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

PERFORMANCE-DRIVEN INTERCONNECT DESIGN BASED ON DISTRIBUTED RC DELAY MODEL

J. Cong
K.-S. Leung
D. Zhou

October 1992
CSD-920043

Performance-Driven Interconnect Design Based on Distributed RC Delay Model

Jason Cong and Kwok-Shing Leung
Department of Computer Science
University of California, Los Angeles
Los Angeles, CA 90024

Dian Zhou
Department of Electrical Engineering
University of North Carolina
Charlotte, NC 28213

Abstract

Most previous works on the interconnect design are based on a simple model which assumes that interconnection delay of a net is proportional to the total wirelength of the net. Such delay model becomes less and less realistic as the fabrication technology reaches submicron device dimensions and gigahertz frequency. In this paper, we study the interconnect design problem under a distributed RC model, and present efficient solutions to interconnect topology design and wiresizing problems for performance optimization. We study the impact of technology factors on the interconnect designs and present general formulations of the interconnect topology design and wiresizing problems. We show that interconnect topology optimization can be achieved by computing optimal generalized rectilinear Steiner arborescences and we present an efficient algorithm which yields optimal or near-optimal solutions. We reveal several important properties of optimal wire width assignments and present a polynomial time optimal wiresizing algorithm. Extensive experimental results indicate that our approach significantly outperforms other routing methods for high-performance IC and MCM designs. Our interconnect designs reduce the interconnection delays by up to 66% as compared to those by the best known Steiner tree algorithm.

1 Introduction

As the VLSI fabrication technology reaches submicron device dimension and gigahertz frequency, interconnection delay has become the dominant factor in determining circuit speed [6, 16]. Most previous works on this problem are based on a simplistic linear delay model of wirelength minimization, and many global routing algorithms based on the Steiner tree formulation have been proposed, such as [12, 2]. Although these methods produce good results in terms of wirelength minimization, they cannot achieve performance optimization in the design of high-speed ICs and MCMs. Moreover, uniform wire width has been used for the connections of most signal nets in the conventional routing methods. Very little is known about proper wiresizing for delay optimization. There is a strong need for systematic studies in the interconnect topology design and wiresizing problems for delay optimization in high-performance system designs.

However, limited progress has been reported in the literature for the performance-driven interconnect design problem. In [7], net priorities are determined based on static timing analysis; nets with high priorities are processed earlier using fewer feedthroughs. In [11], a hierarchical approach to timing-driven routing was outlined. In [13], a timing-driven global router based on the A* heuristic search algorithm was proposed in building-block designs. In [3, 4], a timing-driven global router was proposed to minimize both the total wire-

length and the longest path from the source to any sink simultaneously. Although these routers tried to reduce the interconnection delay by optimizing the routing topology, their objective functions were oversimplified due to the use of linear delay model or the lumped RC delay model. Moreover, none of these algorithms study the impact of wiresizing on interconnect delay minimization. Although wiresizing was used by Fisher and Kung [9] in H-tree clock routing, the general wiresizing problem for arbitrary routing topology has not been well studied before.

In this paper, we study the interconnect design problem under a distributed RC delay model developed by Rubinstein, Penfield and Horowitz [15]. Using this model, we develop a routing algorithm based on the efficient construction of rectilinear arborescences. We have also developed a polynomial time optimal wiresizing algorithm which is applicable to arbitrary routing topology.

The remainder of this paper is organized as follows: In Section 2, we present the general formulation of the routing and wiresizing problems based on the distributed RC delay model. In Section 3, we give an efficient routing algorithm for delay minimization based on the construction of rectilinear arborescences. In Section 4, we present an optimal wiresizing algorithm and a set of speed-up techniques. Section 5 shows the comparisons among different approaches to the interconnect design problem based on circuit simulation results. Section 6 concludes this paper with the discussion of future works.

2 Problem Formulation

Traditionally, the minimum Steiner tree has been the preferred interconnect topology because: (1) it uses the minimum wiring area, and (2) it results in the minimum wire capacitance which used to be the dominating factor contributing to the interconnection delay in the conventional technology due to large driver resistance and small wire resistance. However, in the submicron VLSI designs, the driver resistance is smaller and the wire resistance is relatively larger. In this case, the *distributed* nature of a circuit must be considered. In this case, the minimum delay is no longer achieved by an optimal Steiner tree. Figure 1 shows the simulation results of an optimal Steiner tree and an a minimum-cost shortest-path tree. We can see clearly that the latter tree results in a smaller delay, despite the fact it has larger total wirelength.

In this paper, we use a distributed RC delay model developed by Rubinstein, Penfield, and Horowitz [15]. Given a distributed RC circuit, the signal delay at a node is computed as follows:

$$t = \sum_{all\ nodes\ k} R_k \cdot C_k \tag{1}$$

where R_k is the resistance between the source and the node k and C_k is capacitance at the node k.

There are several reasons that we choose this delay model: (i) Although this model is simpler than the commonly used Elmore delay model [8] for distributed RC circuits, it still captures the distributed nature of the circuit; (ii) It gives an accurate upper bound of signal delay at every node and correlates very well with the Elmore delay [15]; (iii) It is much easier to use for interconnect design optimization since it gives a uniform upper bound of the delay at every node.

Given a routing tree T implementing a net which consists of a source and a set of sinks, we shall use Equation 1 to compute the signal delay $t(T)$ at any node in tree T. In order to model a routing tree as a distributed RC tree accurately, a grid structure is superimposed on the routing plane, and each wire segment in the routing tree is divided into a sequence of wires of unit length as shown in Figure 2. (Grid points are unit

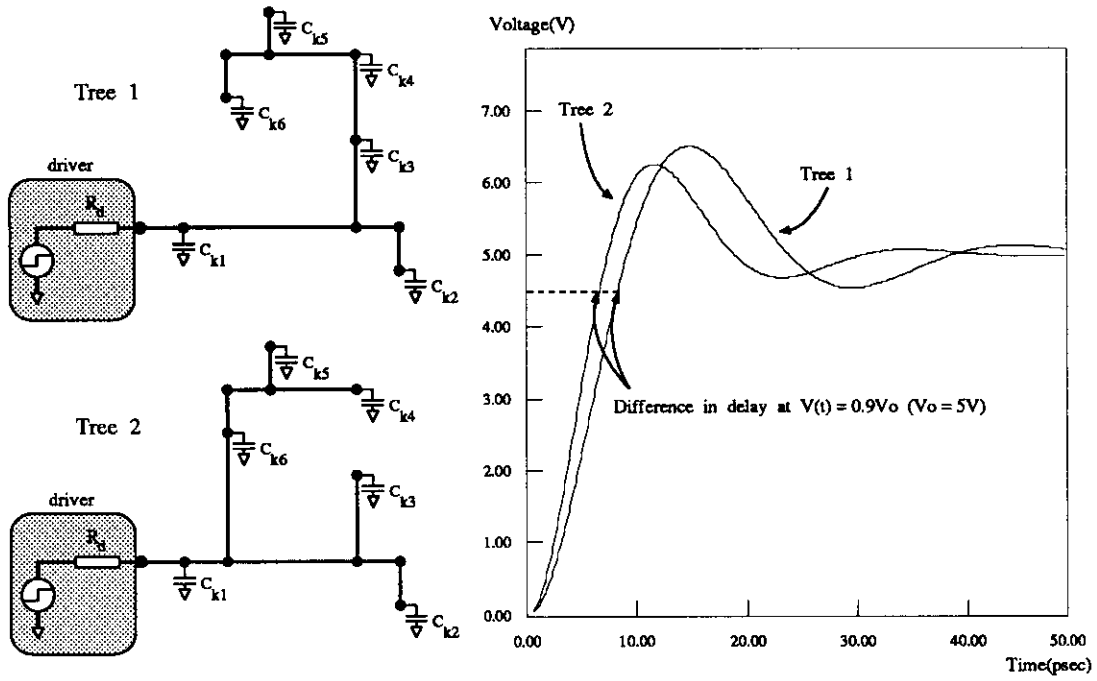


Figure 1: Circuit responses of two different interconnect topologies. Tree 1 is an optimal Steiner tree. Tree 2 is optimized for delay minimization.

length apart.) When the wire width is fixed, both the wire resistance and the wire capacitance are proportional to the wirelength. (We shall discuss the case of variable wire widths later.) Assume that a unit-grid-length wire has wire resistance R_0 and wire capacitance C_0, then according to Equation 1, an upper bound of the delay of a routing tree is:

$$\begin{aligned} t(T) &= \sum_{k \epsilon T} (R_d + R_0 \cdot pl_k(T)) \cdot (C_0 + C_k) \\ &= \sum_{k \epsilon T} R_d \cdot C_0 + \sum_{k \epsilon T} R_0 \cdot pl_k(T) \cdot C_k + \sum_{k \epsilon T} R_0 \cdot pl_k(T) \cdot C_0 + \sum_{k \epsilon T} R_d \cdot C_k \end{aligned} \quad (2)$$

where R_d is the driver resistance, $pl_k(T)$ is the wirelength from the source to node k in the routing tree T, and C_k is the *extra* capacitance (besides the wire capacitance) at node k in T. Notice that the set of nodes includes all grid points in the routing tree T, not just sinks and branching nodes. If node k is a sink, then C_k is the loading capacitance at the node. If there is a via at node k, it can also be formulated by adding some extra capacitance at the node [18], which reflect the distributed nature of interconnection delay. For simplicity, we assume that C_k is non-zero only when node k is a sink, but wire capacitance C_0 presents at every node. Note that the summation in Equation 2 is over all nodes in the routing tree T, not just sinks. Based on this delay model, we shall formulate a number of performance optimization problems in the subsequent subsections.

2.1 The Optimal Interconnect Topology Design

We can rewrite Equation 2 into the following form:

$$t(T) = t_1(T) + t_2(T) + t_3(T) + t_4(T) \quad (3)$$

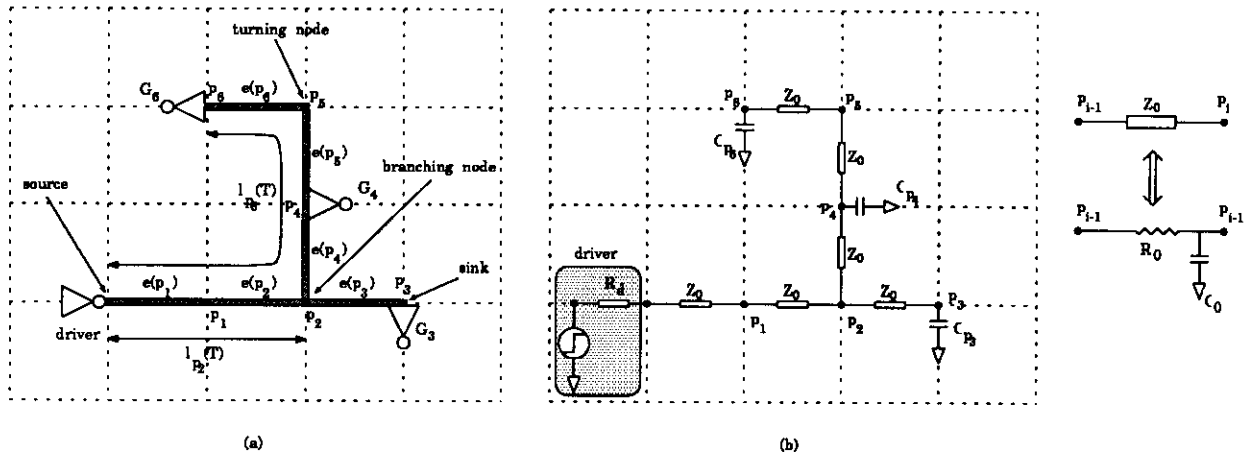


Figure 2: A grid structure for RC distributed model. (a) The layout of an interconnect T with 3 sinks (G_3, G_4, and G_6). (b) The corresponding distributed RC model of T. Each edge in T connecting two adjacent nodes is modeled as a RC element containing a resistance R_0 and a capacitance C_0. C_{p3}, C_{p4}, and C_{p6} are the extra loading capacitance at the sinks respectively.

where

$$t_1(T) = R_d \cdot C_0 \cdot length(T) \tag{4}$$

$$t_2(T) = R_0 \cdot \sum_{all\ sinks\ k} C_k \cdot pl_k(T) \tag{5}$$

$$t_3(T) = R_0 \cdot C_0 \cdot \sum_{k \epsilon T} pl_k(T) \tag{6}$$

$$t_4(T) = R_d \cdot \sum_{all\ sinks\ k} C_k \tag{7}$$

In Equations 4-7, $length(T)$ is the total wirelength of the routing tree T, and $pl_k(T)$ is the path length from the source to the node k. Since the driver resistance R_d and the total loading capacitance at all sinks, $\sum_{all\ sinks\ k} C_k$, are fixed for a given net, $t_4(T)$ is a constant. In order to minimize $t(T)$, we only need to minimize $t_1(T) + t_2(T) + t_3(T)$. It is interesting to see the physical meanings of these three terms.

The first term $t_1(T)$ is the product of the driver's output resistance and the total wire capacitance. This term is minimized when $length(T)$ is minimum. In the conventional technology, $t_1(T)$ is the dominating term since the driver's output resistance is much larger than the wire resistance[1] (i.e. $R_d \gg R_0 \cdot length(T)$). This explains partially why the conventional routing methods emphasize the total wirelength minimization. Clearly, minimizing $t_1(T)$ leads to an optimal Steiner tree (OST).

In the summation of the second term $t_2(T)$, each term is a product of the loading capacitance C_k of each sink and the pathlength $pl_k(T)$ from the source to the sink k in T. Since the loading capacitance of a sink in a given design is fixed, $t_2(T)$ is minimized when all the paths from the source to sinks are minimum in length, which results in a shortest path tree rooted at the source. This shows that when all (or some) loading capacitance are very large, we need to select the shortest paths to connect the source to these sinks. Therefore, minimizing $t_2(T)$ leads to a shortest path tree.

[1] In the 2 μm CMOS technology, the typical values of driver resistance and unit wire capacitance are: $R_d = 1k\Omega - 4k\Omega, R_0 = 0.03\Omega/\mu m$.

The third term is more interesting. It is proportional to the summation of the pathlengths from the source to all nodes (not just sinks) in T. Intuitively, if there is a long source-to-leaf path P in T, $t_3(T)$ will be very large since it contains a term $\sum_{k \epsilon P} pl_k(T)$ which is roughly proportional to the square of the length of P. Therefore, we prefer a routing tree of short paths in order to keep $t_3(T)$ small. This, in fact, justifies the work by Cong, et al. [3] in which the radius (i.e. the longest source-sink path) of a routing tree is kept under certain bound in the layout design. On the other hand, if $length(T)$ is large, the number of nodes in T is large and it may increase $t_3(T)$ as well. Therefore minimizing $t_3(T)$ leads to a routing tree which is "in-between" a shortest path tree and an optimal Steiner Tree. We shall call a tree optimal under the cost function $t_3(T)$ a Quadratic Minimum Steiner Tree ($QMST$).

Notice that the relative importance of these terms is determined by the ratio $\frac{R_d}{R_0}$, which we call the *Resistance ratio*. Because the resistance ratio was very large in the previous technology, conventional routing techniques focused on total wirelength minimization and used minimum wire width for all segments in order to minimize $t_1(T)$. However, as the technology advances to smaller device dimensions, according to the CMOS scaling rule [1], driver resistance decreases while wire resistance increases, which leads to a significant reduction of the resistance ratio. In this case, $t_2(T)$ and $t_3(T)$ can no longer be ignored in the design of new generation VLSI circuits. Although Equations 4-7 are obtained from a distributed RC delay model, the impact of resistance ratio on interconnection design is true for distributed RLC delay model as well. Zhou, Preparata and Kang [17] studied the interconnection delay in a single high-speed transmission line and have observed similar results.

In general, the minimum delay routing tree ($MDRT$) problem can be formulated as follows:

> *Given a signal net $N = \{N_0, N_1, \cdots, N_{n-1}\}$ where N_0 is the source and $N - \{N_0\}$ is the set of sinks, and three nonnegative constants α, β, and γ, find a rectilinear routing tree T of the net N such that*
>
> $$\alpha \cdot length(T) \; + \; \beta \cdot \sum_{all\ sinks\ k \epsilon N} pl_k(T) \; + \; \gamma \cdot \sum_{all\ nodes\ k \epsilon T} pl_k(T) \tag{8}$$
>
> *is minimized, where $length(T)$ is the total wirelength of the tree T and $pl_k(T)$ is the pathlength from the source N_0 to the node k in T.*

The $MDRT$ problem is NP-hard in general since the problem degenerates into the classic Steiner tree problem when both β and γ equal zero. Although the SPT problem alone can be solved optimally in polynomial time, the combination of the three different (possibly conflicting) cost functions makes the problem very difficult. Figure 3 shows an example in which optimizations of $t_1(T)$, $t_2(T)$, and $t_3(T)$ independently leads to three distinct trees. In Section 3, we shall study the problem of constructing $MDRT$ trees in detail.

