New Performance Driven Routing Techniques With Explicit Area/Delay Tradeoff and Simultaneous Wire Sizing*

John Lillis, Chung-Kuan Cheng
CSE Dept., UCSD
La Jolla, CA 92093

Ting-Ting Y. Lin
ECE Dept., UCSD
La Jolla, CA 92093

Ching-Yen Ho
LSI Logic Corp.
Milpitas, CA 95035

Abstract

We present new algorithms for construction of performance driven Rectilinear Steiner Trees under the Elmore delay model. Our algorithms represent a departure from previous approaches in that we derive an explicit area/delay tradeoff curve. We achieve this goal by limiting the solution space to the set of topologies induced by a permutation on the sinks of the net. This constraint allows efficient identification of optimal solutions while still providing a rich solution space. We also incorporate simultaneous wire sizing. Our technique consistently produces topologies equalling the performance of previous approaches with substantially less area overhead.

1 Introduction

In recent years, interconnect delay has become an increasingly critical factor in VLSI systems, in some cases accounting for over 50% of overall delay. This trend is a result of the increased resistance of interconnect as feature sizes enter the sub-micron range and will become more dramatic in the future. To combat this trend, many methods have been proposed to reduce interconnect delay. Two such techniques are the focus of this paper: performance driven routing (e.g. [1],[4], [16],[11],[8]) and wire sizing (e.g. [3], [15], [13]).

Previous Work: Among performance driven routing techniques, the SERT algorithm of Boese et. al. [1] has produced some of the best delay figures reported in the literature. While the performance of the routing trees produced by SERT is promising, the area overhead is discouragingly large. Because of the greedy nature of the algorithm, it often results in high area "star-like" topologies. The later work of Hodes et. al. [8] had some success in reducing this tendency of SERT by use wire sizing to drive the construction.

Other performance driven routing work includes the A-Tree algorithm of Cong et. al. [4] which attempts to find a min-area shortest paths tree. Thus, the A-Tree algorithm uses a geometric abstraction and a linear delay model and does not address such common phenomena as asymmetric loads (and in fact does not take any technological parameters into account). Nevertheless, the routing trees it produces are often of high quality. The work of Vittal and Marek-Sadowska [16] used the abstraction of Alphabetic Trees to heuristically construct routing trees. Their reported results are comparable to those of SERT in both delay and area.

*This work was supported in part by Grants from the NSF project MIP-9315794 and California MICRO program.

Early work in wire-sizing includes that of Cong et. al. (e.g., [3]) which established its effectiveness in optimizing a static topology and provided theoretical insights. The problem formulation proposed was a weighted sum of source-to-sink delays rather than maximum delay or required arrival time. Later, in [15], Sapatnekar addressed the intuitively appealing formulation of minimizing area subject to timing constraints on the sinks. He studied the continuous variant of the problem and solved it by convex programming techniques followed by a heuristic mapping phase to discretize the solution. Recently an efficient dynamic programming solution to the discrete wire sizing problem was given in [13]. A novel feature of this algorithm is that it computes the entire area/delay tradeoff curve rather than a single solution.

Contributions of This Paper: This paper presents two algorithms: the P-Tree$_A$ algorithm whose goal is area minimization and the P-Tree$_{AT}$ algorithm which derives an area/delay tradeoff curve under the Elmore delay model and also incorporates simultaneous wire-sizing. Central to this work is the notion of routing topologies being induced by a permutation on the sinks of a net and the mapping of such topologies to a routing domain. Thus, the routing topologies we produce are *Permutation-constrained routing Trees*, giving rise to the name *P-Tree*.

The algorithm has achieved promising results versus previously proposed approaches in terms of area required to achieve a given delay and the AT metric (the product of area and max delay). For instance, previous approaches can consume over 70% more area on average than the P-Tree$_{AT}$ solution while achieving the same or worse delay.

2 Preliminaries

Delay Model: We adopt the Elmore delay model [5] because of its fidelity with respect to physical delays [2] and its ease of computation. For a wire segment $e = (u, v)$, let r_e and c_e be the resistance and capacitance of e respectively (both of which are proportional to wire length l_e). Further, let $c(T_v)$ be the load at node v. The Elmore delay of the segment is expressed as $r_e(\frac{c_e}{2} + c(T_v))$.

When wire sizing is taken into account, we need a model for resistance r_e, and capacitance c_e as a function of width w_e. A typical model is the following:

$$c_e = \alpha l_e \cdot w_e \qquad r_e = \beta l_e / w_e$$

where α and β are characteristic constants. For consistency with previous work, we adopt this model.

