Preface

I'd like to thank the following people for their invaluable help and assistance: Dr. Gabriel Robins and Mr. Bernard A. McCoy of the Department of Computer Science, and Dr. William Irwin and Dr. John Brown of the Division of Technology, Culture, and Communication.

Dr. Robins helped give me the necessary motivation to "not drop the ball" so that this project could lead to my first conference paper. Mr. McCoy gave me invaluable aid in "getting up to speed" quickly so that I could actually make significant contributions to the project. Dr. Irwin helped me realize the importance of making this work accessible to the layperson, and educated me on how difficult it can be for an academic to do so. Dr. Brown focused me on the relevance of the project, and helped me to recognize the importance of good diction, voice, and grammar.

Also, I'd like to thank Tallison Heather Rausch for her emotional support, and all my friends for their unflappable belief in me. Special thanks to G. Scott Evans and Scott Witherell for introducing me to the research environment here at the university in the department of computer science. They acted as good examples and gave me much early encouragement.

Table of Contents

List of Figures and Tables

I. Glossary of Terms

Key terms and theoretical concepts mentioned in the running text are summarized here. Similar or slightly abridged versions of these definitions exist where the terms first appear in the text.

1.1 Arborescence

Arborescences are a special type of shortest path *tree* with very little total edge length.

1.2 Delay / Propagation Delay

Given two pins on a chip, the propagation delay is the amount of time that lapses between the instant that the sending pin (or *source* pin) transmits a signal and the instant that the receiving pin (or *sink* pin) has accepted enough of the signal to understand it. More simply, *delay* is the time it takes for a signal to propagate from one node to another. The measurement is usually in the order of nanoseconds for modern computing technology, and is affected by many factors in silicon-based integrated circuits. Factors such as wire resistance, wire capacitance, and total wire length affect the delay value, and thus it can be quite hard to compute. Generally, the lower the delay, the better.

1.3 Elmore Delay Formula

A formula that gives relatively good estimates of the propagation delay in a circuit. The Elmore delay calculation requires very little time to run, and thus can be computed many timesquickly. See Appendix 2 for the actual Elmore formula and an analysis of the computational complexity ("speed") of the calculation. Elmore delay is not as accurate as the *SPICE finite-element* method, but has high *fidelity* with respect to it.

1.4 Fidelity

Fidelity is the trait of an estimation calculation that signifies whether comparisons made between the estimated data would hold true in reality. *High fidelity* implies that given two different estimations made by the same predictor, the one with the lower estimated value will also have the lower actual value, and contrariwise. This means that *comparisons* based on high-fidelity data should be correct, even if the actual estimates are imprecise. The *Elmore Delay* calculation has high fidelity with respect to both SPICE delay and real-world delay, and can thus be used to make comparisons reliably.

1.5 Finite Element Analysis

The process of hypothetically segmenting a physical object into many small pieces so that each element can be modeled with a simple differential equation, and then simultaneously solving them. This type of analysis can achieve extremely accurate estimations of the response the object will have to a stimuli. Commonly used to test the wind resistance of airplane wings and cars, and to predict the transfer of heat in various materials.

1.6 Manhattan Distance

A distance metric defined to be the sum of the differences between the x and y values, rather than the usual metric of taking the square root of the sum of the squares of the difference between x and y. Manhattan distances are used in circuit design because wires can only run horizontally or vertically, like the streets of downtown Manhattan. Thus,

$$\text{Euclidean distance} = \sqrt{(\Delta x)^2 + (\Delta y)^2}$$

$$\text{Manhattan distance} = \Delta x + \Delta y$$

1.7 Optimal Routing

The best possible routing. To find this best routing for a given pointset, time exponential in the size of the instance is required, which is a very long time. Because this is computationally prohibitive (it takes too long to find), routing heuristics like those described here and in the *Review of Relevant Literature* are necessary for most nontrivial cases. Performance results are often bounded by some factor worse than optimal, such as “twice worse than optimal” or “within 10% of optimal.”

1.8 Routing

The process of finding a way to connect a group of pins on an integrated circuit. The phase of VLSI design that occurs before the chips are actually fabricated, but after the decision as to which pins must be connected to each other has been made.

1.9 Sink

A pin that receives input from another pin. The *destination* of an electrical pulse, in contrast to the *source* of that pulse.

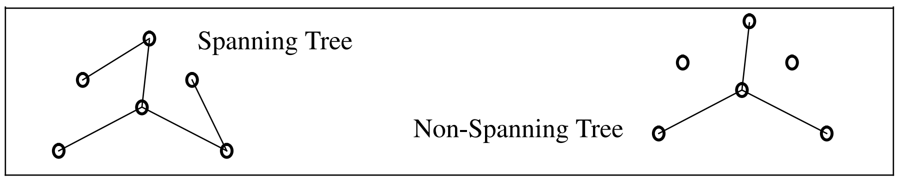
1.10 Source

As in *Source Pin, Source Point,* or *Source Node*. A pin from which electrons originate, rather than terminate. Signals from the *source* propagate (with various delays) to one or more sinks.

1.11 Spanning Tree

A *tree* that includes all the points in the pointset (graph) under consideration.

FIGURE I. Spanning tree versus non-spanning tree example



The spanning tree includes all the points in the pointset

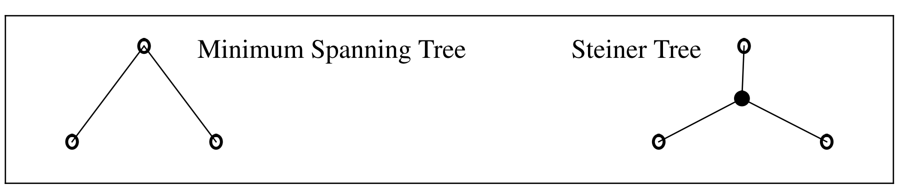
1.12 SPICE Delay Simulator

A program that computes a delay value using *finite element analysis* methods. The de facto standard for high-accuracy delay calculations. It would be used instead of the *Elmore delay formula* to develop routings, but it is much too slow. We use SPICE to test the circuits generated by the various algorithms in order to facilitate very accurate comparisons.