2.2 The Optimal Wiresizing

In the preceding subsection, we assume that wires in a routing tree have uniform width, which is widely used in conventional layout designs. However, our study shows that for the new generation of VLSI technology, proper sizing of the wires can lead to substantial reduction in signal delay. For example, Figure 4 shows two routing trees of a three-terminal net. Both trees have the same routing topology (a T-tree with the source at the end of the T-stem). But the first tree uses the uniform wire width, while in the second tree the width of the T-stem is twice that of the two T-branches. As expected, simulation results show that the second tree has a smaller

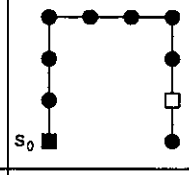
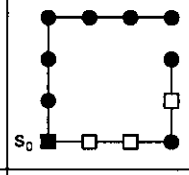
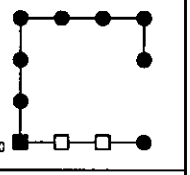
Tree Topology: ■ source ● sink □ other node			
Optimal Tree Topology:	OST	SPT	QMST
Total wirelength t_1(T):	9 (optimal)	11	10
Sum Of Source-to-Sink pathlengths t_2(T):	37	29 (optimal)	31
Sum Of Source-to-grid-point pathlengths t_3(T):	45	36	34 (optimal)

Figure 3: An example in which the OST, SPT, and $QMST$ all differ from each other.

signal delay than the first one. In this subsection, we shall formalize the general wiresizing problem for delay optimization.

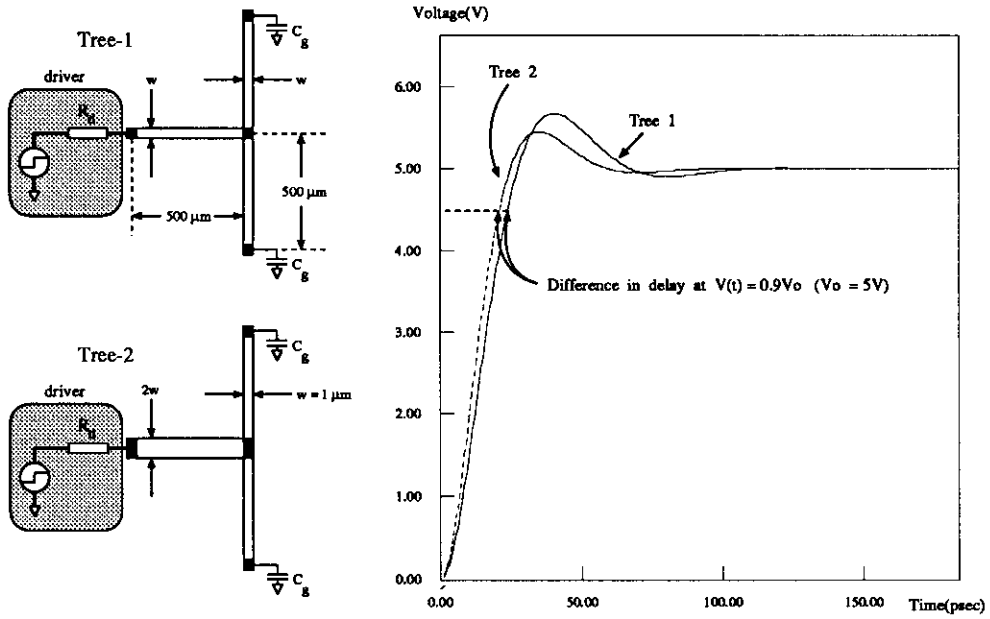


Figure 4: Circuit responses of two trees with the same topologies but different wire widths. Tree 1 has uniform wire width. Tree 2 has larger wire width for the "stem".

Assume that we have a set of discrete wire widths, $\{W_1, W_2, \cdots, W_r\}$, to choose for each wire segment. Let w_k be the width of the incoming grid edge $e(k)$ at node k. If the wire width at node k is w_k, according to

Equation 1, the upper bound of the delay of a routing tree becomes:

$$\begin{aligned} t(T) &= \sum_{k \epsilon T} (R_d + R_0 \cdot \sum_{i \epsilon P_k(T)} \frac{1}{w_i}) \cdot (C_0 \cdot w_k + C_k) \\ &= \sum_{k \epsilon T} R_d \cdot C_0 \cdot w_k + \sum_{k \epsilon T} R_0 \sum_{i \epsilon P_k(T)} \frac{1}{w_i} \cdot C_k + \sum_{k \epsilon T} R_0 \cdot \sum_{i \epsilon P_k(T)} \frac{1}{w_i} \cdot C_0 \cdot w_k + \sum_{k \epsilon T} R_d \cdot C_k \\ &= t_1(T) + t_2(T) + t_3(T) + t_4(T) \end{aligned} \tag{9}$$

where

$$t_1(T) = R_d \cdot C_0 \cdot area(T) \tag{10}$$

$$t_2(T) = R_0 \cdot \sum_{all\ sinks\ k} C_k \cdot \sum_{i \epsilon P_k(T)} \frac{1}{w_i} \tag{11}$$

$$t_3(T) = R_0 \cdot C_0 \cdot \sum_{k \epsilon T} \sum_{i \epsilon P_k(T)} \frac{w_k}{w_i} \tag{12}$$

$$t_4(T) = R_d \cdot \sum_{all\ sinks\ k} C_k \tag{13}$$

In these equations, $area(T)$ is the total wiring area of T, and $P_k(T)$ is the path from the source to the node k in the routing tree T. The meanings of the remaining terms are the same as in the previous subsection. Again, we can show that $t_4(T)$ is a constant, and $t_1(T)$, $t_2(T)$, and $t_3(T)$ have similar physical meanings as in the previous subsection.

The wiresizing problem can be formulated as follows:

> *Given a set of wire segments S implementing a routing tree T and a set of possible widths $W = \{W_1, W_2, \cdots, W_r\}, (W_i < W_{i+1}\ 1 \leq i \leq r-1)$, the wiresizing problem is to find a wire width assignment $f : S \rightarrow W$ such that the delay $t(T)$ defined in Equation 9 is minimized.*

In the formulation, a segment is a straight wire connecting two adjacent non-trivial nodes. A node is *non-trivial* if it is a turning node, a branching node, a sink, or the source. Intermediate grid points on a wire segment are called *trivial* nodes. We assume that each wire segment (rather than each edge) has a set of discrete choices of wire widths since this resembles more closely to the realistic design style and reflects the actual technological constraint where arbitrary width variation *within a segment* is usually undesirable. Nevertheless, this segment-based formulation can easily be generalized to handle the case where variable wire width is allowed within a segment by introducing artificial non-trivial nodes along each segment.

Ideally, we want to determine the topology of the routing tree and perform wire width assignment to the segments in the tree at the same time. However, the complexity involved in this problem is very high and it is unlikely to be solved efficiently. Instead, we adopt the approach of first determining the interconnect topology and then optimizing wire widths in the given routing tree. Solutions to the wiresizing problem will be discussed in Section 4.

2.3 Definitions and Basic Concepts

For future discussion, we use p_x, p_y to refer to the x-coordinate and its y-coordinate of a node p, respectively. We use $dist_x(p,q)$, $dist_y(p,q)$, and $dist(p,q)$ to denote the horizontal distance, the vertical distance, and the rectilinear distance between nodes p and q respectively. Given two nodes p and p', we use $P_T(p',p)$ to denote

the path connecting p and p' in the tree T. Moreover, we define $parent(p)$ as the non-trivial node other than a turning node immediate preceding p along the path $P_T(N_0, p)$.

It is helpful to look at the $QMST$ problem from another point of view for better understanding how tree topology is related to the cost $\sum_{all\ nodes\ k} pl_k(T)$. If we assign $pl_k(T)$ as the cost of the edge $e_T(k', k)$ where k' is the node immediate preceding k in $P_T(k)$, then the total $QMST$ cost can be obtained by summing all the edge costs of the tree T. This is illustrated in Figure 5. We can, therefore, consider the $QMST$ problem as a generalized version of the OST problem where the cost of each edge is a function of its location and the tree topology, and our goal becomes finding a tree of minimum total cost. According to this definition, the difference between the OST and $QMST$ problems is merely the cost function defined on each edge in the routing tree. We use $cost(T)$ to denote the cost of the tree T under the OST cost function, and $cost_{QMST}(T)$ to denote the cost of the tree T under the $QMST$ cost function. Also, we use $cost(P)$ ($cost_{QMST}(P)$) to denote the sum of edge costs of path P under the OST cost ($QMST$ cost).

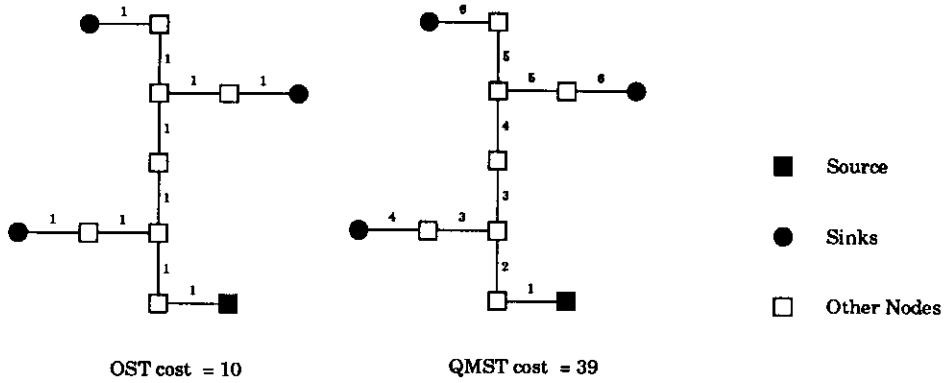


Figure 5: The $QMST$ problem can be transformed into the variable-cost OST problem. The edge cost of the tree to the left is according to the OST cost function, and the edge cost of the tree to the right hand side is defined according to the $QMST$ cost function.

3 Interconnect Topology Design

3.1 The Basic Approach

Since the general $MDRT$ problem is NP-complete and there is no definite correlation between the three terms $t_1(T)$, $t_2(T)$, and $t_3(T)$ for a general interconnect topology, we shall focus our attention to a special type of routing topology, called the A-tree topology, which is defined as follows.

Definition 1 *A rectilinear Steiner tree T is a called an A-tree if every path connecting the source N_0 and any node p on the tree is a shortest path.*

Notice that A-trees are generalization of the rectilinear Steiner arborescences studied in [14]. Given a set N of n nodes lying in the first quadrant of E^2 (the Euclidean Plane), including a node at the origin, a *rectilinear Steiner arborescence* is a directed tree rooted at the origin and contains all nodes in N, composed of horizontal and vertical arcs oriented only from left to right or from bottom to top. However, an A-tree allows the nodes in N to lie anywhere in E^2 as long as paths are always directed away from the origin.

There are several reasons that we are interested in A-trees. First, any A-tree is always a SPT. Therefore, the $t_2(T)$ term is always minimum. Moreover, we shall show later (in Theorem 2) that an optimal Steiner A-tree (OSA) is also a quadratic minimum Steiner A-tree ($QMSA$) in most cases. Therefore, minimizing $t_1(T)$ is equivalent to minimizing $t_3(T)$. As a result, minimizing total wirelength of an A-tree leads to simultaneous optimization of $t_1(T)$, $t_2(T)$, and $t_3(T)$. Such a harmony would be impossible to achieve for general routing topologies. Furthermore, our experimental results show that we can develop efficient algorithm for computing A-tree with optimal or near-optimal wirelength. Therefore, we restrict the solutions to the $MDRT$ problem to the class of A-trees.

Without loss of generality, we consider only the case when all the sinks are restricted to the first quadrant (i.e. when the A-tree becomes an arborescence). There are several well established results regarding arborescences in the literature. In particular, Rao, et al. [14] presented a number of fundamental properties of the rectilinear Steiner arborescence problem. Besides, most of the discussions on arborescences are also applicable to the general A-trees. Moreover, by determining the best rectilinear Steiner arborescence in each quadrant satisfying specific boundary conditions, and selecting the best topology among all possible combinations of tree topologies considered, we can use an arborescence algorithm to construct general A-trees, and the overall complexity of the new algorithm is $O(n^2)$ times that of the original arborescence algorithm [14].

Hence, we focus on solving the rectilinear Steiner arborescence problem from now on. Unless explicitly specified, we will assume that the source N_0 is located at $(0, 0)$, and all other sinks have nonnegative x- and y-coordinates. We use $cost(T)$ to denote the total wirelength of a tree T. Our goal is to construct a rectilinear Steiner arborescence T such that $cost(T)$ is minimized.

3.2 The A-tree Algorithm

The outline of the A-tree algorithm is as follows: Starting with a forest of n single-node arborescences, we apply a sequence of moves. Each move introduces a path consisting of one or more segments, which either "grows" an existing arborescence, or "combines" two arborescences into a new arborescence. This process is completed when there is only one arborescence left in the forest (see Figure 6).

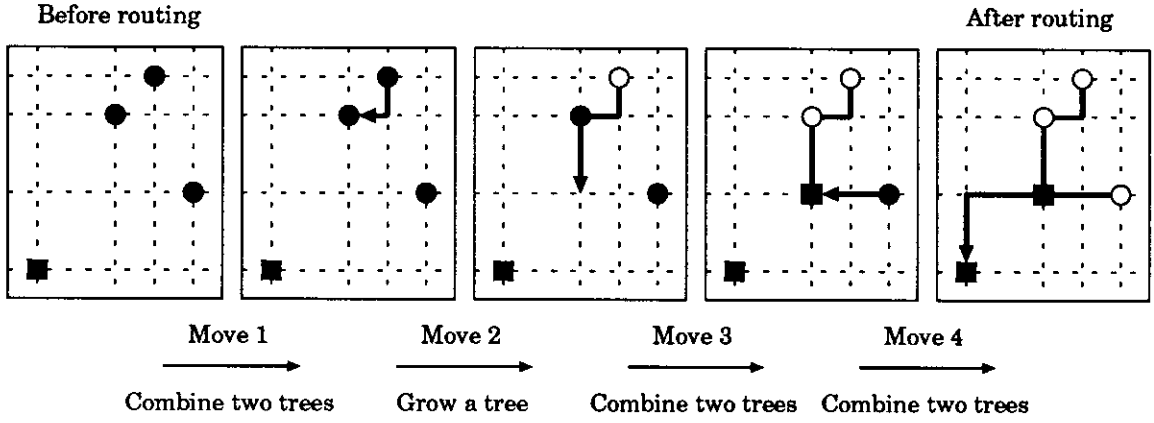


Figure 6: Illustration of how an arborescence is constructed.

The central part of this algorithm is how to choose a good move. After we introduce several notations

describing the topology, we will describe three types of moves that are "optimal".

Definition 2 *Let F_k denote the forest constructed after the k^{th} move by the A-tree algorithm. $ROOT(F_k)$ denotes the set of root nodes in F_k.*

Definition 3 *Given a node p, $NW(p)$ is the region "northwest" of p (i.e. $NW(p) = \{(x, y) | x < p_x, y > p_y\}$).*

$SE(p)$, $SW(p)$, and $NE(p)$ are similarly defined. Also, we use $N(p)$, $E(p)$, $S(p)$, and $W(p)$ to refer to the region to the North, East, South and West of the node p. In other words, for a given node p, its entire neighborhood is partitioned by these eight *regional* sets with respect to p.

Definition 4 *Given two nodes p and q in F_k, p dominates q if $p_x \geq q_x$ and $p_y \geq q_y$. Moreover, $DOM(p, F_k)$ is defined as the set of nodes in F_k which are dominated by p.*

Definition 5 *Given two nodes p and q in F_k such that $q \in NW(p)$, q is said to be blocked from p if there is a node $r \in F_k$ such that $x_r = x_q$ and $y_p \leq y_r < y_q$. Similarly, p is said to be blocked from q if there is a node $r \in F_k$ such that $y_r = y_p$ and $x_q \leq x_r < x_p$. (Note that p is blocked from q does not imply that q is blocked from p, and vice versa.)*

Definition 6 *Given a node p in $ROOT(F_k)$, we use $mx(p, F_k)$ to denote the node in $NW(p) \cap ROOT(F_k)$ that is not blocked from p and has the minimum horizontal distance from p. Moreover, $dx(p, F_k)$ is defined as the horizontal distance between p and $mx(p, F_k)$ (or ∞ if $mx(p, F_k)$ does not exist). Similarly, $my(p, F_k)$ is defined as the node in $SE(p, F_k) \cap ROOT(F_k)$ that is not blocked from p and has the minimum vertical distance from p, and $dy(p, F_k)$ is defined as the vertical distance between p and $my(p, F_k)$ (or ∞ if $my(p, F_k)$ does not exist).*

Definition 7 *Given a node p, we define $MF(p, F_k)$ as the set of nodes in $DOM(p, F_k)$ with the minimum rectilinear distance from p, and $df(p, F_k)$ as the distance between p and any node in $MF(p, F_k)$. Among nodes in $MF(p, F_k)$, we define $mf_{west}(p, F_k)$ as the one with the smallest x-coordinate, and $mf_{south}(p, F_k)$ as the one with the smallest y-coordinate (notice that $mf_{west}(p, F_k) = mf_{south}(p, F_k)$ in some cases).*

The definitions of these terms are illustrated in Figure 7(a)-(d). We now describe three important types of moves, called *safe moves*, used in our algorithm:

1. **Type-1 Safe Move (S1-Move)**

   If $dx(p, F_k) \geq df(p, F_k)$ and $dy(p, F_k) \geq df(p, F_k)$ for a node p in $ROOT(F_k)$, we introduce a path connecting p to $mf_{west}(p, F_k)$[2] (illustrated in Figure 8(a)). After the S1-move, $ROOT(F_{k+1}) = ROOT(F_k) - \{p\}$.