Delay of a driver g is defined similarly as $d_g + r_g \cdot c_l$ where d_g is the intrinsic delay, r_g is the output resistance of the driver and c_l is the capacitive load on g's output. Since d_g is constant, it may be neglected in relative comparisons of routing algorithms.

The required arrival time for a routing tree T driven by gate g is

$$q(T) = \min_{sinks\ u} \{q_u - delay(g \to u)\}$$

where q_u is the required arrival time at sink u. If $q(T) \geq 0$, T is said to meet its timing requirements. A special case of required arrival time is maximum delay: letting $q_u = 0\ \forall u$, the max source to sink delay is $-q(T)$.

Graphs, Trees and Permutations: From Hanan's theorem [7] we know that that there always exists a Rectilinear Steiner Minimum Tree (RSMT) for a terminal set N ($n = |N|$) where all Steiner points are drawn from the n^2 points formed by the intersection of horizontal and vertical lines through the terminals. The grid formed by this set of points forms the the *Grid Graph* of N [9], $GG(N)$ where vertices are the terminals and points of intersection and there is an edge between pairs of vertices adjacent on a grid line; the length of an edge is the distance between its end-points.

Our algorithms produce routing topologies embedded in a graph structure such as $GG(N)$. We use $v_{\{r,c\}}$ to denote the vertex in $GG(N)$ located at the r'th row and c'th column (position $(1, 1)$ being upper left). For convenience, we also use v_i and v_d to denote the vertex in $GG(N)$ coincident with the i'th pin of a permutation and the vertex coincident with the driving pin respectively.

As previously stated, we constrain routing topologies to be induced by a sink permutation. We give the following definitions relating permutations and routing topologies.

Definition 1 Permutation Induced Abstract Topology: *Consider a permutation π on the sinks of terminal set N. A binary tree T is an* abstract topology *induced by π if its leaves are identified with the terminals in N and it obeys the ordering imposed by π when T is interpreted as a binary search tree (the driver being implicitly attached to the root).*

Definition 2 Abstract Topology Embedding: *Given an abstract topology T and a graph $G = (V, E)$ an embedding of T into G is a mapping of the internal nodes of T to V.*

To avoid confusion, we use the term "abstract topology" to emphasize that such a topology is not a true routing topology in that its internal nodes are not mapped. Only when such a topology is embedded in the plane or a routing graph such as $GG(N)$, does a physical routing topology result.

The idea of *consistency* given in the following definition is a convenient way of assessing the quality of a permutation.

Definition 3 Consistency: *Given terminals set N and a tree T embedded in a routing graph and connecting N, a permutation π on N is* consistent *with T if π can induce a structure isomorphic to T (up to 0-length edges). Further, let H be a binary tree with all points $p \in N$ represented as leaves. We call H a* hierarchical decomposition *of N. H is* consistent *with T if* all possible depth-first traversals of *H visit the leaves in an order consistent with T.*

Thus, if an algorithm for constructing a permutation ensures consistency with the MST, we guarantee that the min area solution induced by π is no worse than $\frac{3}{2}$ times the optimal Steiner tree [10]. Such an algorithm is sketched in the next section. Previous permutation based approaches such as [16] provide no such assurance. Rather, they use a circular ordering of the sinks which can result in unbounded overhead vs. the MST.

3 Finding High Quality Permutations

The method we propose for constructing high quality permutations is broken into three phases: (1) construction of a hierarchical decomposition consistent with a given structure such as the MST, (2) reorientation of the hierarchical structure such that the driver is attached to the root while maintaining consistency and (3) application of a dynamic programming algorithm to further optimize the induced permutation. Pseudo-code of an algorithm for phase 1 appears in Figure 2. The algorithm is easily generalized to produce hierarchies consistent with a given Steiner tree.

Phase 3 is based on the following observation. Since the hierarchy from step (1) is consistent with the given topology, *any* depth-first traversal of the hierarchy yields a permutation consistent with the given topology (e.g., MST). We propose that an appropriate metric to choose from among these possible permutations is the tour length in the traditional sense of the Travelling Salesman Problem. The intuition is that good TSP tours provide good clustering information which identifies good candidate sets for sub-trees of a Steiner Tree. This problem can be solved in $O(n^4)$ time by dynamic programming [14] – i.e., over all DFS orderings, we can identify the one with minimal tour length.

