 2(c)-(d).

**Note:** 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, the observation regarding Dijkstra’s algorithm suggests our ALG2 tradeoff: iteratively add edge \( e_{ij} \) and sink \( v_j \) to \( T \), where \( e_{ij} \) and \( v_j \) are chosen to minimize

\[
||P_0, d_{ij}|| \text{ s.t. } v_j \in T, v_i \in V - T
\]
TABLE III
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 THE AVERAGE BEST PARAMETER VALUE FOR EACH ALGORITHM.

<table>
<thead>
<tr>
<th>Spanning Trees</th>
<th>Max sink delay vs. MST (best parameter)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ALG.</td>
</tr>
<tr>
<td>4</td>
<td>ALG1</td>
</tr>
<tr>
<td></td>
<td>ALG2</td>
</tr>
<tr>
<td></td>
<td>KRY</td>
</tr>
<tr>
<td></td>
<td>BRBC</td>
</tr>
<tr>
<td>8</td>
<td>ALG1</td>
</tr>
<tr>
<td></td>
<td>ALG2</td>
</tr>
<tr>
<td></td>
<td>KRY</td>
</tr>
<tr>
<td></td>
<td>BRBC</td>
</tr>
<tr>
<td>16</td>
<td>ALG1</td>
</tr>
<tr>
<td></td>
<td>ALG2</td>
</tr>
<tr>
<td></td>
<td>KRY</td>
</tr>
<tr>
<td></td>
<td>BRBC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Spanning Trees</th>
<th>Avg sink delay vs. MST (best parameter)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ALG.</td>
</tr>
<tr>
<td>4</td>
<td>ALG1</td>
</tr>
<tr>
<td></td>
<td>ALG2</td>
</tr>
<tr>
<td></td>
<td>KRY</td>
</tr>
<tr>
<td></td>
<td>BRBC</td>
</tr>
<tr>
<td>8</td>
<td>ALG1</td>
</tr>
<tr>
<td></td>
<td>ALG2</td>
</tr>
<tr>
<td></td>
<td>KRY</td>
</tr>
<tr>
<td></td>
<td>BRBC</td>
</tr>
<tr>
<td>16</td>
<td>ALG1</td>
</tr>
<tr>
<td></td>
<td>ALG2</td>
</tr>
<tr>
<td></td>
<td>KRY</td>
</tr>
<tr>
<td></td>
<td>BRBC</td>
</tr>
</tbody>
</table>

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.

IV. 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 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.

A. 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 II). 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 $T_{ALG1}$ for $c \geq 1.5$, we study the family of 51 trees generated by BRBC with the parameter $c$ ranging from 0 to 1 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 1 cm by 1 cm Manhattan square; each point in Fig. 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 tradeoff, 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.

B. 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 I 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 SPICE3c [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 5.0 V, 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 III 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 maximum 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 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 Fig. 6), we believe that KRY benefits from its tendency to branch early from \( r_0 \), 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.

### C. 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

![Fig. 5](image_url)

*Fig. 5. Graph of radius ratio (\( \frac{r(T)}{r(T_s)} \)) versus cost ratio (\( \frac{w(T)}{w(T_s)} \)) 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, (a) averaged over 250 instances, and (b) the same experiments with the edges overlapped to induce a Steiner topology.*

![Fig. 6](image_url)

*Fig. 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.*
than its spanning tree precursor.)
result since the overlapping process diminishes the star-like nature of
cost without affecting radius, we can afford spanning trees that use

to Steiner trees in this manner. The performance-driven spanning

trees can have high-degree nodes, and do not always satisfy
their code is 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.\textsuperscript{3}

Fig. 5(b) and Table IV 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 point is 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 ALGI
or ALGI-Steiner (i.e., ALGI followed by greedy edge-overlapping to
create a Steiner tree) with fixed $c = 1.00$ already constitutes a reason-
ably good heuristic Steiner arborescence, or "A-tree," construction\textsuperscript{2}.
We may similarly compare our constructions against the standard
minimum Steiner tree heuristic of edge-overlapping a minimum
spanning tree (this is identical to ALGI-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 ALGI execution
using the fixed, best $c$ value from Table III will attain ratios of 0.788,
0.590, and 0.461, respectively. When edge-flipping is added in ALGI-
Steiner and we use the best $c$ values from Table IV, the average-delay
ratios are 0.756, 0.586, and 0.468, respectively. (Delay ratios obtained
by considering the family of ALGI trees can be read off directly from
Tables III and IV.)