2. **Type-2 Safe Move (S2-Move)**

   If $dx(p, F_k) \geq df(p, F_k)$ and $dy(p, F_k) < df(p, F_k)$ for a node p in $ROOT(F_k)$, we introduce a vertical path of length $\min\{dist_y(mf_{south}(p, F_k), p), dy(p, F_k)\}$ from p southward to p' (illustrated in Figure 8(b)-(c)). After the S2-move, $ROOT(F_{k+1}) = ROOT(F_k) - \{p\} + \{p'\}$.

[2] In fact we can connect p to any node in $MF(p, F_k)$.

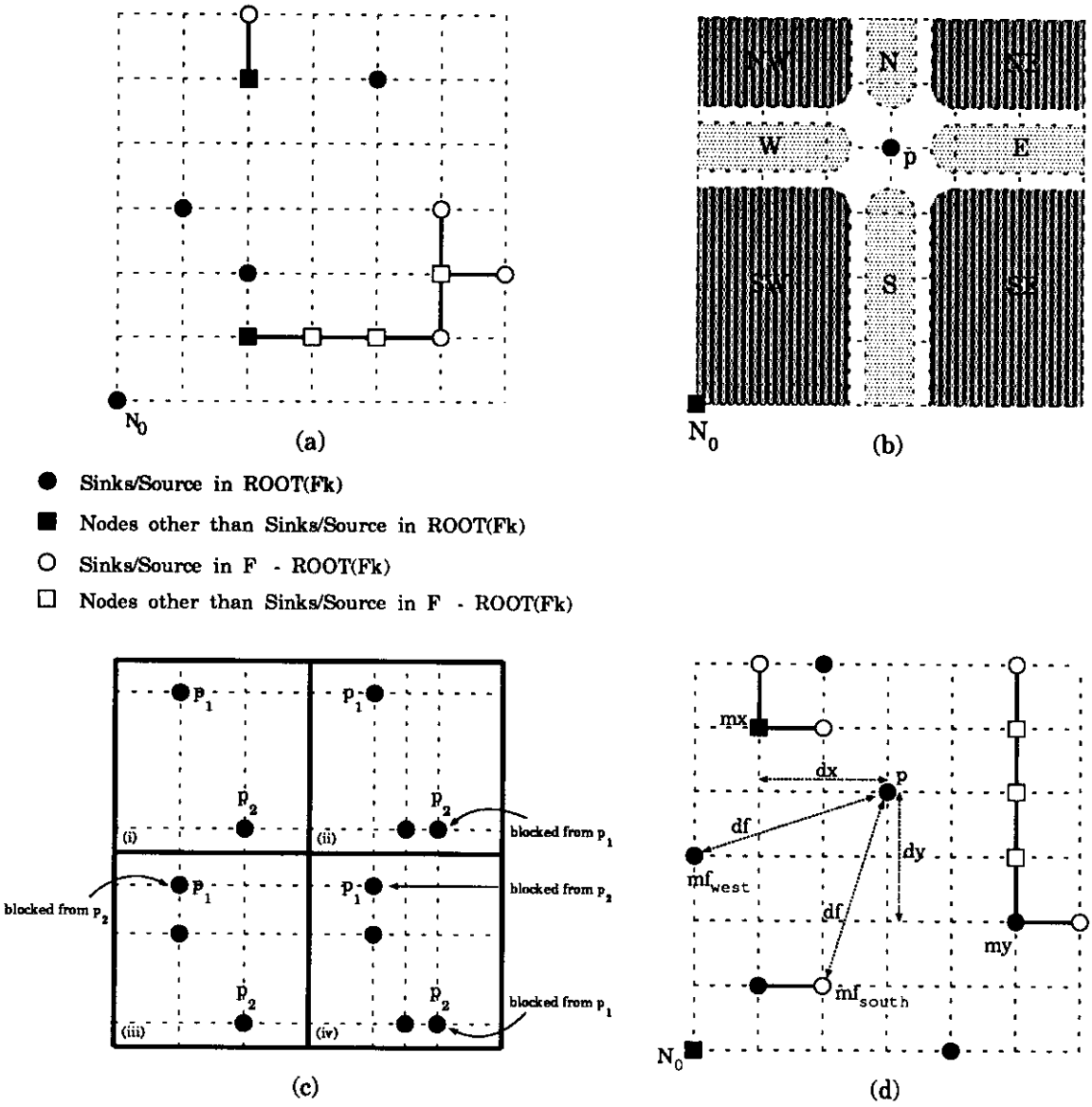


Figure 7: (a) Definitions of $ROOT(F_k)$. (b) Definitions of the regional sets. (c) Illustration of accessibility (whether a node is blocked from another). (d) An example to clarify the definitions of $mx(p, F_k)$, $my(p, F_k)$, and $mf(p, F_k)$. In this example, $dx(p, F_k) = 2$, $dy(p, F_k) = 2$, and $df(p, F_k) = 4$ (rectilinear distance). There are 8 arborescences in the forest F_k, containing 16 nodes.

3. Type-3 Safe Move (S3-Move)

If $dx(p, F_k) < df(p, F_k)$ and $dy(p, F_k) \geq df(p, F_k)$ for a node p in $ROOT(F_k)$, we introduce a horizontal path of length $\min\{dist_x(mf_{west}(p, F_k), p), dx(p, F_k)\}$ from p westward to p' (illustrated in Figure 8(d)-(e)). After the S3-move, $ROOT(F_{k+1}) = ROOT(F_k) - \{p\} + \{p'\}$.

In the next subsection, we will show that these three types of safe moves are optimal under a natural definition of optimality. However, it is possible that after a sequence of safe moves, no safe move exists with respect to the current forest (see Figure 9(a) for an example). In this case, we need to perform a *heuristic move* – a move that may not be optimal. In this paper, we will use the following two heuristic moves suggested by

Rao et al. [14].

1. **Type-1 Heuristic Move (H1-Move)**

   Select a node p in $ROOT(p,k)$ such that $p' = mf_{west}(p, F_k)$ is farthest away from the source and introduce a path connecting p to p'. After the H1-move, $ROOT(F_{k+1}) = ROOT(F_k) - \{p\}$.

2. **Type-2 Heuristic Move (H2-Move)**

   Select two nodes p_1 and p_2 in $ROOT(F_k)$ such that $p' = (\min\{(p_1)_x, (p_2)_x\}, \min\{(p_1)_y, (p_2)_y\})$ is farthest from the origin, and introduce two paths (one vertical and one horizontal) connecting p_1 to p', and p_2 to p' respectively (illustrated in Figure 9(c)). After the H2-move, $ROOT(F_{k+1}) = ROOT(F_k) - \{p_1, p_2\} + \{p'\}$.

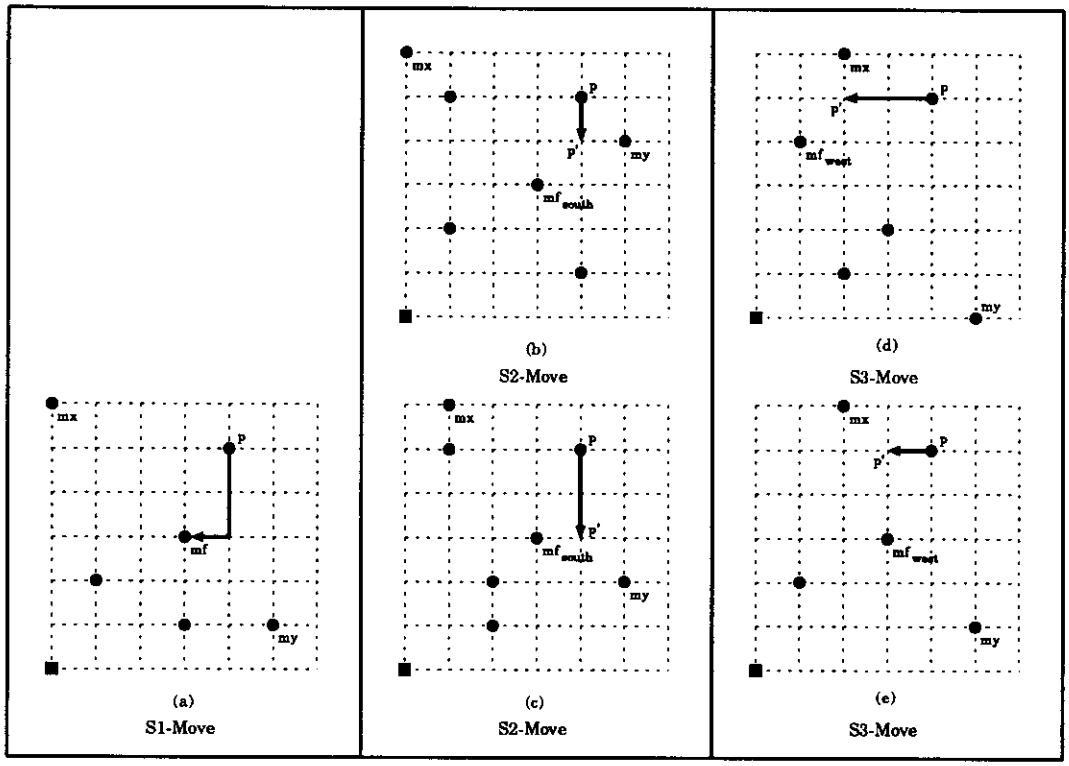


Figure 8: (a) Type-1 safe move. (b) Type-2 safe move: $dy(p, F_k) < dist_y(mf_{south}(p, F_k), p)$. (c) Type-2 safe move: $dy(p, F_k) \geq dist_y(mf_{south}(p, F_k), p)$. (d) Type-3 safe move: $dx(p, F_k) < dist_x(mf_{west}(p, F_k), p)$. (e) Type-3 safe move: $dx(p, F_k) \geq dist_x(mf_{west}(p, F_k), p)$.

In the A-tree algorithm, whether an H1-move or an H2-move is performed depends on which move gives a farther p' from the source N_0. Notice that safe moves are given higher priority than heuristic moves – we perform a heuristic move only if there is *no* safe move, but we stop making heuristic moves as soon as we find *one* safe move. In the subsequent discussions, we shall use $p' \leftarrow p$ to represent an S1-, S2-, S3-, or H1-move which connects p ($\epsilon\ ROOT(F_k)$) to p', and use $(p' \leftarrow p_1) \cup (p' \leftarrow p_2)$ to represent an H2-move which connects both $p1$ and $p2$ ($\epsilon\ ROOT(F_k)$) to p'.

The A-tree algorithm for the construction of rectilinear Steiner arborescence can be summarized in Table 1.

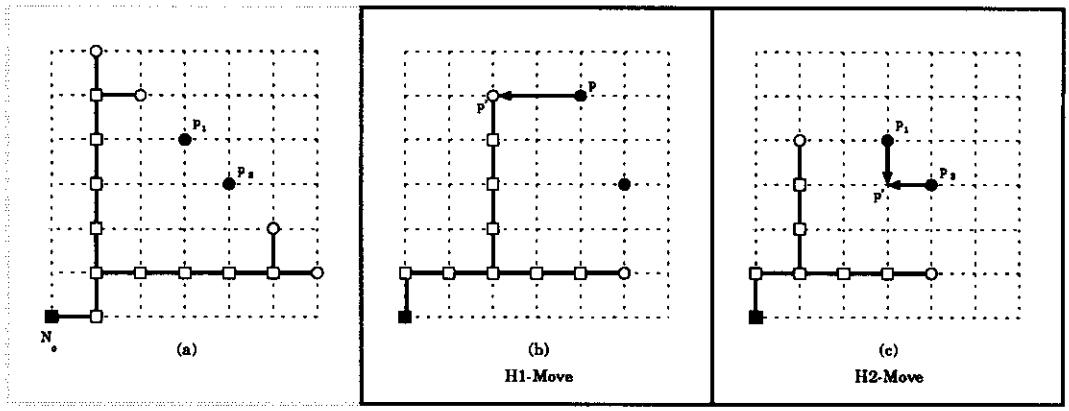


Figure 9: (a) An example where no safe move exists with respect to the forest F_k. In this example, $dx(p_1, F_k) = dy(p_2, F_k) = \infty$, $dy(p_1, F_k) = dx(p_2, F_k) = 1$, and $df(p_1, F_k) = df(p_2, F_k) = 2$. (b) Type-1 heuristic move. (c) Type-2 heuristic move.

A-tree Algorithm

Procedure *ATree*()
  $ROOT(F_k) \leftarrow F_k \leftarrow$ *the source and all sinks*;
  for each *node p in ROOT* **do**
    Compute $dx(p, F_k)$, $dy(p, F_k)$, *and* $df(p, F_k)$;
  end for;
  while $|ROOT(F_k)| > 1$ **do**
    if *there exist safe moves* **do**
      perform one safe move;
    else
      perform one heuristic move;
    end if;
    update $dx(p, F_k)$, $dy(p, F_k)$, $df(p, F_k)$, *and* $ROOT(F_k)$;
  endwhile;
end procedure

Table 1: The A-tree algorithm.

3.3 Optimality Analysis

In this subsection, we shall show that safe moves are *optimal* with respect to the current forest which has been constructed so far. We first give a precise definition of the *optimality* of a move.

Definition 8 *Let F_k be the forest constructed after the k^{th} move by the A-tree algorithm. We define $T^*(F_k)$ as the minimum-cost rectilinear Steiner arborescence containing F_k as a subgraph.*

According to this definition, $T^*(F_0)$ is the optimal rectilinear Steiner arborescence. $T^*(F_1)$ is the optimal Steiner arborescence containing the path generated by the first move, and $T^*(F_M)$ is the arborescence constructed by the A-tree algorithm, where M is the total number of moves generated by the A-tree algorithm. Since $F_k \subset F_{k+1}$, $cost(T^*(F_k)) \leq cost(T^*(F_{k+1}))$.

Definition 9 *If $cost(T^*(F_{k+1})) = cost(T^*(F_k))$, the $k + 1^{th}$ move is called an optimal move.*

For convenience, we shall assume that there exists an imaginary optimal algorithm $A_{opt}(F_k)$ to construct $T^*(F_k)$ from the forest F_k. $A_{opt}(F_k)$ uses the same iterative procedure as the A-tree algorithm, except every move that $A_{opt}(F_k)$ makes is optimal. $A_{opt}(F_k)$ starts with a given forest $F_k^* = F_k$ and iteratively introduces optimal moves (i.e. the $k+1^{th}$ (optimal) move, the $k+2^{th}$ (optimal) move, $\cdots$) until there is only one arborescence left in the forest. The corresponding forests are F_{k+1}^*, F_{k+2}^*, and so on. The sequence of moves generated by $A_{opt}(F_k)$ can be derived from $T^*(F_k)$ according to the following procedure:

Let T^* be the optimal arborescence $T^*(F_k)$ and $LEAF(T^*)$ be the set of leaves in T^*. Let $N^* = ROOT(F_k) \cup N$. We redefine $parent(p)$ for all p in T^* such that all nodes in N^* are considered as candidates of $parent(p)$. This guarantees that every segment in T^* is either entirely contained in F_k or entirely outside F_k. Then, we iteratively remove paths from T^* and derive the sequence of optimal moves generated by $A_{opt}(F_k)$: (i) If there is a node p in $LEAF(T^*)$ whose parent $p^* \in N^*$, we remove the path $P_{T^*}(p^*, p)$ from T^*. If the path is not in F_k, we output it as the optimal move generated by $A_{opt}(F_k)$, and call it an O1-move; (ii) If no parent of the nodes in $LEAF(T^*)$ is in N^*, we select two nodes p_1 and p_2 from $LEAF(T^*)$ with the same parent p^* (which is a Steiner node) and remove the paths $P_{T^*}(p^*, p_1)$ and $P_{T^*}(p^*, p_2)$ from T^*. If both paths are not in F_k, we output both of them together as the optimal move generated by $A_{opt}(F_k)$ and call it an O2-move. If only one path is not in F_k, we output that path as the optimal move generated by $A_{opt}(F_k)$, and call it an O1-move. Otherwise, if both of the paths are in F_k, we do not output any move. After each removal, $LEAF(T^*)$ is updated. The process is completed after all segments in T^* have been removed.