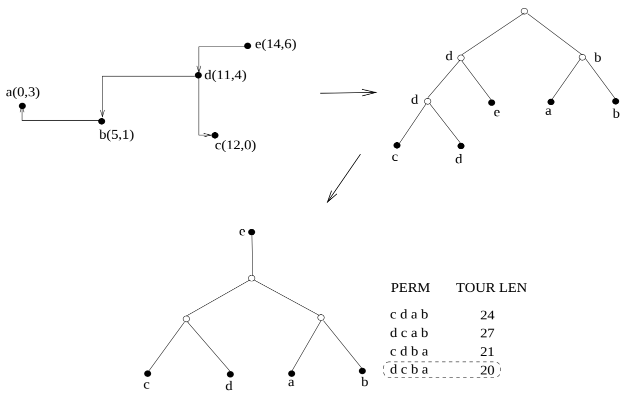
For space considerations we do not give the details of this process here referring the reader to [14]. However, Figure 2 illustrates the process. In the upper left we have the MST of a point set from which we construct a hierarchical decomposition. Then the hierarchy is reoriented, placing the driver e at the root and finally, the permutation *"d c b a"* is selected over all DFS traversals of the tree.

```
Algorithm: MST_to_hierarchy
  Let $T_m$ be a RMST on terminal set $N$
  For each $v \in T_m$, let $h(v)$ be a single node tree labeled $v$
  Repeat $n - 1$ times
     1. select a leaf node $v$ and its parent $u$ in $T_m$
     2. delete edge $(u, v)$ from $T_m$
     3. replace $h(u)$ with a new tree $t$ where
          $t.left = h(u)$ and
          $t.right = h(v)$
  return last tree formed
```

Figure 1: *Constructing Hierarchy Consistent with MST*

4 The P-Tree Algorithms

This section describes the P-Tree algorithms which take as input a sink permutation established by the techniques of the previous section. Using the the example point set begun in Figure 2, we give Figure 3 where an abstract topology is induced in 3(a) and is embedded in $GG(N)$ to form the routing topology in 3(b) (yielding, in this case, a min-area



PERM	TOUR LEN
c d a b	24
d c a b	27
c d b a	21
d c b a	20

Figure 2: *Construction of Sink Permutation*

Steiner Tree). It is this process which the P-Tree algorithms perform optimally.

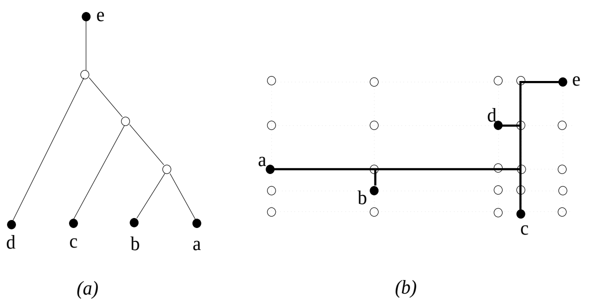


Figure 3: *A permutation-induced abstract topology and an embedding in $GG(N)$.*

Our algorithms inductively compute the following solutions or solution sets for the given sink permutation:

$S(v,i,j)$: the cost of the optimal solution(s) over all permutation induced (and embedded) routing topologies driving sinks $i..j$ and rooted at v $(i \leq j)$.

$S_b(v,i,j)$: the cost of the optimal solution(s) over all permutation induced (and embedded) routing topologies driving sinks $i..j$ and rooted at v where v is additionally constrained to to be a *branching point* $(i < j)$.

When minimizing area only, $S(v,i,j)$ and $S_b(v,i,j)$ are scalars giving the solution area. However, in performance driven routing we are also interested in the capacitance and timing properties of sub-solutions. To capture these properties of sub-solutions we maintain sets of load, required-time, or (c,q), pairs much in the same manner as in [13].

While there are fundamental differences in the sub-solutions computed by P-Tree$_A$ and P-Tree$_{AT}$, both algorithms follow the same basic structure given in Figure 4. The details of the individual steps in Figure 4 for P-Tree$_A$ and P-Tree$_{AT}$ are given in the next two sections respectively.

4.1 The P-Tree$_A$ Algorithm

The first step in P-Tree$_A$ is computation of base cases $S(v,i,i)$. Since we are just interested in the length of a wire from v to pin i, we set

$$S(v,i,i) = d(v,v_i) \quad \forall i \in \{1..n-1\}$$

```
1. Compute $S(v,i,i)$    $\forall v \in V, 1 \leq i < n$
2. For $I = 1..n-2$
3.     For $i = 1..n-1-I$
4.         $j = i + I$
5.         Compute $S_b(v,i,j)$    $\forall v \in V$
6.         Compute $S(v,i,j)$    $\forall v \in V$
7. $S(v_d,1,n-1)$ now gives the sub-solution(s)
       to be paired with the driver.
```

Figure 4: *Algorithm Framework*

where $d(v,v_i)$ is the distance between vertices v and v_i in $GG(N)$.