\textsuperscript{2}Reference [9] presents numerical results for one set of interconnect
technology parameters, namely, MCM; the parameters and simulation method-
ology are identical to those we use here. By normalizing the reported A-tree
results to those of BRBC with parameter $c = 1.0$—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 ALGI as given in Tables III and IV, we obtain the
following results: ii) ALGI/BRBC-$1.0 = 0.805$ (4 sinks), 0.598 (8 sinks),
0.463 (16 sinks); and iii) ALGI-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 ALGI trees,
the results improve to ii\textsuperscript{3} ALGI/BRBC-$1.0 = 0.702$ (4 sinks), 0.547 (8 sinks),
0.431 (16 sinks); and iii\textsuperscript{3} ALGI-Steiner/BRBC-$1.0 = 0.738$ (4 sinks), 0.598
(8 sinks), 0.502 (16 sinks).

\textsuperscript{3}Our greedy edge-overlapping method simply examines the bounding boxes
of every pair of adjacent edges in the tree, and calculates the cost reduction
achieved 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 \textit{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 9.9\% 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.

TABLE IV

<table>
<thead>
<tr>
<th>Steiner Trees</th>
<th>Max sink delay vs. MST (best parameter)</th>
</tr>
</thead>
<tbody>
<tr>
<td>#sinks</td>
<td>ALG1</td>
</tr>
<tr>
<td>4</td>
<td>ALG1</td>
</tr>
<tr>
<td></td>
<td>ALG2</td>
</tr>
<tr>
<td></td>
<td>KRY</td>
</tr>
<tr>
<td></td>
<td>BRBC</td>
</tr>
<tr>
<td>8</td>
<td>ALG1</td>
</tr>
<tr>
<td></td>
<td>ALG2</td>
</tr>
<tr>
<td></td>
<td>KRY</td>
</tr>
<tr>
<td></td>
<td>BRBC</td>
</tr>
<tr>
<td>16</td>
<td>ALG1</td>
</tr>
<tr>
<td></td>
<td>ALG2</td>
</tr>
<tr>
<td></td>
<td>KRY</td>
</tr>
<tr>
<td></td>
<td>BRBC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Steiner Trees</th>
<th>Avg sink delay vs. MST (best parameter)</th>
</tr>
</thead>
<tbody>
<tr>
<td>#sinks</td>
<td>ALG1</td>
</tr>
<tr>
<td>4</td>
<td>ALG1</td>
</tr>
<tr>
<td></td>
<td>ALG2</td>
</tr>
<tr>
<td></td>
<td>KRY</td>
</tr>
<tr>
<td></td>
<td>BRBC</td>
</tr>
<tr>
<td>8</td>
<td>ALG1</td>
</tr>
<tr>
<td></td>
<td>ALG2</td>
</tr>
<tr>
<td></td>
<td>KRY</td>
</tr>
<tr>
<td></td>
<td>BRBC</td>
</tr>
<tr>
<td>16</td>
<td>ALG1</td>
</tr>
<tr>
<td></td>
<td>ALG2</td>
</tr>
<tr>
<td></td>
<td>KRY</td>
</tr>
<tr>
<td></td>
<td>BRBC</td>
</tr>
</tbody>
</table>
V. CONCLUSION

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 $T_{xy}$ and insert shortest paths as needed to maintain a prescribed radius bound. In contrast, our ALG1 and ALG2 constructions directly combine the recurrences for Prim's MST 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 "probably good" BRBC approach are particularly substantial. Our delay reductions over fixed constructions are also substantial over the "probably good" BRBC approach are particularly substantial. It is therefore of interest to pursue integration of ALG1 within existing performance-driven global routers.

ACKNOWLEDGMENT

The authors would like to thank Professor J. Salowe, Professor S. K黄ler, and Dr. N. 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 W. 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


A Preprocessor for Improving Channel Routing Hierarchical Pin Permutation

C. Y. Roger Chen, Cliff Yungchin Hou, and Bradley S. Carlson

Abstract—In standard cell design, many cell terminals and gates are permutable, and it is important for a channel router to take advantage of this to obtain better results. An efficient hierarchical algorithm is presented to determine the proper positions of permutable gates and cell terminals such that the results of the subsequent channel routing can be significantly improved. Experimental results show that our proposed algorithm considerably reduces the number of tracks and vias, and its time complexity is linear in the number of cell terminals.

I. INTRODUCTION

Channel routing is one of the critical problems in VLSI design. Algorithms for channel routing with fixed terminals have been studied extensively [11]-[5]. Optimal or near optimal results have been obtained by these algorithms. The problem of permutable channel routing in which some of the terminals are interchangeable has received considerable attention in recent years [6]-[12]. For example, in programmable logic cells (e.g., PLA's and ROM's), the terminals

Manuscript received August 20, 1991; revised January 19, 1995. This paper was recommended by Associate Editor R. H. J. M Otten.

The authors are with the Department of Electrical and Computer Engineering, Syracuse University, Syracuse, NY 13244 USA.
IEEE Log Number 9411259.