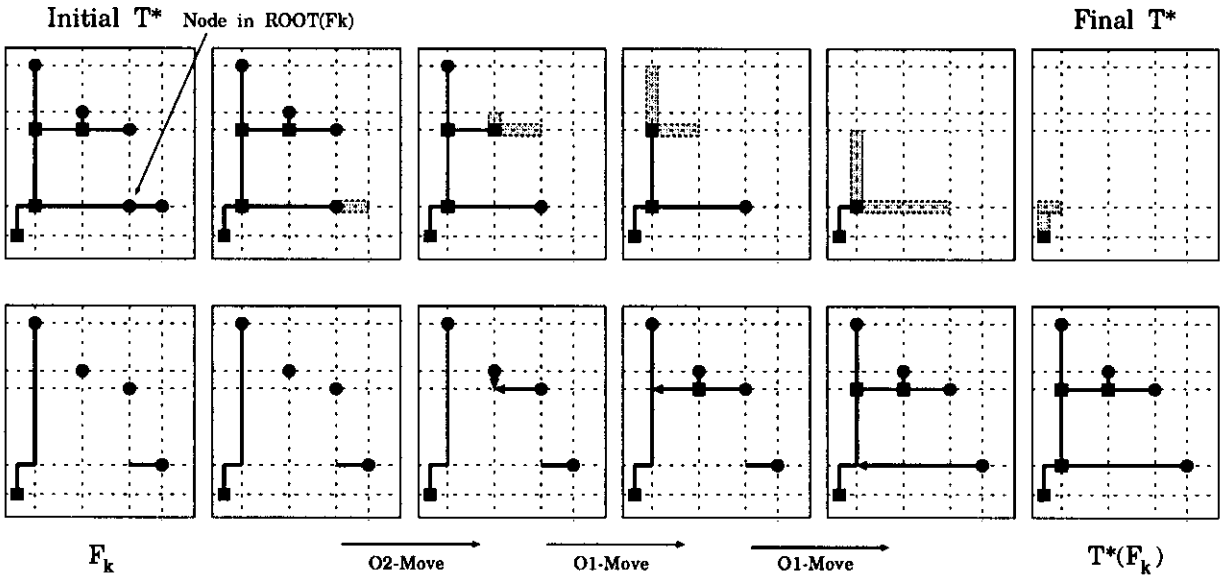


Figure 10: Illustration of how the moves generated by $A_{opt}(F_k)$ are derived from T^*. The upper sequence shows the removals of segments from T^*, and the lower sequence shows the generation of optimal moves. Note that some nodes in $ROOT(F_k)$ are trivial nodes in T^*.

The derivation of the sequence of optimal moves by $A_{opt}(F_k)$ starting with T^* is illustrated in Figure 10.

We extend all of the previous definitions (i.e. $dx(p, F_k)$, $dy(p, F_k)$, etc.) to $A_{opt}(F_k)$ by considering the optimal algorithm as an extension of the A-tree algorithm. We can easily see that the O1- and O2-moves generated by $A_{opt}(F_k)$ have the same structures as the H1- and H2-moves. Notice that starting with a forest with n roots, the $A_{opt}(F_k)$ algorithm always completes the construction of the optimal arborescence after

exactly $n-1$ moves, whereas the A-tree algorithm might output more than $n-1$ moves.

With these definitions, we proceed to prove that the safe moves are optimal.

Lemma 1 *In the A-tree algorithm, if* $p \in ROOT(F_k) \cap ROOT(F_{k+1})$, *then* $dx(p, F_k) \leq dx(p, F_{k+1})$ *and* $dy(p, F_k) \leq dy(p, F_{k+1})$.

Proof: Assume that there exist a node p and an index k such that $dx(p, F_k) > dx(p, F_{k+1})$. Let $q' \leftarrow q$ be the $k+1^{th}$ move (take the vertical move if it is an H2-move), then it must be the case that $q' = mx(p, F_{k+1}) \neq mx(p, F_k)$.

The $k+1^{th}$ move cannot be an S1-move or an H1-move because they do not introduce any new node to $ROOT(F_{k+1})$. It also cannot be an S2-move or an H2-move because otherwise q' has the same x-coordinate and accessibility (i.e. whether blocked from p or not) as q, which implies that $dx(p, F_{k+1}) = dx(p, F_k)$. Hence, the $k+1^{th}$ move must be an S3-move. Moreover, $q'_x \neq mx(q, F_k)_x$ because otherwise q' and $mx(q, F_k)$ would then have the same horizontal distance and accessibility from p. Therefore, $q'_x = mf_{west}(q, F_k)_x$. We also know that $mf_{west}(q, F_k)$ is in the region $SW(p, F_k)$ (rather than in $W(p, F_k)$ or $NW(p, F_k)$) since q' is not blocked by $mf_{west}(q, F_k)$ from p.

There are two cases: (i) If q is in $N(p, F_k)$ or $NE(p, F_k)$, then $dist(p, q) < dist(mf_{west}(q, F_k), q)$ (see Figure 11(i)); (ii) If q is in $NW(p, F_k)$, then q must be blocked from p by a node r in F_k (otherwise $mx(p, F_k)_x \geq q_x \geq q'_x = mx(p, F_{k+1})_x$), and $dist(r, q) < dist(mf_{west}(q, F_k), q)$ (see Figure 11(ii)). In both cases, it contradicts to the fact that $mf_{west}(p, F_k)$ is the closest node to q among nodes in $DOM(q, F_k)$. Therefore, the $k+1^{th}$ move cannot be an S3-move either.

As a result, $dx(p, F_k) \leq dx(p, F_{k+1})$ after each move by the A-tree algorithm. Similarly, we can also show that $dy(p, F_k) \leq dy(p, F_{k+1})$ after each move. □

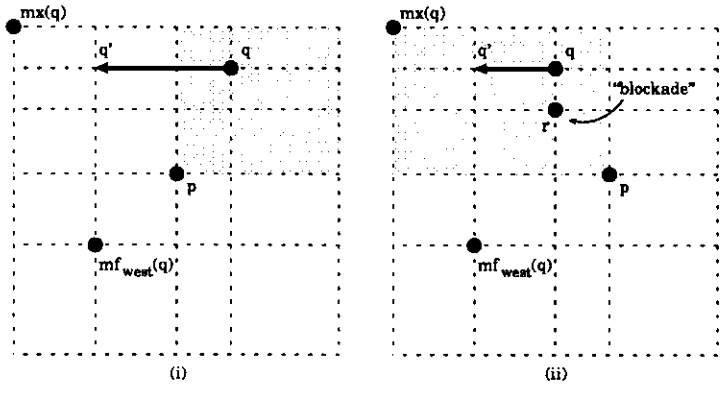


Figure 11: If $dx(p, F_k) > dx(p, F_{k+1})$, then the $k+1^{th}$ move is an S3-move originated from a node q to the vertical line $x = mf_{west}(q, F_k)_x$ and $mf_{west}(q, F_k)$ is in $SW(p)$. In this case, q is either (i) in $N(p) \cup NE(p)$, or (ii) in $NW(p)$ and blocked from p by a node r in F_k.

Lemma 2 *In the A-tree algorithm, if* $p \in ROOT(F_k) \cap ROOT(F_{k+1})$, *then* $df(p, F_k) \geq df(p, F_{k+1})$. *Moreover, if* $df(p, F_k) > df(p, F_{k+1})$, *then* $df(p, F_{k+1}) \geq \min\{dx(p, F_k), dy(p, F_k)\}$.

Proof: It is trivial to see that $df(p, F_k) \geq df(p, F_{F_{k+1}})$ because $df(p, F_k) = min\{dist(p, q) | q \in DOM(p, F_k)\}$ and $DOM(p, F_k) \subset DOM(p, F_{k+1})$ for any node p in F_k.

Assume that there exist a node p and an index k such that $df(p, F_k) > df(p, F_{k+1})$, and that the $k+1^{th}$ move $q' \leftarrow q$ contains a new node $z \in MF(p, F_{k+1}) - MF(p, F_k)$. We can easily see that q' must be in $DOM(p, F_k)$. The move $q' \leftarrow q$, however, cannot be completely dominated by p, because otherwise $z = q \in F_k$. Therefore, the move $q' \leftarrow q$ spans at least two regional sets and $z \in W(p) \cup S(p)$. In the following, we will consider the case where $z \in W(p)$.

If the $k + 1^{th}$ move is an S1-move (or an H1-move), q must be a node in $NW(p)$, $N(p)$, or $NE(p)$. There are three cases: (i) If q is in $NW(p)$ and $dist_x(q, p) \geq dx(p, F_k)$, then $df(p, F_{k+1}) = dist(z, p) = dist_x(z, p) \geq dist_x(q, p) \geq dx(p, F_k)$ (see Figure 12(i)); (ii) If q is in $NW(p)$ and $dist_x(q, p) < dx(p, F_k)$, then q must be blocked from p by a node r in F_k and $dist(r, q) < dist(q', q) = df(q, F_k)$, which leads to a contradiction (see Figure 12(ii)); (iii) If q is in $N(p)$ or $NE(p)$, then $dist(p, q) < dist(q', q) = df(q, F_k)$, which also leads to a contradiction (see Figure 12(iii)).

If the $k + 1^{th}$ move is an S2-move, $dist_x(q, p)$ is at least $dx(p, F_k)$ and therefore $df(p, F_{k+1}) = dist(z, p) = dist_x(z, p) = dist_x(q, p) \geq dx(p, F_k)$. Moreover, the $k + 1^{th}$ move cannot be an S3-move since no horizontal path contains $q \in W(p)$ unless the path is completely dominated by p. If the $k + 1^{th}$ is an H2-move, we can consider the vertical path added as an S2-move (and the horizontal path as an S3-move), which again implies that $df(p, F_{k+1}) = dist(z, p) = dist_x(z, p) \geq dx(p, F_k)$.

Therefore, $dist(z, p) \geq dx(p, F_k)$ if z is in $W(p)$. Using a similar argument, we can show that $dist(z, p) \geq dy(p, F_k)$ if z is in $S(p)$. Since $df(p, F_{k+1}) = dist(z, p)$, in both cases, we have that $df(p, F_{k+1}) = dist(z, p) \geq min\{dx(p, F_k), dy(p, F_k)\}$. Therefore, given a node p in $ROOT(F_k) \cap ROOT(F_{k+1})$, $df(p, F_k) > df(p, F_{k+1})$ implies that $df(p, F_{k+1}) \geq \min\{dx(p, F_k), dy(p, F_k)\}$. □

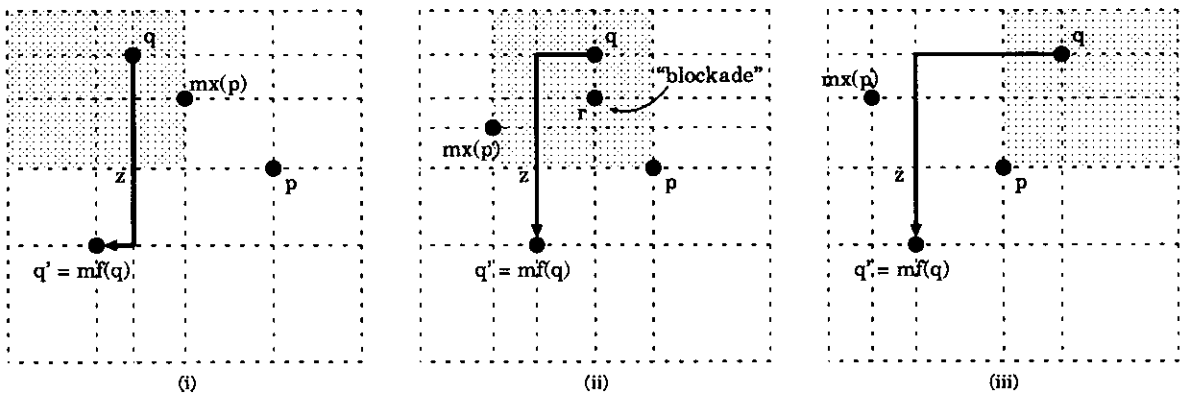


Figure 12: The case when the $k+1^{th}$ move is an S1-move or an H1-move, which introduces a path from a node q to a node $q' \in MF(q, F_k)$ in $SW(p)$. Assume that the new node $z \in MF(p, F_{k+1}) - MF(p, F_k)$ is in $W(p)$. Then, q is (i) in $NW(p)$ and $q_x \leq mx(p, F_k)_x$, (ii) in $NW(p)$ and $q_x > mx(p, F_k)_x$, or (iii) in $N(p)$ or $NW(p)$.

Using similar arguments, we can also show that Lemma 1 and Lemma 2 are also true for the sequence of forests $F_k^*, F_{k+1}^*, \cdots$ generated by the $A_{opt}(F_k)$ algorithm since O1- and O2-moves behave in the same way as H1- and H2-moves.

Corollary 1 *Let* $LB(p, F_k) = \min\{dx(p, F_k), dy(p, F_k), df(p, F_k)\}$ *for each* $p \in ROOT(F_k)$. *Then,* $LB(p, F_k) \leq LB(p, F_{k+1})$ *if* $p \in ROOT(F_{k+1})$.

Proof: According to Lemma 1, $min\{dx(p, F_{k+1}), dy(p, F_{k+1})\} \geq min\{dx(p, F_k), dy(p, F_k)\}$. If $df(p, F_k) = df(p, F_{k+1})$, $LB(p, F_{k+1}) = min\{dx(p, F_{k+1}), dy(p, F_{k+1}), df(p, F_{k+1})\} \geq min\{dx(p, F_k), dy(p, F_k), df(p, F_k)\} = LB(p, F_k)$. Otherwire, $df(p, F_k) > df(p, F_{k+1})$. According to Lemma 2, $df(p, F_{k+1}) \geq min\{dx(p, F_k), dy(p, F_k)\}$. $LB(p, F_{k+1}) = min\{min\{dx(p, F_{k+1}), dy(p, F_{k+1})\}, df(p, F_{k+1})\} \geq min\{dx(p, F_k), dy(p, F_k)\} \geq LB(p, F_k)$. □

Given a node p in $ROOT(F_k)$, it is clear that $LB(p, F_k)$ gives a lower bound of $dist(p', p)$, where p' is the parent of p in the arborescence constructed by the A-tree algorithm. We shall show that $LB(p, F_k)$ also gives a lower bound of $dist(p^*, p)$, where p^* is the parent of p in $T^*(F_k)$ constructed by $A_{opt}(F_k)$.

Corollary 2 *Let* p *be a node in* $ROOT(F_k)$, *and* p^* *be the parent of* p *in* $T^*(F_k)$. *Then* $dist(p^*, p) \geq LB(p, F_k)$.

Proof: Assume that the path $P_{T^*(F_k)}(p^*, p)$ is introduced by the $A_{opt}(F_k)$ algorithm in the j^{th} move ($j > k$). If the j^{th} move is an O1-move, $dist(p^*, p) \geq df(p, F^*_{j-1})$ because the O1-move introduces a path from p to an *existing* node in $DOM(p, F^*_{j-1})$.

If the j^{th} move is an O2-move, then p^* (a Steiner node) is the parent of p and another node $\bar{p}$ from $ROOT(F^*_{j-1})$. If $p^* \in W(p)$, $dist(p^*, p) \geq dx(p, F^*_{j-1})$; If $p^* \in S(p)$, $dist(p^*, p) \geq dy(p, F^*_{j-1})$.

In all cases, $dist(p^*, p) \geq \min\{dx(p, F^*_{j-1}), dy(p, F^*_{j-1}), df(p, F^*_{j-1})\} = LB(p, F^*_{j-1})$. And according to Corollary 1, we have,

$$dist(p^*, p) \geq LB(p, F^*_{j-1}) \geq LB(p, F^*_{j-2}) \geq \cdots \geq LB(p, F^*_k) = LB(p, F_k) \tag{14}$$

Hence, $dist(p^*, p) \geq LB(p, F_k)$. □

Theorem 1 *S1-move, S2-move, and S3-move are optimal.*

Proof: We shall show that there exists a $T^*(F_k)$ which contains F_{k+1} as a subgraph if the $k + 1^{th}$ move is a safe move. Assume that the $k + 1^{th}$ move is $p' \leftarrow p$ and that the parent of p in $T^*(F_k)$ is p^*.

1. **S1-move:** $dx(p, F_k) \geq df(p, F_k)$ and $dy(p, F_k) \geq df(p, F_k)$
   According to Corollary 2, $dist(p^*, p) \geq LB(p, F_k) = \min\{dx(p, F_k), dy(p, F_k), df(p, F_k)\} = df(p, F_k) = dist(p', p)$. Let T' be the tree modified from $T^*(F_k)$ by replacing the path $P_{T^*(F_k)}(p^*, p)$ with a rectilinear shortest path from p to p'. Then, $cost(T') \leq cost(T^*(F_k))$ and T' contains F_{k+1} as a subgraph.

2. **S2-move:** $dx(p, F_k) \geq df(p, F_k)$ and $dy(p, F_k) < df(p, F_k)$
   There are three cases: (i) If $p^* \in S(p)$, the path $P_{T^*(F_k)}(p^*, p)$ always contains the vertical path generated by the move $p' \leftarrow p$; (ii) If $p^* \in SW(p)$, then p^* must be one of the nodes in $MF(p, F_k)$ because all new nodes introduced to $MF(p, F_{k+1}), MF(p, F_{k+2}), \cdots$ are either in $W(p)$ or $S(p)$. Hence, replacing the path $P_{T^*(F_k)}(p^*, p)$ with a shortest path from p to $mf_{south}(p, F_k)$ does not increase the cost of the tree $T^*(F_k)$. Moreover, by restricting the new path $P_{T^*(F_k)}(mf_{south}(p, F_k), p)$ to go first southward from p and then westward to $mf_{south}(p, F_k)$, the modified tree contains F_{k+1} as a subgraph; (iii) If $p^* \in W(p)$,

then $dist(p^*, p) = dist_x(p^*, p) \geq \min\{dx(p, F_k), df(p, F_k)\} = df(p, F_k)$. Once again, replacing the path $P_{T^*(F_k)}(p^*, p)$ with a shortest path from p to $mf_{south}(p, F_k)$ does not increase the cost of the tree $T^*(F_k)$. Moreover, by restricting the new path $P_{T^*(F_k)}(mf_{south}(p, F_k), p)$ to first go southward from p and then westward to $mf_{south}(p, F_k)$, the modified tree contains F_{k+1} as a subgraph.