We compute $S_b(v,i,j)$ by visiting all possible partition points k as follows

$$S_b(v,i,j) = \min_{k \in \{i..j-1\}} \{S(v,i,k) + S(v,k+1,j)\}$$

Note that since $k-i < j-i$ and $j-k-1 < j-i$, $S(v,i,k)$ and $S(v,k+1,j)$ have been previously computed.

A naive approach to computing $S(v,i,j)$ is to implement the following.

$$S(v,i,j) = \min_{v' \in V} \{d(v,v') + S_b(v',i,j)\}$$

To compute $S(v,i,j)$ for a fixed i,j and all $v \in V$ by this method takes $O(n^4)$ time. Because of the structure of the grid graph, this complexity is unnecessary.

In fact, we can compute $S(v,i,j)$ for all $v \in V$ in $O(n^2)$ time by noticing that the optimal branching point at v is either v itself or the same as that of one of its neighbors. Without loss of generality, assume that the wire connecting v to its optimal branching point v' first travels along v's column to v''s row and then finishes the trip horizontally to v'. This leads to a four phase strategy given in Figure 5 (for fixed i,j).

We compute intermediate solutions $LS(v)$, $RS(v)$ and $US(v)$, the meanings of which are summarized as follows.

$LS(v)$:	area of the optimal single-stem tree rooted at v with branching point v' constrained to be at v or to v's *left* in v's row.
$RS(v)$:	area of the optimal single-stem tree rooted at v with branching point v' constrained to be in the same *row* as v.
$US(v)$:	area of the optimal single-stem tree rooted at v with branching point v' constrained to be in v's row or a row *above* v.

This is done in four linear passes over the graph where each pass is similar to the computation of the prefix sum or prefix max of a list of numbers.

This yields an $O(|V|) = O(n^2)$ complexity for computing, $S(v,i,j)$ for a fixed i,j and all $v \in V$. Thus the overall complexity of *P-Tree$_A$* is dominated by the computation of $S_b(v,i,j)$ which is $O(n)$ for each v,i,j triple. Since there are $O(n^4)$ such triples, this gives an overall complexity of $O(n^5)$.

Finally, we can show the optimality of the solution computed by P-Tree$_A$ inductively.

Lemma 1 *$S(v_d,1,n-1)$ computed by the P-Tree$_A$ algorithm is the minimum area over all possible Rectilinear Steiner trees induced by given sink ordering π. This solution is computed in $O(n^5)$ time.*

```
/* Phase 1 */
$LS(v_{\{r,1\}}) \leftarrow S_b(v_{\{r,1\}}) \quad \forall r \in \{1..n\}$
for $r = 1$ to $n$
    for $c = 2$ to $n$
        $LS(v_{\{r,c\}}) \leftarrow \min\{S_b(v_{\{r,c\}}),$
            $LS(v_{\{r,c-1\}}) + d(v_{\{r,c\}}, v_{\{r,c-1\}})\}$
/* Phase 2 */
$RS(v_{\{r,n\}}) \leftarrow LS(v_{\{r,n\}}) \quad \forall r \in \{1..n\}$
for $r = 1$ to $n$
    for $c = n-1$ to 1
        $RS(v_{\{r,c\}}) \leftarrow \min\{LS(v_{\{r,c\}}),$
            $RS(v_{\{r,c+1\}}) + d(v_{\{r,c\}}, v_{\{r,c+1\}})\}$
/* Phase 3 */
$US(v_{\{1,c\}}) \leftarrow RS(v_{\{1,c\}}) \quad \forall c \in \{1..n\}$
for $c = 1$ to $n$
    for $r = 2$ to $n$
        $US(v_{\{r,c\}}) \leftarrow \min\{RS(v_{\{r,c\}}),$
            $US(v_{\{r-1,c\}}) + d(v_{\{r,c\}}, v_{\{r-1,c\}})\}$
/* Phase 4 */
$S(v_{\{n,c\}}, i, j) \leftarrow US(v_{\{n,c\}}) \quad \forall c \in \{1..n\}$
for $c = 1$ to $n$
    for $r = n-1$ to 1
        $S(v_{\{r,c\}}, i, j) \leftarrow \min\{US(v_{\{r,c\}}),$
            $S(v_{\{r+1,c\}}, i, j) + d(v_{\{r,c\}}, v_{\{r+1,c\}})\}$
```