3. **S3-move:** $dx(p, F_k) < df(p, F_k)$ **and** $dy(p, F_k) \geq df(p, F_k)$
   Because of the symmetry between S2-moves and S3-moves, we can show similarly that S3-moves are optimal. □

Corollary 3 *If all moves introduced by the A-tree algorithm during the construction of T are safe moves, T is an optimal Steiner arborescence.* □

Moreover, it can be shown that Corollary 3 is also true under the $QMST$ cost function defined in Section 2, where $cost_{QMST}(T) = \sum_{k \in T} pl_k(T)$. Give a node p and an integer d, let $\sigma_{QMST}(p, d)$ denote $\sum_{i=1}^{d}(p_x + p_y - i)$, then we have the following result.

Lemma 3 *Given an arborescence T and a node p. Let $p' = parent(p)$ and $d = dist(p', p)$. Then, the total cost of the path $P_T(p', p)$ under the $QMST$ cost function is $cost_{QMST}(P_T(p', p)) = \sigma_{QMST}(p, d)$.*

Proof: Let $p_0 = p', p_1, p_2, \cdots, p_{d-1}, p_d = p$ be the sequence of nodes on the path $P_T(p', p)$ from p' to p, and $e(p_i)$ be the unit-grid-length edge connecting p_i and p_{i-1}. Since $cost_{QMST}(e(p_i)) = dist(N_0, p_0) + i = dist(N_0, p_d) - d + i$, we have:

$$\begin{aligned} & cost_{QMST}(P_T(p', p)) \\ = & \sum_{i=1}^{d} cost_{QMST}(e(p_i)) \\ = & \sum_{i=1}^{d}(dist(N_0, p_d) - d + i) \;=\; \sum_{i=0}^{d-1}(dist(N_0, p_d) - i) \;=\; \sum_{i=0}^{d-1}(p_x + p_y - i) \;=\; \sigma_{QMST}(p, d) \end{aligned}$$

□

Note that if p is fixed, we have $\sigma_{QMST}(p, d_1) \geq \sigma_{QMST}(p, d_2)$ if and only if $d_1 \geq d_2$. Based on this lemma, we have the following result:

Theorem 2 *Safe moves are optimal under the $QMST$ cost function.*

Proof: Given a forest F_k, assume that the $k + 1^{th}$ move $p' \leftarrow p$ is a safe move. We define $T^*_{QMST}(F_k)$ as the rectilinear Steiner arborescence containing F_k as a subgraph such that $cost_{QMST}(T^*_{QMST}(F_k))$ is minimum. Again, it suffices to show that there exists a $T^*_{QMST}(F_k)$ containing F_{k+1} as a subgraph. Let p^* denote $parent(p)$ in $T^*_{QMST}(F_k)$. According to the proof of Theorem 1, we can replace the path P^* connecting p^* and p in $T^*_{QMST}(F_k)$ by a path P' of the same or smaller length which contains the path generated by the $p' \leftarrow p$ move. Let T' denote the resulting tree. Let $d^* = dist(p^*, p)$ and d' be the length of P'. Since T' contains F_{k+1}

and $d' \leq d^*$, we have:

$$\begin{aligned} & cost_{QMST}(T^*_{QMST}(F_{k+1})) - cost_{QMST}(T^*_{QMST}(F_k)) \\ \leq\ & cost_{QMST}(T') - cost_{QMST}(T^*_{QMST}(F_k)) \\ =\ & \sigma_{QMST}(p, d') - \sigma_{QMST}(p, d^*) \ \leq\ 0 \end{aligned}$$

Hence, $cost_{QMST}(T^*_{QMST}(F_{k+1})) \leq cost_{QMST}(T^*_{QMST}(F_k))$. □

Corollary 4 *If the complete construction of an arborescence is done with only safe moves, then the A-tree is an optimal A-tree under the SPT, OST, and QMST cost.* □

Our experimental results indicate that 96% of the moves used by our A-tree algorithm are optimal in practice, and 65% of the A-trees constructed by our algorithm are optimal under both of the OST and $QMST$ cost function since they are generated by safe moves only.

3.4 Lower Bound of the Optimal Cost

In this subsection, we shall derive a lower bound of the cost of the optimal Steiner arborescence (i.e. $cost(T^*(F_0))$) based on the information obtained during the construction of the A-tree T. Let π_{k+1} denote the $k+1^{th}$ move. We define $SB(\pi_{k+1})$ as follows: If π_{k+1} is a safe move, $SB(\pi_{k+1}) = 0$; If π_{k+1} is an H1-move $p' \leftarrow p$, $SB(\pi_{k+1}) = dist(p', p) - LB(p, F_k)$; If π_{k+1} is an H2-move $(p' \leftarrow p_1) \cup (p' \leftarrow p_2)$, $SB(\pi_{k+1}) = dist(p', p_1) + dist(p', p_2) + df(p', F_{k+1}) - LB(p_1, F_k) - LB(p_2, F_k)$.

We keep an accumulated error count $ERROR$ (initialized to 0) during our A-tree construction. After we make the $k+1^{th}$ move, we set $ERROR \leftarrow ERROR + SB(\pi_{k+1})$. Now we shall prove that $cost(T) - ERROR$ is a lower bound of the cost of the optimal A-tree (i.e. $cost(T^*(F_0)) \geq cost(T) - ERROR$).

Lemma 4 $cost(T^*(F_{k+1})) - cost(T^*(F_k)) \leq SB(\pi_{k+1})$.

Proof: If the $k+1^{th}$ move π_{k+1} is a safe move, according to Theorem 1, $cost(T^*(F_{k+1})) = cost(T^*(F_k))$. If the $k+1^{th}$ move is an H1-move $p' \leftarrow p$, let the parent of p in $T^*(F_k)$ be p^*. According to Corollary 2, $dist(p^*, p) \geq LB(p, F_k)$. We obtain a modified tree T' containing F_{k+1} from $T^*(F_k)$ by replacing the path $P_{T^*(F_k)}(p^*, p)$ with a shortest path from p to p'. Hence, the difference between $cost(T^*(F_{k+1}))$ and $cost(T^*(F_k))$ is given by:

$$\begin{aligned} & cost(T^*(F_{k+1})) - cost(T^*(F_k)) \\ \leq\ & cost(T') - cost(T^*(F_k)) \\ =\ & dist(p', p) - dist(p^*, p) \ \leq\ dist(p', p) - LB(p, F_k) \ =\ SB(\pi_{k+1}) \end{aligned} \tag{15}$$

Otherwise, if the $k+1^{th}$ move π_{k+1} is an H2-move $(p' \leftarrow p_1) \cup (p' \leftarrow p_2)$, let the parents of p_1 and p_2 in $T^*(F_k)$ be p_1^* and p_2^* respectively. According to Corollary 2, $dist(p_1^*, p_1) \geq LB(p_1, F_k)$ and $dist(p_2^*, p_2) \geq LB(p_2, F_k)$. We obtain a modified tree T' containing F_{k+1} from $T^*(F_k)$ by replacing the paths $P_{T^*(F_k)}(p_1^*, p_1)$ and $P_{T^*(F_k)}(p_2^*, p_2)$ with shortest paths from p_1 to p', from p_2 to p', and from p' to $mf_{west}(p', F_{k+1})$. The difference between $cost(T^*(F_{k+1}))$ and $cost(T^*(F_k))$ is given by:

$$cost(T^*(F_{k+1})) - cost(T^*(F_k))$$

$$
\begin{aligned}
&\leq cost(T') - cost(T^*(F_k)) \\
&= dist(p', p_1) + dist(p', p_2) + df(p', F_{k+1}) - dist(p_1^*, p_1) - dist(p_2^*, p_2) \\
&\leq dist(p', p_1) + dist(p', p_2) + df(p', F_{k+1}) - LB(p_1, F_k) - LB(p_2, F_k) \\
&= SB(\pi_{k+1}) \qquad (16)
\end{aligned}
$$

□

Theorem 3 *The cost of the arborescence constructed by the A-tree algorithm is at most* $\sum_{j=1}^{M} SB(\pi_k)$ *higher than that of the optimal arborescence, where* M *is the number of moves made by the A-tree algorithm.*

Proof: Assume that F_M is the forest constructed by the A-tree algorithm after M moves. According to Lemma 4,

$$cost(T^*(F_{k+1})) - cost(T^*(F_k)) \leq SB(\pi_{k+1}) \qquad (17)$$

Summing Equation 17 from $k = 0$ to $M - 1$, we obtain the following inequality:

$$cost(T^*(F_M)) - cost(T^*(F_0)) \leq \sum_{k=0}^{M-1} SB(\pi_{k+1}) \qquad (18)$$

Since $cost(T^*(F_M)) = cost(F_M)$, we have,

$$cost(T^*(F_0)) \geq cost(F_M) - \sum_{k=1}^{M} SB(\pi_k) \qquad (19)$$

□

Note that $\sum_{k=1}^{M} SB(\pi_k)$ can be computed easily during the A-tree construction. Using a similar argument, we can derive the *suboptimality bound* for the heuristic moves under the $QMST$ cost function as follows: If π_{k+1} is a safe move, $SB_{QMST}(\pi_{k+1}) = 0$; If π_{k+1} is an H1-move $p' \leftarrow p$, $SB_{QMST}(\pi_{k+1}) = \sigma_{QMST}(p, dist(p', p)) - \sigma_{QMST}(p, LB(p, F_k))$; If π_{k+1} is an H2-move $(p' \leftarrow p_1) \cup (p' \leftarrow p_1)$, $SB(\pi_{k+1}) = \sigma_{QMST}(p_1, dist(p', p_1)) + \sigma_{QMST}(p_2, dist(p', p_1)) + \sigma_{QMST}(p', df(p', F_{k+1})) - \sigma_{QMST}(p_1, LB(p_1, F_k)) - \sigma_{QMST}(p_2, LB(p_2, F_k))$.

Finally, a lower bound of the cost of the optimal quadratic minimum Steiner arborescence is given by:

$$cost_{QMST}(F_M) - \sum_{k=1}^{M} SB_{QMST}(\pi_k) \qquad (20)$$

In fact, there is a better strategy for finding a lower bound of $cost(T^*(F_0))$ (and $cost_{QMST}(T^*_{QMST}(F_0))$): We follow the A-tree algorithm, except that when a heuristic move is needed, we try to find a move π which *minimizes* $SB(\pi)$ rather than to find a move that *maximizes* $dist(N_0, p')$ as is the normal A-tree algorithm. This strategy usually gives a lower bound of $cost(T^*(F_0))$ (and $T^*_{QMST}(F_0)$) in practice.

Using the above strategy for lower bound computation, we are able to conclude that the arborescences generated by our A-tree algorithm are on average at most 3% from the optimal.

Since the A-tree algorithm discussed above constructs an A-tree by iteratively introducing new edges, we can directly extend the A-tree algorithm to handle the case when no restriction is imposed on the location

of the sinks. In the generalized version, routing is performed for all quadrants simultaneously. Most of the results discussed, including the optimality analysis and the lower bound analysis, still hold for this generalized version if the definitions of $dx(p, F_k)$, $dy(p, F_k)$, etc. are generalized accordingly. Experimental results show that 94 % of the moves introduced by the generalized A-tree algorithm are optimal, and that 45 % of the A-trees constructed are optimal under both of the OST and $QMST$ cost. On average, the A-trees generated by the generalized A-tree algorithm are at most 4 % from the optimal.

4 Wiresizing

In this section, we shall discuss the properties of the wiresizing problem and present an efficient wiresizing algorithm for performance optimization. First, we introduce a few definitions used in the discussion of the wiresizing problem.

Given a fixed tree T, we denote $w(f, S_i)$ as the width assignment of segment S_i in the assignment f (or simply w_i if the context is clear), and f^* as the optimal wire width assignment. In most cases, we will focus on a special kind of trees, called *single-stem trees* (SS-trees). Each SS-tree contains only one segment (called the *stem* of that tree) incident on the root node. A general routing tree can be decomposed into a set of SS-trees, as illustrated in Figure 13. Although segments, rather than nodes, are involved in the discussion of the wiresizing problem, we will borrow the notions of *ancestors* and *descendants* to refer to segments in the same way as these terms are defined for nodes. In particular, we use $ans(S_i)$ to denote the set of all segments on the unique path from the source N_0 to the segment S_i, excluding S_i, and refer $des(S_i)$ to the set of segments $\{S_k \mid S_i \in ans(S_k)\}$. Moreover, we use $T_{SS}(S_i)$ to denote the single-stem subtree implemented by S_i (as the stem) and the set $des(S_i)$. We use $sink(S_i)$ to refer to the set of all sinks on $T(S_i)$.

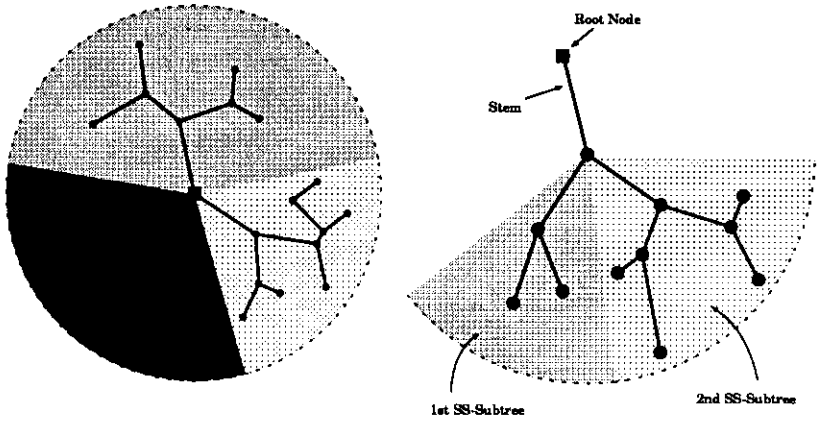


Figure 13: (a) Any general tree T can be decomposed into a set of independent single-stem trees; (b) A single-stem tree consists of a stem and a set of single-stem subtrees.

From Equation 9, we know that the minimum wire width is desirable if the resistance ratio is very high, making $t_1(T)$ the dominant factor in delay optimization. However, when the resistance ratio is small, wider widths help reduce $t_1(T)$ and $t_2(T)$, especially when the signal path is very long. According to Equations 11 and 12, we can see intuitively that larger wire width should be used for the segments near the source, and

thinner widths for the segments far away from the source. In fact, this turns out to be a very general property of any optimal wire width assignment based on our distributed delay model.

Definition 10 *Given a routing tree T, a wire width assignment f on T is a monotonic assignment if $w_p \geq w_c$ whenever segment S_p is an ancestor of segment S_c.*

Theorem 4 *Any optimal wire width assignment f^* is a monotonic assignment.*

Proof: The basic idea of the proof is to show that if we find two segments S_p and S_c in the optimal assignment f^* such that S_p is the parent of S_c and $w_p < w_c$, then the total delay will be reduced by simply increasing the segment width of S_p from w_p to w_c (see Figure 14), which leads to a contradiction to the optimality assumption of f^*. The formal proof is as follows:

Denote $t(f)$ as the total delay of the routing tree T with wire width assignment f. We can rewrite Equation 9 to show the relationship between the width of a particular segment S_i with other segments in the delay formula as follows:

$$\begin{aligned} t(f) &= K(\overline{S_i}) + R_d \cdot C_0 \cdot w_i \cdot l_i + R_0 \cdot \sum_{k \epsilon sink(S_i)} C_k \cdot \frac{1}{w_i} \cdot l_i \\ &\quad + R_0 \cdot C_0 \cdot \sum_{k \epsilon des(S_i)} \frac{w_k}{w_i} \cdot l_i + R_0 \cdot C_0 \cdot \sum_{k \epsilon ans(S_i)} \frac{w_i}{w_k} \cdot l_i \end{aligned} \tag{21}$$

where l_i is the length of the segment S_i, and $K(\overline{S_i})$ involves terms independent of the width (w_i) of the segment S_i. Note that the terms involved in Equation 21 refer to segments rather than grids.

Assume f^* is not monotonic, then there must exist segments S_p and S_c such that S_p is the parent of S_c and $w_p < w_c$. Since f^* is optimal, reducing the wire width of S_c from w_c to w_p would not reduce total delay (see Figure 14).

Define $\delta t(f^*, S_c, w_c \rightarrow w_p)$ as the change in total delay of the routing tree when the width of S_c in f^* is changed from w_c to w_p. We can obtain from Equation 21 the following inequality:

$$\begin{aligned} \delta t(f^*, S_c, w_c \rightarrow w_p) &= R_d \cdot C_0 \cdot (w_p - w_c) \cdot l_c + R_0 \cdot \sum_{k \epsilon sink(S_c)} C_k \cdot (\frac{1}{w_p} - \frac{1}{w_c}) \cdot l_c \\ &\quad + R_0 \cdot C_0 \cdot \sum_{k \epsilon des(S_c)} (\frac{w_k}{w_p} - \frac{w_k}{w_c}) \cdot l_c + R_0 \cdot C_0 \cdot \sum_{k \epsilon ans(S_c)} (\frac{w_p}{w_k} - \frac{w_c}{w_k}) \cdot l_c \\ &= tm_1(S_c) + tm_2(S_c) + tm_3(S_c) + tm_4(S_c) \\ &\geq 0 \end{aligned} \tag{22}$$

where $tm_j(S_c)$ is the j^{th} term in the expansion of $\delta(f^*, S_c, w_c \rightarrow w_p)$.

Now reconsider the original optimal solution f^*. Increasing the wire width of S_p from w_p to w_c results in the following change in delay:

$$\begin{aligned} \delta t(f^*, S_p, w_p \rightarrow w_c) &= R_d \cdot C_0 \cdot (w_c - w_p) \cdot l_p + R_0 \sum_{k \epsilon sink(S_p)} C_k \cdot (\frac{1}{w_c} - \frac{1}{w_p}) \cdot l_p \\ &\quad + R_0 \cdot C_0 \sum_{k \epsilon des(S_p)} (\frac{w_k}{w_c} - \frac{w_k}{w_p}) \cdot l_p + R_0 \cdot C_0 \sum_{k \epsilon ans(S_p)} (\frac{w_c}{w_k} - \frac{w_p}{w_k}) \cdot l_p \\ &= tm_1(S_p) + tm_2(S_p) + tm_3(S_p) + tm_4(S_p) \end{aligned} \tag{23}$$

$$\tag{24}$$

where $tm_j(S_p)$ is the j^{th} term in the expansion of $\delta(f^*, S_p, w_p \rightarrow w_c)$.

Since S_c is a child of S_p, we have $des(S_c) \subset des(S_p)$, $sink(S_c) \subseteq sink(S_p)$, and $ans(S_c) \supset ans(S_p)$. Therefore, we can express $\delta(f^*, S_p, w_p \rightarrow w_c)$ in terms of $\delta(f^*, S_c, w_c \rightarrow w_p)$ plus some adjustment terms.

$$tm_1(S_p) = -tm_1(S_c) \cdot \frac{l_p}{l_c} \tag{25}$$

$$tm_2(S_p) = -tm_2(S_c) \cdot \frac{l_p}{l_c} + R_0 \cdot \sum_{k \in sink(S_p) - sink(S_c)} C_k \cdot (\frac{1}{w_c} - \frac{1}{w_p}) \cdot l_p \tag{26}$$

$$tm_3(S_p) = -tm_3(S_c) \cdot \frac{l_p}{l_c} + R_0 \cdot C_0 \cdot \sum_{k \in des(S_p) - des(S_c)} (\frac{w_k}{w_c} - \frac{w_k}{w_p}) \cdot l_p \tag{27}$$

$$tm_4(S_p) = -tm_4(S_c) \cdot \frac{l_p}{l_c} - R_0 \cdot C_0 \cdot \sum_{k=S_p} (\frac{w_c}{w_k} - \frac{w_p}{w_k}) \cdot l_p \tag{28}$$

Sum all these terms together, we obtain:

$$\begin{aligned} & \delta(f^*, S_p, w_p \rightarrow w_c) \\ = \; & -\delta(f^*, S_c, w_c \rightarrow w_p) \cdot \frac{l_p}{l_c} - A \cdot (\frac{1}{w_p} - \frac{1}{w_c}) - B \cdot (\frac{1}{w_p} - \frac{1}{w_c}) - C \cdot (w_c - w_p) \end{aligned} \tag{29}$$

where

$$A = R_0 \cdot \sum_{k \in sink(S_p) - sink(S_c)} C_k \cdot l_p \geq 0$$

$$B = R_0 \cdot C_0 \cdot \sum_{k \in des(S_p) - des(S_c)} w_k \cdot l_p \geq 0$$

$$C = R_0 \cdot C_0 \cdot \sum_{k=S_p} \frac{1}{w_k} \cdot l_p > 0$$

Note that $w_c > w_p$ implies $w_c - w_p > 0$ and $\frac{1}{w_c} - \frac{1}{w_p} < 0$. Hence $\delta(f^*, S_p, w_p \rightarrow w_c) < -\delta(f^*, S_c, w_c \rightarrow w_p) \leq 0$. As a result, we can reduce the cost of f^* by increasing the width of S_p from w_p to w_c (see Figure 14), which leads to a contradiction. Hence, f^* must be a monotonic assignment. □

According to the monotone property, the optimal wire width assignment f^* can be represented by a set of "wavefronts" $V_1, V_2, \cdots, V_r$ radiating outward from the source N_0. Each wavefront defines the boundary where the segment width decreases, as illustrated in Figure 15. Wavefronts do not intersect (except that they may touch each others at non-trivial nodes in the tree), and all the segments enclosed between V_{k+1} and V_k have width W_k.

4.1 Optimal Wiresizing Algorithm

According to Equation 21, the width assignment of segment S_i depends only on the widths of its ancestors and its descendants. Therefore, once S_i and every segment in $ans(S_i)$ are assigned the appropriate widths, the optimal wire width assignment for the single-stem subtrees $T_{SS}(S_{i,1}), T_{SS}(S_{i,2}), \cdots, T_{SS}(S_{i,k})$ (with respect to the width assignment of S_i and segments in $ans(S_i)$) can be *independently* determined, where the segments $S_{i,1}, \cdots, S_{i,k}$ are the children of S_i (see Figure 16). Based on this property and the monotone property, we have developed an optimal wiresizing algorithm.

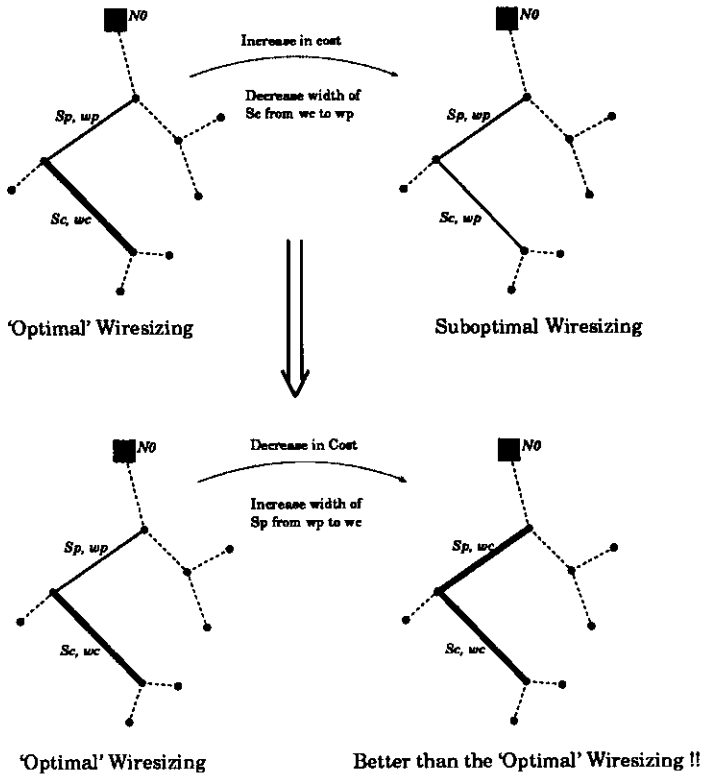


Figure 14: Illustration of the proof of the monotone property

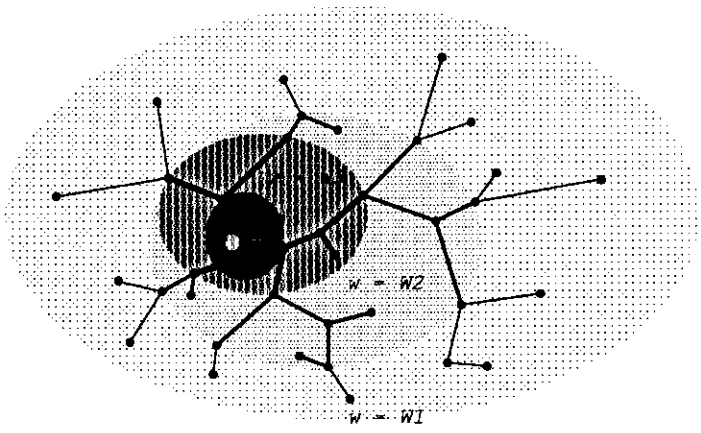


Figure 15: The optimal wire width assignment can be represented by a set of wavefronts.

Assume we are given a single-stem tree $T_{SS}(S_i)$ with stem segment S_i, and a set of possible widths $\{W_1, W_2, \cdots, W_r\}$, we can determine the optimal assignment f^* on $T_{SS}(S_i)$ by enumerating all the possible width assignments of S_i. For each of the possible width assignment W_k of S_i $(1 \leq k \leq r)$, we determine the optimal assignment for each single-stem subtree $T_{SS}(S_{i,j})$ independently by recursively applying the same procedure to each $T_{SS}(S_{i,j})$ with $\{W_1, W_2, \cdots, W_k\}$ as the set of possible widths (to guarantee the monotone property). The optimal assignment for S_i is the one which gives the smallest total delay.

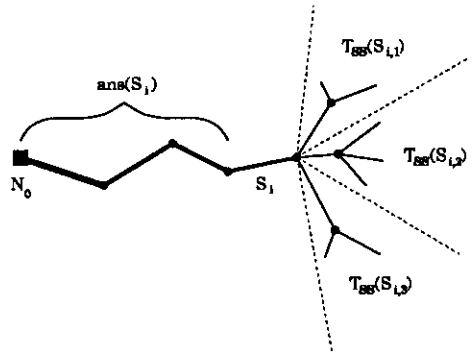


Figure 16: If S_i and every segment in $ans(S_i)$ are assigned the proper widths, the optimal wire width assignment for the single-stem subtrees $T_{SS}(S_{i,1}), T_{SS}(S_{i,2}), \cdots, T_{SS}(S_{i,k})$ can be *independently* determined.

If the original routing tree T is not a single-stem tree, we can decompose T into b single-stem trees, where b is the degree of the root of T, and apply the algorithm to each individual single-stem tree separately (see Figure 13(a)). Our optimal wiresizing algorithm ($OWSA$) is formally described in Figure 2.

Optimal Wire Sizing Algorithm (OWSA)

```
Function OWSA(T_SS, r, assignment)
    best_assignment ← minimum width W_1 for all segments in T_SS;
    for each width W_k, 2 ≤ k ≤ r do
        stem width of T_SS ← W_k;
        for each single stem subtree T_SS,i of T_SS do
            OWSA(T_SS,i, k, assignment);
        end for;
        if delay(assignment) < delay(best_assignment) then
            best_assignment ← assignment;
        end if;
    end for;
    return best_assignment;
end function;
```

Table 2: The Optimal Wiresizing Algorithm (OWSA).

Theorem 5 *Given a routing tree with n segments and r possible wire widths, the $OWSA$ algorithm has a worst case time complexity of $O(n^{r-1})$.*

Proof: First, assume that T is a single-stem tree. Let $N(n,r)$ be the *maximum* number of calls to *Function OWSA* among all possible single-stem routing trees with n segments and r possible choices of wire widths. For $n = 1$, or $r = 1$, it is easy to see that $N(1,r) = 1$ and $N(n,1) = 1$. Consider a general n-segment single-stem tree with b single-stem subtrees connected to the stem. Assume that the k^{th} subtrees has n_k segments. For each $2 \leq k \leq r$, there are $\sum_{i=1}^{b} N(n_i,k)$ calls to *Function OWSA*. Hence, the maximum number of calls to *Function OWSA* would be $1 + \sum_{j=2}^{r} \sum_{i=1}^{b} N(n_i,j)$. In short, we can express $N(n,r)$ as follows:

$$N(n,r) = \begin{cases} \max_{0<b\leq n-1, n_1+\cdots+n_b=n-1} \left\{1 + \sum_{j=2}^{r} \sum_{i=1}^{b} N(n_i,j)\right\} & n,r \geq 2 \\ 1 & n = 1 \text{ or } r = 1 \end{cases} \quad (30)$$

We proceed to prove that $N(n,r)$ is bounded by n^{r-1} using mathematical induction on n and r: For $n = 1$, we have $N(1,r) = 1 \leq 1^{r-1}$ for $r > 0$. For $r = 1$, we have $N(n,1) = 1 \leq n^{1-1}$ for $n > 0$. Assume that $N(k,r) \leq k^{r-1}$ for $r > 0$ and $k \leq m$. Then,

$$\begin{aligned}
N(m+1,r) &= 1 + \max_{b,n_i>0,n_1+\cdots+n_b=m} \left\{ \sum_{j=2}^{r} \sum_{i=1}^{b} N(n_i,j) \right\} \\
&\leq 1 + \max_{b,n_i>0,n_1+\cdots+n_b=m} \left\{ \sum_{j=2}^{r} \sum_{i=1}^{b} n_i^{j-1} \right\} \\
&\leq 1 + \max_{b,n_i>0,n_1+\cdots+n_b=m} \left\{ \sum_{j=2}^{r} \left(\sum_{i=1}^{b} n_i \right)^{j-1} \right\} \\
&= 1 + \max_{b,n_i>0,n_1+\cdots+n_b=m} \left\{ \sum_{j=2}^{r} (m^{j-1}) \right\} \\
&= 1 + \sum_{j=1}^{r-1} m^j \leq 1 + \sum_{j=1}^{r-1} \binom{r-1}{j} \cdot m^j = (m+1)^{r-1} \qquad (31)
\end{aligned}$$

By induction, $N(n,r) \leq n^{r-1}$ for any n and r.

If T is not a single-stem tree, however, we can consider T as a union of b single-stem trees where b is the degree of the root (see Figure 13(a)), and apply $OWSA$ to determine the optimal wire width assignment for each tree independently. The overall complexity is $O(n_1^{r-1}) + O(n_2^{r-1}) + \cdots + O(n_b^{r-1}) = O(n^{r-1})$, where n_i is the number of segments in the i^{th} single-stem tree, and $\sum_{i=1}^{b} n_i = n$. Each call to *Function OWSA* requires at most $O(B \cdot r)$ time, where B is the maximum degree in the tree. Both B and r can be considered as constants (in the Manhattan plane, $B \leq 4$, and r is a small constant in practice). Hence the overall complexity of OWSA applying to single-stem trees is $O(n^{r-1})$. □

In essence, $OWSA$ enumerates *all* the possible combinations of monotonic wire width assignments along every source-to-leaf path in the routing tree. The complexity indeed can grow exponentially with respect to r. This is the case when the tree is simply a chain of segments, where the total number of possible assignments evaluated by the OWSA algorithm equals to $\binom{n+r-1}{r-1} = \Omega(n^{r-1})$.

Nevertheless, our optimal wiresizing algorithm is a significant improvement over the brute-force enumeration method which has complexity $O(r^n)$. In the next two subsections, we shall show how to further improve the runtime of the $OWSA$ algorithm.

We would like to point out that a simple *bottom-up dynamic programming* approach, where the width assignment of each subtree is determined independent of its ancestors, does not produce optimal solutions in general. This is due to the fact that the optimal width assignment of any particular segment S_i depends on the wire width assignment of both its descendants and ancestors. In fact, our experimental results indicate that the wire width assignments generated by such an approach are in general relatively poor in quality.

4.2 Greedy Wiresizing Algorithm

In this subsection, we present a simple greedy approach based on an iterative refinement technique for efficient wire width assignment. We also present several important results that characterize the wiresizing solutions

produced by the greedy algorithm. These results allow us to achieve significant speed-up of the optimal wiresizing algorithm when we incorporate the greedy algorithm into the *OWSA* algorithm.

Definition 11 *Given a routing tree T, a wire width assignment f on S, and a particular segment $S_i \in T$. A local refinement on S_i is defined as the operation to determine the optimal segment width of S_i (with respect to the fixed assignment of f on the other segments). $\tilde{w}_i$ is called the locally optimal width of S_i with respect to f.*

It is obvious that a single local refinement operation can be performed in linear time (in fact, constant time if sufficient information is kept). Based on this operation, we have developed a greedy algorithm: starting with an initial wire width assignment (say, all segments have the minimum width), we traverse the tree and perform refinement on each segment whenever possible. This process is repeated until no improvement is achieved on any segment in the last round of traversal. The greedy wiresizing algorithm (GREWSA) is described formally in Table 3.

Greedy Wire Sizing Algorithm (GREWSA)

```
Function Greedy_Improvement(T_SS)
    width of the stem of T_SS ← locally_optimal_width(current assignment);
    For each single stem subtree T_i(k) of T_SS do
        Greedy_Improvement(T_i(k));
    end for;
end function;

Procedure GREWSA(T)
    For each single – stem subtree T_SS of T do
        repeat
            Greedy_Improvement(T_SS);
        until no further improvement;
    end for;
end procedure;
```

Table 3: The Greedy Wiresizing Algorithm (GREWSA).