Figure 5: *Computation of $S(v,i,j)$ $\forall v \in V$ in P-Tree$_A$*

4.2 The P-Tree$_{AT}$ Algorithm

The basic structure of P-Tree$_{AT}$ is essentially identical to P-Tree$_A$ except in the primitives we use to manipulate solutions. Recall that both $S_b(v,i,j)$ and $S(v,i,j)$ were scalars in P-Tree$_A$. In contrast, P-Tree$_{AT}$ manipulates sets of *load, required-time* pairs. We refer to such sets as *cq-sets* and introduce new primitives for manipulating them. For instance, a load, required-time pair $(c,q) \in S(v,i,j)$ indicates that there exists a legal routing topology rooted at v and driving sinks $i..j$ which has lumped capacitance c at v (also proportional to routing area) and required-arrival time q at v. Note that since we may be performing simultaneous wire sizing, we not only need to determine a routing topology but also the width of each wire segment between Steiner nodes.

In [17] van Ginneken gave the following simple property of cq-sets (also used later in [13]).

Property 1 *For $(c,q), (c',q') \in S$, if $c' \geq c$ and $q' < q$ then (c',q') is sub-optimal.*

This property holds since a larger capacitance can only slow down a path from the driver. Thus, given a cq-set S, all pairs (c',q') as described by Property 4.1 may be eliminated. As a result we can organize cq-sets in strictly increasing order of c *and* q.

To manipulate cq-sets, we need some primitive operations as follows.

join_cq_sets(S_1, S_2): Let solution sets S_1 and S_2 give solutions rooted at v, where S_1 (S_2) drives sinks $i..k$ ($k+1..j$) for some $k \in \{i..j-1\}$. This primitive produces a set of solutions rooted at v driving sinks $i..j$, the size of which is proportional to $|S_1| + |S_2|$.

augment_cq_set(S, l, w): This primitive takes cq-set S and computes a new cq-set where each $(c,q) \in S$ is augmented by a wire of length l and width w. This is used in constructing single stem solution sets $S(v,i,j)$.

prune_cq_set(S): This primitive takes a cq-set S and eliminates sub-optimal solutions by Property 1.

The routine *join_cq_sets*(S_1, S_2) is similar to the merging of two sorted lists. Pseudo-code appears in Figure 6 (sets S_1 and S_2 are in increasing order of c and q and indexed).

```
$S \leftarrow \emptyset$
$i \leftarrow 1$ ; $j \leftarrow 1$
While ($i \leq |S_1|$ and $j \leq |S_2|$)
    Let $(c_1, q_1) = S_1[i]$
    Let $(c_2, q_2) = S_2[j]$
    $S \leftarrow S \cup \{(c_1 + c_2, \min(q_1, q_2))\}$
    If ($q_1 \leq q_2$)      /* $S_1$ Critical */
        $i \leftarrow i + 1$
    If ($q_2 \leq q_1$)      /* $S_2$ Critical */
        $j \leftarrow j + 1$
return $S$.
```

Figure 6: *Primitive join_cq_sets(S_1, S_2)*

The routine *augment_cq_set*(S, l, w) examines each $(c,q) \in S$ and produces (c', q') where

$$c' = c + \alpha w l \qquad\qquad q' = q - \frac{\beta l}{w}\left(\frac{\alpha l w}{2} + c\right).$$

Finally, we implement *prune_cq_set*(S) by making a linear pass over S in increasing order of c eliminating sub-optimal solutions as we go.

Given these primitives, we now describe the core of the algorithm. To get started, we have the base case

$$S(v_i, i, i) \leftarrow \{(c_i, q_i)\} \qquad \forall i \in \{1..n-1\}$$

where c_i and q_i are the input capacitance and required arrival time for sink i respectively. Computation of $S(v,i,i)$ where $v \neq v_i$ is done by calling *augment_cq_set()* W times with $S(v_i,i,i)$, $d(v,v_i)$ and each $w \in \{1..W\}$ as arguments and taking the union of the resulting sets (W is the largest allowable wire width as a multiple of the base wire width). We then apply *prune_cq_set()* to the resulting set.

```
$S \leftarrow \emptyset$
for $k = i..j-1$
    $S \leftarrow S \cup join\_cq\_sets(S(v,i,k), S(v,k+1,j))$
    $S \leftarrow prune\_cq\_set(S)$
$S_b(v,i,j) \leftarrow S$
```

Figure 7: *Computation of $S_b(v,i,j)$ in P-Tree$_{AT}$.*

Computation of $S_b(v,i,j)$ is illustrated in Figure 7. The algorithm visits the possible partition points k in the subsequence and takes the union of the resulting cq-sets and prunes the result per Property 4.1.