Despite its greedy nature, GREWSA performs very well in terms of the quality of assignment and runtime (see Section 5). In fact, GREWSA generates optimal assignments when there are only two choices of wire widths.

Theorem 6 *GREWSA is optimal when r equals 2.*

Proof: Assume that GREWSA is not optimal when r equals 2, i.e. the final wire width assignment f generated by GREWSA is not identical to f^*. Then there must exist a segment S_j on a source-to-leaf path $S_0, \cdots, S_i, \cdots, S_j, \cdots, S_k$ such that S_i and S_j are the *first* and *last* segments in the path that the two width assignments f and f^* disagree (i.e. for every segment which is either an ancestor of S_i or a descendant of S_j, its width assignments in f and f^* are the same).

Let's assume that $w_j^* > w_j$ (we will omit the proof for the case $w_j^* < w_j$, which is similar to this proof). The width assignments of f and f^* must satisfy the following condition:

$$w_k^* = \begin{cases} W_2 & S_k \in ans(S_j) \\ W_2 & S_k = S_j \\ W_1 & S_k \in des(S_j) \end{cases} \quad \text{and } w_k = \begin{cases} W_2 & S_k \in ans(S_i) \\ W_1 & S_k = S_i \\ W_1 & S_k \in des(S_i) \end{cases} \tag{32}$$

and therefore,

$$w_k^* \begin{cases} = w_k & S_k \in ans(S_i) \\ > w_k & S_k \in \{S_i, \cdots, S_j\} \\ = w_k & S_k \in des(S_j) \end{cases} \tag{33}$$

According to Equation 21, for the optimal assignment f^*, the increase in cost induced by reducing the width of S_j from w_j^* $(= W_2)$ to w_j $(= W_1)$ is given by:

$$\begin{aligned} \delta t(f^*, S_j, W_2 \rightarrow W_1) &= R_d \cdot C_0 \cdot (W_1 - W_2) \cdot l_j + R_0 \cdot \sum_{k \in sink(S_j)} C_k \cdot (\frac{1}{W_1} - \frac{1}{W_2}) \cdot l_j \\ &\quad + R_0 \cdot C_0 \sum_{k \in des(S_j)} (\frac{w_k^*}{W_1} - \frac{w_k^*}{W_2}) \cdot l_j + R_0 \cdot C_0 \sum_{k \in ans(S_j)} (\frac{W_1}{w_k^*} - \frac{W_2}{w_k^*}) \cdot l_j \\ &\geq 0 \end{aligned} \tag{34}$$

According to Equation 21, for the assignment f generated by GREWSA, the increase in cost induced by reducing the width of S_i from w_i $(= W_1)$ to w_i^* $(= W_2)$ is given by:

$$\begin{aligned} \delta t(f, S_i, W_1 \rightarrow W_2) &= R_d \cdot C_0 \cdot (W_2 - W_1) \cdot l_i + R_0 \cdot \sum_{k \in sink(S_i)} C_k \cdot (\frac{1}{W_2} - \frac{1}{W_1}) \cdot l_i \\ &\quad + R_0 \cdot C_0 \sum_{k \in des(S_i)} (\frac{w_k}{W_2} - \frac{w_k}{W_1}) \cdot l_i + R_0 \cdot C_0 \sum_{k \in ans(S_i)} (\frac{W_2}{w_k} - \frac{W_1}{w_k}) \cdot l_i \end{aligned} \tag{35}$$

As in the proof of the monotone property, we can obtain a set of inequalities by comparing $\delta t(f^*, S_j, W_2 \rightarrow W_1)$ and $\delta t(f, S_i, W_1 \rightarrow W_2)$ term by term. We also use the fact that S_j is a descendant of S_i, $W_2 > W_1$, and $w_k^* \geq w_k$ for all segments S_k in $ans(S_j) \cup \{S_j\} \cup des(S_j)$:

$$R_0 \cdot C_0 \cdot (W_1 - W_2) \cdot l_j = -\{R_0 \cdot C_0 \cdot (W_2 - W_1) \cdot l_i\} \cdot \frac{l_j}{l_i} \tag{36}$$

$$\begin{aligned} R_0 \cdot \sum_{k \in sink(S_j)} C_k \cdot (\frac{1}{W_1} - \frac{1}{W_2}) \cdot l_j &\leq R_0 \cdot \sum_{k \in sink(S_i)} C_k \cdot (\frac{1}{W_1} - \frac{1}{W_2}) \cdot l_j \\ &= -\left\{ R_0 \cdot \sum_{k \in sink(S_i)} C_k \cdot (\frac{1}{W_2} - \frac{1}{W_1}) \cdot l_i \right\} \cdot \frac{l_j}{l_i} \end{aligned} \tag{37}$$

$$\begin{aligned} R_0 \cdot C_0 \sum_{k \in des(S_j)} (\frac{w_k^*}{W_1} - \frac{w_k^*}{W_2}) \cdot l_j &= R_0 \cdot C_0 \sum_{k \in des(S_j)} (\frac{w_k}{W_1} - \frac{w_k}{W_2}) \cdot l_j \\ &\leq -\left\{ R_0 \cdot C_0 \sum_{k \in des(S_i)} (\frac{w_k}{W_2} - \frac{w_k}{W_1}) \cdot l_i \right\} \cdot \frac{l_j}{l_i} \end{aligned} \tag{38}$$

$$\begin{aligned} R_0 \cdot C_0 \sum_{k \in ans(S_j)} (\frac{W_1}{w_k^*} - \frac{W_2}{w_k^*}) \cdot l_j &\leq R_0 \cdot C_0 \sum_{k \in ans(S_i)} (\frac{W_1}{w_k^*} - \frac{W_2}{w_k^*}) \cdot l_j \\ &= -\left\{ R_0 \cdot C_0 \sum_{k \in ans(S_i)} (\frac{W_2}{w_k} - \frac{W_1}{w_k}) \cdot l_i \right\} \cdot \frac{l_j}{l_i} \end{aligned} \tag{39}$$

By adding up these inequalities, we obtain the following result:

$$0 \leq \delta t(f^*, S_j, W_2 \rightarrow W_1) \leq -\delta t(f, S_i, W_1 \rightarrow W_2) \cdot \frac{l_j}{l_i} \tag{40}$$

The equality holds when $\delta t(f^*, S_j, W_2 \rightarrow W_1)$ is zero and $S_i = S_j$, meaning that no disagreement of assignments among the ancestors and descendants of S_j. If this is the case, a local refinement on S_j will result in an optimal assignment of w_j, leading to a contradiction.

Hence, $\delta t(f, S_i, W_1 \rightarrow W_2) < 0$, and a local refinement on S_i will reduce the total delay of the assignment f. This is a contradiction to the assumption that f is the final assignment generated by GREWSA. □

In addition to the good performance of the GREWSA algorithm on the designs with a small number of width choices, we can show that assignments generated by GREWSA have the *dominance property* (defined below), which allows us to derive the lower and upper bounds of each wire segment using the GREWSA algorithm very efficiently. In most circumstances, we are able to obtain identical lower and upper bounds of all segments in the tree using the GREWSA algorithm, which lead to an optimal assignment.

Definition 12 *Given two wire width assignments f and f' on the same tree T, f dominates f' if and only if $w(f, S_i) \geq w(f', S_i)$ for all $S_i \in T$.*

Theorem 7 *Given a wire width assignment f (possibly suboptimal) and a segment S_i on the routing tree T. Assume that f^* is the optimal assignment and that f' is an assignment obtained by performing a local refinement on the wire segment S_i such that*

$$f'(S) = \begin{cases} f(S), & S \neq S_i \\ \tilde{w}(f, S), & S = S_i \end{cases} \tag{41}$$

where $\tilde{w}(S_i)$ is the locally optimal width of S_i with respect to f.

Then f' dominates (is dominated by) f^* if and only if f dominates (is dominated by) f^*.

Proof: According to Equation 21, the delay associated with the routing tree before the local refinement is given by:

$$\begin{aligned} t(f) = {} & K(\overline{S_i}) + R_d \cdot C_0 \cdot w_i \cdot l_i + R_0 \cdot \sum_{k \in sink(S_i)} C_k \cdot \frac{1}{w_i} \cdot l_i \\ & + R_0 \cdot C_0 \cdot \sum_{k \in des(S_i)} \frac{w_k}{w_i} \cdot l_i + R_0 \cdot C_0 \cdot \sum_{k \in ans(S_i)} \frac{w_i}{w_k} \cdot l_i \end{aligned} \tag{42}$$

Grouping appropriate terms together, we obtain the following sum:

$$t(f) = \Psi(f, S_i) + \Theta(f, S_i) \cdot w_i + \Phi(f, S_i) \cdot \frac{1}{w_i} \tag{43}$$

where

$$\Psi(f, S_i) = K(\overline{S_i}) \tag{44}$$

$$\Theta(f, S_i) = R_d \cdot C_0 \cdot l_i + R_0 \cdot C_0 \cdot \sum_{k \in ans(S_i)} \frac{1}{w_k} \cdot l_i \tag{45}$$

$$\Phi(f, S_i) = R_0 \cdot \sum_{k \in sink(S_i)} C_k \cdot l_i + R_0 \cdot C_0 \cdot \sum_{k \in des(S_i)} w_k \cdot l_i \tag{46}$$

Note that $\Psi(f, S_i)$ is a constant during the refinement for S_i since f is fixed for the rest of the tree. It is easy to see that for any two assignments f_1 and f_2 such that f_1 dominates f_2, we have $\Theta(f_1, S_i) \leq \Theta(f_2, S_i)$ and $\Phi(f_1, S_i) \geq \Phi(f_2, S_i)$.

During the local refinement of S_i, the total cost of the routing tree is locally optimal *if and only if* the following expression is minimized:

$$\Psi(f, S_i) \; + \; \Theta(f, S_i) \cdot w_i \; + \; \Phi(f, S_i) \cdot \frac{1}{w_i} \tag{47}$$

Let $\tilde{w}_i$ be the locally optimal width of S_i with respect to f, and w_i^* is the segment width of S_i in the optimal assignment f^*, the following inequality must hold:

$$\Psi(f, S_i) \; + \; \Theta(f, S_i) \cdot \tilde{w}_i \; + \; \Phi(f, S_i) \cdot \frac{1}{\tilde{w}_i} \;\; \leq \;\; \Psi(f, S_i) \; + \; \Theta(f, S_i) \cdot w_i^* \; + \; \Phi(f, S_i) \cdot \frac{1}{w_i^*} \tag{48}$$

Moreover, since w_i^* is locally optimal with respect to f^*, the following inequality must also hold:

$$\Psi(f^*, S_i) \; + \; \Theta(f^*, S_i) \cdot w_i^* \; + \; \Phi(f^*, S_i) \cdot \frac{1}{w_i^*} \;\; \leq \;\; \Psi(f^*, S_i) \; + \; \Theta(f^*, S_i) \cdot \tilde{w}_i \; + \; \Phi(f^*, S_i) \cdot \frac{1}{\tilde{w}_i} \tag{49}$$

After summing up Equation 48 and Equation 49, and grouping appropriate groups together, we will get the following inequalities:

$$\left\{ (\Theta(f^*, S_i) - \Theta(f, S_i)) \; + \; \frac{\Phi(f, S_i) - \Phi(f^*, S_i)}{\tilde{w}_i w_i^*} \right\} \cdot (w_i^* - \tilde{w}_i) \;\; \leq \;\; 0 \tag{50}$$

If f is dominated by f^*, we have $\Theta(f^*, S_i) \leq \Theta(f, S_i)$ and $\Phi(f, S_i) \leq \Phi(f^*, S_i)$. Therefore the factor $(w_i^* - \tilde{w}_i)$ in Equation 50 must be nonnegative, i.e. $w_i^* \geq \tilde{w}_i$. Therefore f' is also dominated by f^*.

If f dominates f^*, we have $\Theta(f^*, S_i) \geq \Theta(f, S_i)$ and $\Phi(f, S_i) \geq \Phi(f^*, S_i)$. Therefore the factor $(w_i^* - \tilde{w}_i)$ in Equation 50 must be nonpositive, i.e. $w_i^* \leq \tilde{w}_i$. Therefore f' also dominates f^*. □

Theorem 7 immediately suggests a strategy of using the GREWSA algorithm to compute the lower and upper bounds of each segment width of the optimal assignment. If we start with the initial assignment where each segment has the minimum wire width, the resulting assignment computed by the GREWSA algorithm gives a lower bound of the optimal width for each segment, since each intermediate assignment computed by the GREWSA algorithm, including the last one, is dominated by the optimal assignment. Similarly, if we start with an initial assignment where each segment has the maximum wire width, the resulting assignment computed by the GREWSA algorithm gives an upper bound of the optimal width for each segment.

These lower bounds and upper bounds of wire segments can further be translated into lower and upper bounds of the delay of the optimal assignment. Let $w_{i,lower}$ and $w_{i,upper}$ be the lower and upper bound of the wire width of segment S_i in T computed by the GREWSA algorithm, and f_{lower} and f_{upper} be the "lower bound assignment" and the "upper bound assignment" respectively (i.e. $f_{lower}(S_i) = w_{i,lower}$, and $f_{upper}(S_i) = w_{i,upper}$ for all segments S_i). We can obtain the following inequalities from Equation 10-13:

$$t_1(T) \;\; \geq \;\; R_d \cdot C_0 \cdot \sum_{k \epsilon T} l_k \cdot w_{k,lower} \tag{51}$$

$$t_2(T) \;\; \geq \;\; R_0 \cdot \sum_{all \;\; sinks \;\; k} C_k \cdot \sum_{i \epsilon P_k(T)} \frac{1}{w_{i,higher}} \tag{52}$$

$$t_3(T) \;\; \geq \;\; R_0 \cdot C_0 \cdot \sum_{k \epsilon T} \sum_{i \epsilon P_k(T)} \frac{w_{k,lower}}{w_{i,higher}} \tag{53}$$

$$t_4(T) \geq R_d \cdot \sum_{all\ sinks\ k} C_k \tag{54}$$

Therefore the sum of the right hand sides of the above inequalities gives a lower bound of the delay of the optimal assignment. Similarly, since both the lower bound assignment and the upper bound assignment computed by the GREWSA algorithm are realizable, the upper bound of the delay is given by $\min\{t(f_{lower}), t(f_{higher})\}$.

Given a routing tree with n segments, if we start with an assignment that is dominated by f^*, say the minimum width assignment, each iteration will generate a better assignment that remains dominated by f^*. Therefore, GREWSA will converge after at most $n \cdot (r-1)$ calls to *Function Greedy_Improvement*, each of which taking $O(n)$ time. By the same argument, GREWSA will also converge after at most $n \cdot (r-1)$ calls if the initial assignment dominates f^*. As a result, the worst case complexity of GREWSA is $\Omega(n^2 \cdot r)$ ($\Omega(n^2)$ if we consider r as a small constant).

4.3 The Combined Approach to the Wiresizing Problem

We have presented an $O(n^{r-1})$ time optimal algorithm, and an $O(n^2 \cdot r)$ time greedy algorithm with very good performance. It turns out that these two algorithms can be combined to a new algorithm which guarantees the optimal assignment and runs extremely fast. The combined algorithm, called GREWSA-OWSA, is described as follows:

First, we obtain the lower and upper bounds of each wire segment using the GREWSA algorithm. Then, we run a modified version of OWSA which only considers the assignments whose segment widths are consistent with the lower and upper bounds computed by the GREWSA algorithm. Since the lower and upper bounds obtained from the GREWSA algorithm are very close or even identical in most cases, the total number of candidate assignments ever generated by OWSA algorithm is much smaller than that by the OWSA algorithm alone. As a result, upper and lower bounds obtained help speed-up the algorithm significantly. For instance, GREWSA-OWSA runs 10 times faster than OWSA for the case of $n = 16$ and $r = 4$.

5 Experimental Result

We have implemented a generalized version of the A-tree algorithm and the wiresizing algorithm in ANSI C for the IBM-PC and Sun SPARC station environments. We compared the A-tree and wiresizing algorithms with other existing routing techniques on both MCM and advanced IC technologies. Section 5.1 – 5.3 show the improvement achieved by the A-tree algorithm, the wiresizing algorithms, and the A-tree + Wiresizing algorithms, and Section 5.4 shows the impact of resistance ratio and transistor sizing.

The results in Section 5.1 – 5.3 are based on a typical MCM technology [5] as shown in Table 4. We tested our algorithms on signal nets of 4, 8, and 16 sinks. For each net size, 100 nets were generated on a $100mm$ x 100 mm routing region for the MCM technology. The grid resolution is 25 μm per unit-grid-length.

5.1 Effect of Interconnect Topology Optimization

The generalized A-tree algorithm used in this subsection does not restrict the sinks to be in the first quadrant, and routing is performed for all quadrants simultaneously. The complexity of the general A-tree algorithm is the same as the first-quadrant version of the A-tree algorithm, and in all cases the runtime is no more than

Technology:	Multi-Chip Modules (MCMs)
Driver Resistance:	25 Ω
Unit Wire Resistance:	0.008 $\Omega/\mu m$
Loading Capacitance:	1000 fF
Unit Wire Capacitance:	0.060 $fF/\mu m$
Unit Wire Inductance:	380 $fH/\mu m$
Total Area:	100 mm x 100 mm (4000 grids x 4000 grids)

Table 4: Technology parameters based on advanced MCM designs.

0.3 seconds. For comparison purpose, we have also implemented the batched 1-Steiner algorithm proposed by Kahng and Robins [10], and the bounded-radius-bounded-cost (BRBC) algorithm proposed by Cong et al. [3]. Each of the three objective functions in Equation 8 and the average delay were compared in Table 5 for the A-tree, 1-Steiner, and BRBC-algorithms. The average delay of each net is obtained by averaging the signal delay at every sink using the two-pole circuit simulator developed by Zhou et al. [18]. Extensive experimental results have shown that the two-pole simulator is comparable to SPICE in terms of delay simulation, but runs much faster [18].

# sinks	Weight Function	A-tree	1-Steiner	BRBC-0.5	BRBC-1.0
4	$length(T)$	6.039 x 10^3	5.982 x 10^3 (-0.94%)	7.292 x 10^3 (+20.75%)	6.699 x 10^3 (+10.93%)
	$\sum_{k\epsilon N} l_k(T)$	1.068 x 10^4	1.129 x 10^4 (+5.71%)	1.145 x 10^4 (+7.21%)	1.133 x 10^4 (+6.09%)
	$\sum_{k\epsilon T} l_k(T)$	1.218 x 10^7	1.324 x 10^7 (+8.70%)	1.533 x 10^7 (+25.86%)	1.411 x 10^7 (+15.85%)
	Delay	8.07ns	9.10ns (+12.76%)	8.09ns (+0.25%)	7.88ns (-2.35%)
8	$length(T)$	9.174 x 10^3	8.759 x 10^3 (-4.52%)	11.447 x 10^3 (+24.78%)	10.441 x 10^3 (+13.81%)
	$\sum_{k\epsilon N} l_k(T)$	2.122 x 10^4	2.463 x 10^4 (+16.07%)	2.355 x 10^4 (+10.98%)	2.457 x 10^4 (+15.79%)
	$\sum_{k\epsilon T} l_k(T)$	1.892 x 10^7	2.291 x 10^7 (+21.09%)	2.608 x 10^7 (+37.84%)	2.521 x 10^7 (+33.25%)
	Delay	10.49ns	14.57ns (+38.89%)	11.85ns (+12.96%)	12.57ns (+19.83%)
16	$length(T)$	1.356 x 10^4	1.242 x 10^4 (-8.41%)	1.686 x 10^4 (+24.34%)	1.540 x 10^4 (+13.57%)
	$\sum_{k\epsilon N} l_k(T)$	4.347 x 10^4	5.666 x 10^4 (+30.34%)	5.296 x 10^4 (+21.83%)	5.663 x 10^4 (+30.27%)
	$\sum_{k\epsilon T} l_k(T)$	3.038 x 10^7	3.966 x 10^7 (+30.55%)	4.567 x 10^7 (+50.33%)	4.523 x 10^7 (+48.88%)
	Delay	14.92ns	26.14ns (+75.20%)	21.04ns (+41.02%)	23.31ns (+56.23%)

Table 5: Comparisons among the A-tree algorithm, the batched 1-Steiner algorithm, and the BRBC algorithms in terms of the three cost functions defined in the formulation of the $MDRT$ problem and the average delay under the MCM technology specified in Table 4. BRBC-0.5 and BRBC-1.0 are two parameterized versions of the BRBC algorithm with the control parameters ϵ chosen to be 0.5 and 1.0, respectively.

The batched 1-Steiner algorithm is one of the best known Steiner heuristics [10] and it is not surprising that the batched 1-Steiner algorithm outperforms the A-tree algorithm (by 1-8 %) in terms of wirelength, especially when the number of sinks is large. However, the A-tree algorithm significantly outperforms the batched 1-Steiner algorithm in terms of the objective functions $\sum_{k \epsilon N} pl_k(T)$ (which determines the $t_2(T)$ term) and $\sum_{k \epsilon T} pl_k(T)$ (which determines the $t_3(T)$ term), especially when the size of the signal net is large. The reduction in these terms offsets the wirelength advantage by the batched 1-Steiner algorithm. As a result, the A-tree algorithm reduces the average by up to 43 % as compared to the batched 1-Steiner algorithm. Also, the A-tree algorithm dominates the BRBC-0.5 and BRBC-1.0 algorithms in every term compared, and the A-tree algorithm reduces the average delay by up to 29 % and 36 %, respectively.

5.2 Effect of Wiresizing Optimization

We tested the OWSA, GREWSA, and GREWSA-OWSA algorithms on 100 16-sink routing trees generated by the A-tree algorithm. We compared the average delay, the runtime, and the average number of assignments examined by these algorithms. The results are summarized in Tables 6-7.

Number of Possible Wire Widths:	2	3	4	5	6
Average Delay					
No Wiresizing (ns):	18.7152	18.7152	18.7152	18.7152	18.7152
GREWSA starting from f_{lower} (ns):	13.0633	11.1765	10.2442	9.7072	9.3699
GREWSA starting from f_{upper} (ns):	13.0633	11.1765	10.2441	9.7074	9.3695
OWSA (ns):	13.0633	11.1760	10.2440	9.7070	9.3691
GREWSA-OWSA (ns):	13.0633	11.1760	10.2440	9.7070	9.3691
Average Run Time					
GREWSA starting from f_{lower} (s):	0.056	0.051	0.053	0.054	0.056
GREWSA starting from f_{upper} (s):	0.055	0.056	0.054	0.054	0.059
OWSA (s):	0.078	0.173	0.538	1.624	4.710
GREWSA-OWSA (s):	0.061	0.059	0.062	0.061	0.060

Table 6: Comparisons of the following wiresizing algorithms: GREWSA started with the minimum wire width assignment f_{lower}, GREWSA started with the maximum wire width assignment f_{upper}, OWSA, and GREWSA-OWSA. Widths used in this experiment are chosen from the set $\{W_1, 2 \cdot W_1, \cdots, r \cdot W_1\}$, where r is the number of possible widths and $W_1 = 15\mu m$.

These results show that the optimal wiresizing can further reduce average delay in routing trees by up to 50%. Moreover, we observed that the wire assignment solutions generated by GREWSA starting from f_{upper} and f_{lower} are both very close to optimal in practice. While OWSA has achieved a significant speed-up over the exhaustive enumeration methods, the lower and upper bounds of optimal wire widths computed by GREWSA can further reduce the number of assignments examined by OWSA significantly. In most cases, the lower and upper bounds of optimal wire widths computed by GREWSA uniquely determine the optimal assignments and OWSA is rarely invoked. Therefore, GREWSA-OWSA algorithm computes the optimal wire width assignments with far less computation time. In general, the runtime of OWSA is very sensitive to the parameter r, but the runtime of GREWSA-OWSA hardly changes as r increases.

Number of Possible Wire Widths:	2	3	4	5	6
Total Number of Assignment Examined					
Exhaustive Enumeration:	7.49×10^{9}	5.60×10^{15}	9.22×10^{19}	1.83×10^{23}	9.37×10^{25}
Exhaustive Enumeration (with MP):	1.75×10^{5}	7.67×10^{8}	5.27×10^{11}	1.16×10^{14}	1.16×10^{16}
OWSA:	3.35×10^{1}	2.02×10^{2}	8.53×10^{2}	3.00×10^{3}	9.34×10^{3}
GREWSA-OWSA:	1.00×10^{0}	1.27×10^{0}	2.13×10^{0}	2.51×10^{0}	2.30×10^{0}
Average Number of Choices per Segment					
OWSA:	2.0000	3.0000	4.0000	5.0000	6.0000
GREWSA-OWSA:	1.0000	1.0055	1.0129	1.0169	1.0151

Table 7: Comparison among the exhaustive enumeration methods (with and without considering the monotone property), OWSA, and GREWSA-OWSA in terms of the average number of assignments examined. The statistics are collected from the same set of test cases studied in the previous table, and the average number of segments is 32.53.

5.3 Effect of A-tree + Wiresizing

We also tested the combined A-tree + Wiresizing algorithm and compared it with the batched 1-Steiner algorithm and the BRBC algorithms (BRBC-0.5 and BRBC-1.0). The results of the comparison is summarized in Table 8.

# sinks	A-tree + Wiresizing	1-Steiner	BRBC-0.5	BRBC-1.0
4	5.27 *ns*	9.10 *ns* (+72.68%)	8.09 *ns* (+53.51%)	7.88 *ns* (+49.53%)
8	6.57 *ns*	14.57 *ns* (+121.77%)	11.85 *ns* (+80.37%)	12.57 *ns* (+91.32%)
16	8.94 *ns*	26.14 *ns* (+192.39%)	21.04 *ns* (+135.35%)	23.31 *ns* (+160.74%)

Table 8: Comparison of average delay under the MCM design technology. The percentages quoted is the increase in average delay of the routing tree as compared to the average delay in the wiresized A-tree.

It is clear that the combined algorithm significantly outperforms the batched 1-Steiner algorithm and the BRBC algorithms. Moreover, the performance advantage of the A-tree + Wiresizing algorithm over the batched 1-Steiner algorithm becomes more significant as the net size becomes larger. This is because the distributed nature of interconnect structures become more significant for large nets.

5.4 Effect of the Resistance Ratio and Transistor Sizing

Finally, we studied the impact of resistance ratio and transistor sizing on the improvement of the A-tree algorithm over the existing routing algorithms. In particular, we compared the A-tree algorithm and the batched 1-Steiner algorithm under four IC technologies, including the 2.0 μm, 1.5 μm, 1.2 μm, and 0.5 μm CMOS designs. The technology parameters are given in Table 9. The values of R_d and C_k are computed for the minimum-size transistors.

For each technology, we scaled the width of the driver transistor to 4, 6, 8, and 10 times its minimum width

(which is a commonly used technique to speed-up the critical nets), and obtained four different values of R_d, which led to four different resistance ratios for each of the technology. We generated 100 8-sink signal nets (uniformly distributed in a routing area of 0.5 mm x 0.5 mm) and routed them by the A-tree algorithm and the batched 1-Steiner algorithm, respectively. Figure 17 shows the improvement of wiresized A-trees over the batched 1-Steiner trees in terms of average delay for different technology and different transistor sizes.

Technology	2.0 μm CMOS	1.5 μm CMOS	1.2 μm CMOS	0.5 μm CMOS
R_d	2970 Ω	1430 Ω	1280 Ω	1560 Ω
Scaled R_d	297-743 Ω	143-357 Ω	128-321 Ω	156-390 Ω
R_0	0.0206 $\Omega/\mu m$	0.0150 $\Omega/\mu m$	0.0164 $\Omega/\mu m$	0.1120 $\Omega/\mu m$
C_k	5.175 fF	6.210 fF	4.416 fF	1.000 fF
C_0	0.0540 $fF/\mu m$	0.0042 $fF/\mu m$	0.0053 $fF/\mu m$	0.0391 $fF/\mu m$
$\frac{R_d}{R_0}$ (x $10^6 \mu m$)	0.144	0.095	0.078	0.014

Table 9: Technology parameters for 4 different IC technologies The 2.0 μm, 1.5 μm, and 1.2 μm CMOS technologies are provided by Robit Foresight Inc., and the 0.5 μm CMOS technology is provided by MCNC. The driver resistances (R_d) and loading capacitances (C_k) are derived for minimum-size transistors.

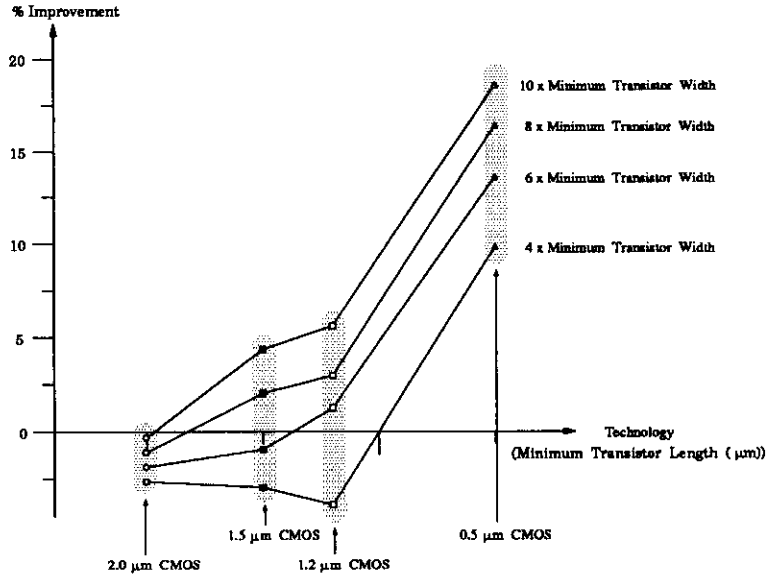


Figure 17: Performance improvement of the A-tree algorithm over the batched 1-Steiner algorithm in terms of delay as a function of the IC technology and the transistor size.

The results show clearly that given a fixed technology, the improvement of the A-tree approach over the classical Steiner approach becomes more significant when the width of the driver transistor is increased, which results in lower resistance ratio. Moreover, we have also observed a trend of increasing improvement by the A-tree algorithm as the device dimension decreases. For the conventional 2.0 μm CMOS technology, the A-tree algorithm in fact performs worse than the batched 1-Steiner algorithm since the total wire capacity is the dominating factor. However, for the advanced 0.5 μm CMOS technology, the A-tree algorithm consistently outperforms the batched 1-Steiner algorithm by a significant margin[3]. Since technological advances result in

[3] In general, the performance improvement obtained by the A-tree algorithm for IC technology is less than for MCM technology

smaller and smaller device dimensions, we expect that our A-tree algorithm will achieve even more significant improvement over the traditional Steiner tree approach.

6 Conclusion

In this paper we have studied the routing and wiresizing problems under a distributed RC delay model, and presented efficient solutions to the interconnect topology design and wiresizing problems for performance optimization. We have studied the impact of technology factors on the interconnect designs. We have formulated the performance-driven interconnect topology design problem as one of computing optimal A-trees and presented an efficient A-tree algorithm which achieves optimal solutions in most cases. We have proved several important properties of optimal wire width assignments, including the monotone property and the dominance property, and presented a polynomial time wiresizing algorithm GREWSA-OWSA. Extensive experimental results indicate that our approach significantly outperforms other routing methods for high-performance IC and MCM designs. Our methods reduce the interconnection delays by up to 66 % as compared to those by the best known Steiner tree algorithms.

We plan to further investigate the interconnect design problem in several directions. First, we shall extend the interconnect topology design problem to the case where the required time at each sink is not uniform (i.e. the sinks on the critical paths require a smaller delay and the sinks on the non-critical paths may have a longer delay). In this case, we can modify the A-tree algorithm by introducing "forbidden region" for each critical sink so that the critical sinks are connected directly or almost directed to the source. Also, we are interested to develop a simple yet accurate delay model for distributed RLC circuits (where the non-monotonic circuit response presents a great difficulty) so that such a model can be used effectively for interconnect design optimization when inductance is taken into consideration. Finally, we shall study the interconnect topology design and wiresizing problems under distributed RLC models so that we can consider the effect of reflection and cross-talk in the interconnect designs.

References

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

[2] C. Chiang, M. Sarrafzadeh, and C. K. Wong, "Global routing based on Steiner min-max trees", *IEEE Intl. Conf. on Computer-Aided Design*, 1989, pp. 2-5.

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

[4] J. Cong, A. B. Kahng, G. Robins, M. Sarrafzadeh, and C. K. Wong, "Provably good performance-driven global routing", *IEEE Trans. on CAD*, 11(6), June 1992, pp. 739-752.

[5] W. Dai, Private communication, 1992.

[6] W. E. Donath, R. J. Norman, B. K. Agrawal, and S. E. Bello, "Timing Driven Placement Using Complete Path Delays", *Proc. ACM/IEEE Design Automation Conf.*, 1990, pp. 84-89.

[7] A. E. Dunlop, V. D. Agrawal, D. N. Deutsh, M. F. Jukl, P. Kozak, and M. Wiesel, "Chip layout optimization using critical path weighting", *Proc. ACM/IEEE Design Automation Conf.*, 1990, pp. 133-136.

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

[9] A. L. Fisher, and H. T. Kung, "Synchronizing Large Systolic Arrays", *Proc. SPIE* 341, May 1982, pp. 44-52.

[10] A. B. Kahng, and G. Robins, "A New Class of Iterative Steiner Tree Heuristics with Good Performance", *IEEE Intl. Conf. on Computer-Aided Design*, July 1992, pp. 893-902.

[11] 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*, 1987, pp. 518-519.

[12] K. W. Lee, and C. Sechen, "A New Global Router for Row-Based Layout", *IEEE Intl. Conf. on Computer-Aided Design*, 1988, pp. 180-183.

[13] S. Prastjutrakul, and W. J. Kubitz, "A timing-driven global router for custom chip design", *IEEE Intl. Conf. on Computer-Aided Design*, 1990, pp. 48-51.

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

[15] J. Rubinstein, P. Penfield, and N. A. Horowitz, "Signal delay in RC tree networks", *IEEE Trans. on CAD*, 2(3) (1983) pp. 202-211.

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

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

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