As in the min-area case we compute $S(v,i,j)$ for all $v \in V$ and fixed i,j in a four phase process. During the computation we partition the intermediate solution sets into W disjoint subsets to ensure that wire segments between Steiner points are of uniform width.[1] For instance, $LS(v,w)$ is a

[1]The algorithm can be structured to allow multiple widths per wire-segment. However, this is at the expense of higher computational complexity – such fine-tuning might be more appropriate at a later stage on a fixed topology.

```
/* Phase 1 */
$LS(v_{\{r,n\}}, w) \leftarrow S_b(v_{\{r,n\}})$    $\forall r \in \{1..n\}, w \in \{1..W\}$
for $r = 1$ to $n$
  for $c = 2$ to $n$
    $d \leftarrow d(v_{\{r,c-1\}}, v_{\{r,c\}})$
    for $w = 1$ to $W$
      $LS(v_{\{r,c\}}, w) \leftarrow S_b(v_{\{r,c\}}) \bigcup$
          $augment\_cq\_set(LS(v_{\{r,c-1\}}, w), d, w)$
      $LS(v_{\{r,c\}}, w) \leftarrow prune\_cq\_set(LS(v_{\{r,c\}}, w))$
```

Figure 8: *Computation of LS(v) in P-Tree$_{AT}$*

cq-set rooted at v where the wire to a branching point v' must have width w and v' is constrained to be at v or to v's left in the same row. In the fourth stage of the algorithm we collect in $S(v, i, j)$ all solutions regardless of wire width.

For space considerations Figure 8 gives pseudo-code for only the first phase – i.e., the computation of $LS(v, w)$. The other phases follow analogously as in the P-Tree$_A$ algorithm. Notice that instead of taking scalar minimums, we now apply our cq-set primitives and we now must deal with the parameterization of the sets by w.

To obtain the final area/delay tradeoff set S_{final} we augment each pair $(c, q) \in S(v_d, 1, n-1)$ by considering the resistance of the driver (much in the same way as we did with an augmenting wire) as follows.

$$S_{final} \leftarrow \{(c, q - r_d \cdot c) | (c, q) \in S(v_d, 1, n-1)\}$$
$$S_{final} \leftarrow prune\_cq\_set(S_{final})$$

Finally, we can show the optimality of the solutions derived by the P-Tree$_{AT}$ algorithm analogously to Lemma 1.

Lemma 2 *Consider $(c, q) \in S_{\text{final}}$ computed by the P-Tree$_{AT}$ algorithm. There exists no Rectilinear Steiner tree induced by sink permutation π with Steiner points on the Hanan Grid which yields required-time $q' \geq q$ and capacitance $c' < c$ (recall that load is proportional to area).*

The complexity of P-Tree$_{AT}$ depends on the size of the cq-sets it manipulates is reflected in the next lemma.

Lemma 3 *Letting m be a bound on the maximum size of any cq-set, P-Tree$_{AT}$ executes in time $O(\max(Wn^4m, n^5m)) = O(n^5m)$ where W is the number of allowed wire-widths (W assumed to be $O(n)$).*

It can be argued that m is polynomially bounded if the individual capacitive values are polynomially bounded integers or can be mapped to such with sufficient precision. This yields a *pseudo-polynomial* complexity [6]. However, this does not give much insight into the running time and practicality of the algorithm. As will be seen in the next section, the algorithm is quite practical for sized nets which make up the vast majority of nets in typical VLSI systems.

However, depending on net-size and W, cq-sets can grow to be quite large. To combat this we have implemented a simple heuristic to limit cq-sets to be no larger than a pre-specified size. Given a maximum allowable cq-set size of m and a cq-set S with $|S| > m$ we first select $m/2$ entries in S uniformly by their ordering (always including the first entry) and select the remaining $m/2$ by their "marginal benefit"; i.e., the marginal benefit of i'th entry $(c_i, q_i) \in S$ is $(q_i - q_{i-1})/(c_i - c_{i-1})$. This strategy has proved effective in improving run-time while maintaining high solution quality.

Before giving experimental results, we would like to point out that the P-Tree$_A$ algorithm is in a sense "performance oriented". By using required-time as a tie-breaker, we can find the maximum performance tree among the min-area trees at no extra cost. It is typically the case that there are many distinct min-area topologies induced by a permutation and thus we propose that this is a useful property.

5 Experiments

Our main experimental results appear in Table 1. Technology parameters are the same as those used in previous works such as [8] and [1]. For each net-size and technology pair we routed 25 nets (with terminal positions generated randomly and uniformly). In cases with wire sizing, we compared P-Tree with the dynamically wire sized variant SERT and the statically wire-sized variant of A-Tree described in [8]. Since the algorithms we compare with are designed to minimize maximum source to sink delay, we use that as our timing metric (but we note that the ability of P-Tree to deal with varying required times is one of its main advantages).

In the first set of experiments, we ran SERT and A-Tree (and their wire-sized variants) on each net and recorded the resulting max delay and area; we then ran P-Tree$_{AT}$ on those same nets and recorded the min-area solution which was able to match the max delay of SERT; we did the same versus A-Tree. The figures in the tables denote the average ratio of the area of the corresponding solution to the area of the competing P-Tree solution. For instance, for 12-pin nets in MCM technology without wire sizing, the SERT solutions used 66% more area on average than the P-Tree solution.[2]

The second set of experiments examine another way of assessing area/delay tradeoff: the AT metric – i.e., we take the product of a topology's area and its max delay. We choose the solution produced by P-Tree which minimized the AT metric and report the ratio of the AT value of the other algorithms to that of P-Tree. Again we see significant improvements.

Figure 9 shows an area/delay tradeoff curve from the P-Tree$_{AT}$ algorithm when run on a 12 pin MCM net with wire sizing. We also show the wire-sized SERT and A-Tree solutions. It is interesting to see that the solution produced by SERT lies near the flat portion of the P-Tree$_{AT}$ curve while the P-Tree$_{AT}$ solution with the same delay is near the steep portion of the curve. The same phenomenon holds to a lesser degree vs. A-Tree.

Finally, we mention that more extensive experimental results appear in [14]. In particular, [14] also reports min-area and min-delay results and gives data on the improvement afforded by allowing Steiner points off of the Hanan grid.

Run-Times: Finally, we give an idea of the run-time complexity of our algorithms in practice. We implemented the

[2] In two of the 12-pin $1.2\mu m$ trials and three of the $0.5\mu m$ trials, the min-delay P-Tree$_{AT}$ solution was beaten by A-Tree. In these cases the difference in delay was less than 3% percent on average and never greater than 7%; area overhead was comparable to that in the table. We suspect that this is an artifact of the heuristic for limiting cq-sets used in these cases. The figures in the table reflect the nets for which P-Tree$_{AT}$ equaled or bettered A-Tree skewing the results in favor of P-Tree. However, there is no such skew under the AT metric.

Average Area Overhead vs. P-Tree							
		MCM		$1.2\mu m$		$0.5\mu m$	
	$\|N\|$	S	A	S	A	S	A
w/o wire sizing	6	1.67	1.06	1.35	1.12	1.38	1.08
	9	1.41	1.06	1.61	1.16	1.53	1.08
	12	1.66	1.08	1.63	1.17	1.66	1.16
with wire sizing	6	1.29	1.12	1.46	1.17	1.64	1.17
	9	1.30	1.11	1.57	1.19	1.70	1.18
	12	1.36	1.05	1.57	1.19	1.73	1.20

Comparision under AT metric							
		MCM		$1.2\mu m$		$0.5\mu m$	
	$\|N\|$	S	A	S	A	S	A
w/o wire sizing	6	1.41	1.13	1.54	1.17	1.57	1.12
	9	1.70	1.17	2.01	1.23	1.75	1.16
	12	1.91	1.21	2.01	1.24	1.95	1.23
with wire sizing	6	1.53	1.37	1.78	1.26	1.86	1.29
	9	1.50	1.28	1.93	1.29	1.87	1.26
	12	1.58	1.21	1.84	1.28	1.94	1.23

Table 1: *Comparision under two metrics. Values are relative to P-Tree solutions. S=SERT and A=A-Tree.*

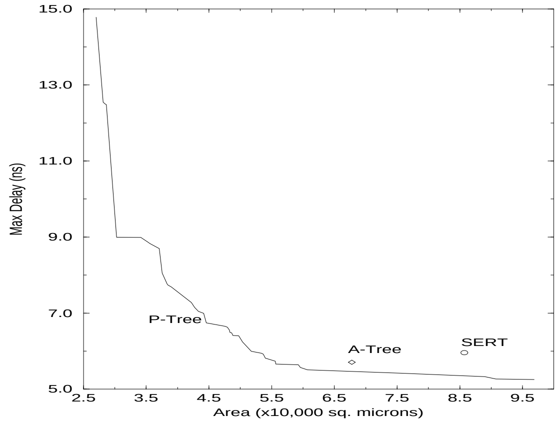


Figure 9: *Area vs. Delay for a 12 pin MCM net (with wire-sizing)*

P-Tree algorithms in C on a Sun SPARC 20 workstation.

Representative run-times for P-Tree$_A$ range from 0.05 cpu seconds for $n = 6$ to 1.8 cpu seconds for $n = 15$ and 49 cpu seconds for $n = 30$. In the case of P-Tree$_{AT}$ without wire sizing, run times ranged from 0.1 cpu seconds for $n = 9$, 7.3 cpu seconds for $n = 12$ and 120 cpu seconds for $n = 20$. When running P-Tree$_{AT}$ *with* wire sizing, run-times ranged from 0.8 cpu seconds for $n = 6$ to 70 cpu seconds for $n = 12$ and 150 cpu seconds for $n = 15$.

While our algorithms have higher computational complexity than the other algorithms evaluated in this paper, we suggest that they are quite practical for the vast majority of net-sizes typically seen in VLSI systems (particularly when the distribution of those net-sizes is taken into account). In addition, we have shown that the payoff for this additional computational complexity in terms of reduced area and improved timing can be very high. Nevertheless, it is the topic of on-going research to derive comparable results in terms of delay and area in less cpu time and for larger nets.

6 Conclusions/Comments

We have proposed the P-Tree$_A$ and P-Tree$_{AT}$ algorithms for finding high performance, low area Rectilinear Steiner Trees. The algorithms find optimal solutions in the rich solution space of routing topologies induced by a permutation on the sinks of the net. We have sketched a technique for finding high-quality sink permutations. Experimental results versus previously proposed approaches are promising in terms of both delay and routing area.

We would like to thank Professor Gabriel Robins of the University of Virginia and Todd Hodes of UC Berkeley for providing implementations of the One Steiner, SERT and A-Tree algorithms. We also thank Rebecca Malley for help in tabulating the experimental results and Robert Carragher for help in proof reading this manuscript.

References

[1] K. D. Boese, A. B. Kahng, G. Robins, "High-Performance Routing Trees With Identified Critical Sinks," *Proc. ACM/IEEE Design Automation Conf.,* 1993, pp. 182-187.

[2] K. D. Boese, A. B. Kahng, B. A. McCoy, G. Robins, "Fidelity and Near-Optimality of Elmore-Based Routing Constructions," *Proc. Proc. IEEE Intl. Conf. Computer-Aided Design*, 1993.

[3] J.J. Cong, K.S. Leung, "Optimal Wiresizing Under Elmore Delay Model," *IEEE Trans. on CAD,* v. 14 no. 3 (1995) pp. 321-336.

[4] J.J. Cong, K.S. Leung, D. Zhou, "Performance-driven interconnect design based on distributed RC delay model," *Proc. ACM/IEEE Design Automation Conf.,* 1993 pp. 606-611.

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

[6] M. R. Garey, D. S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness," *W. H. Freeman,* San Francisco (1979).

[7] M. Hanan, "On Steiner's Problem With Rectilinear Distance," *SIAM J. Applied Math.,* 14 (1966), pp. 255-265.

[8] T. D. Hodes, B. A. McCoy, G. Robins, "Dynamically-Wiresized Elmore-Based Routing Constructions," *Proc. IEEE Intl. Symp. Circuits and Systems*, 1994.

[9] F. K. Hwang, D. S. Richards, P. Winter, "The Steiner Tree Problem," Elsevier Science Publishers, (1992), pp. 213-214.

[10] F.K. Hwang, "On Steiner Minimal Trees with Rectilinear Distance," *SIAM J. Applied Math.* 30 (1976), pp. 104-114.

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

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

[13] J. Lillis, C. K. Cheng, T. T. Lin, "Optimal Wire Sizing for Low Power and a Generalized Delay Model," *Proc. IEEE Intl. Conf. Computer-Aided Design*, 1995.

[14] J. Lillis, C. K. Cheng, T. T. Lin, C.-Y. Ho, "New Techniques for Performance Driven Routing with Explicit Area/Delay Tradeoff and Simultaneous Wire Sizing," Technical Report #CS96-469, CSE Dept., UCSD.

[15] S.S. Sapatnekar, "RC Interconnect Optimization under the Elmore Delay Model," *Proc. ACM/IEEE Design Automation Conf.,* 1994, pp. 387-391.

[16] A. Vittal, M. Marek-Sadowska, "Minimal Delay Interconnect Design Using Alphabetic Trees," *Proc. ACM/IEEE Design Automation Conf.,* 1994, pp. 392-396.

[17] L.P.P.P van Ginneken, "Buffer Placement in Distributed RC-tree Networks for Minimal Elmore Delay," *Proc. International Symposium on Circuits and Systems,* 1990, pp 865-868.