1.13 Steiner Tree

A spanning tree over a set of vertices in which extra points (called Steiner points) have been added in order to reduce the total edge length. Proven to be a very good minimum spanning construction. Steiner points are added by routing algorithms in order to generate better topologies. For example, in the diagram below, if the total edge length of the tree at left was 1, adding the Steiner point (black dot) gives a total length of about $^3/_4$ for the tree on the right.

FIGURE II. Comparison of minimum spanning tree and Steiner tree



The Steiner Tree has an added point (in black) which reduces total edge length

1.14 Technology Parameters

The size of the components on a chip or chips, which varies widely. For each set of technology parameters, separate tests are necessary, because the routings can change drastically. My project will concentrate on state-of-the-art Multi-Chip Module (MCM) technology, with parameters supplied by the Microelectronics division of AT&T. The MCM technology is prone to *topological degeneracies* due to the large distances involved. Typical parameters include such attributes as wire resistance per unit volume, wire capacitance per unit volume, inductance per unit volume, and average distance between pins.

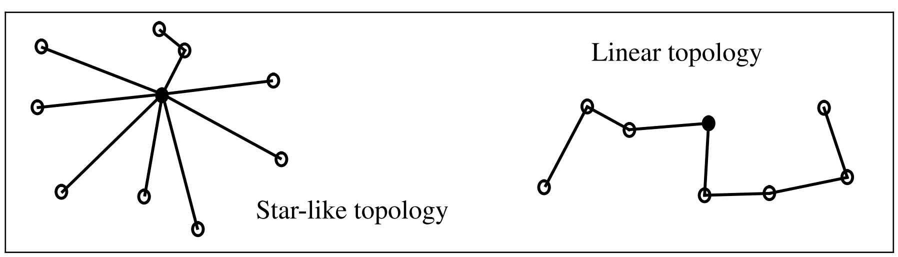
1.15 Topology

The geometric configuration of a particular layout. The general picture of what a layout looks like. Typical easy-to-recognize topologies may be "star-like" or "linear" ones. See Figure 3 for two example topologies.

1.16 Topological Degeneracy

A "uninteresting" topology, such as a star or straight line. Topologically degenerate routings are usually a poor choice because they either exhibit high maximum delay or cannot be practically implemented. Our algorithm typically avoids these degeneracies. See Figure 3 for two examples of topological degeneracy.

FIGURE III. Two types of topological degeneracy

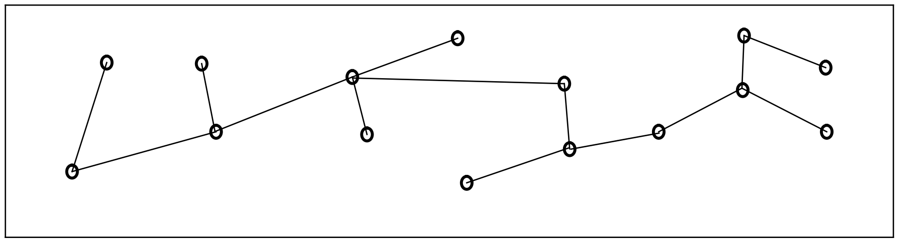


The topologies are usually either too expensive to implement or too slow

1.17 Tree

A spanning graph with no cycles. A set of vertices and edges such that there is exactly one path from any point to any other. A set of vertices and edges such that there is one less edge than there are vertices, but where the edges span the entire set of vertices.

FIGURE IV. Example of a tree in a random pointset



The tree is a spanning tree because all points are connected

1.18 VLSI

Very Large Scale Integration. The process that allows millions of transistors to be packaged on a single microchip. Although routing can be applied to many problems, the algorithms we consider are specifically tuned for aiding the design of VLSI circuits.

1.19 Wiresizing

The process of allowing a routing's edges to have width in addition to length in an attempt to decrease maximum delay. Contrast this with the current standard that all wire widths remain exactly equal. The concept of wiresizing is analogous to having the ability to build roads with multiple lanes rather than only one.

II. Abstract

Much current research is directed toward increasing the clock speed of integrated circuits, especially for use in high-performance computer equipment. The *clock speed* of a circuit is a measure of how fast the circuit can perform. Increasing a processor's clock speed is an effective way to increase the speed of a computer using that processor. The *routing* of the circuit accounts for a large portion of its clock speed. Thus, generating good routings produces faster computers. Detailed herein is an algorithm that produces extremely high-performance chip routings. At the heart of this algorithm is a new idea called *wiresizing*, which is a technique that involves allowing the widths of the wires in the circuit to vary.

Routing is the process of determining which pins on an integrated circuit to connect with wire, given various constraints. Current routing constructions generate fixed topologies. These can then have their performance augmented by increasing certain wire widths in the topology. We show that this post-processing augmentation, or *static wiresizing,* technique can increase the performance of current routings by as much as 38%. We then extend this scheme and introduce a *dynamically-wiresized* routing algorithm. Instead of first starting with a fixed topology and then increasing wire widths in two discrete steps, the new algorithm accounts for wire width *during construction.* Our dynamically-wiresized algorithm averages as much as 9.9% better performance than the best-performing statically-wiresized techniques, which is up to 72% better than traditional near-optimal Steiner routing.

In addition to simple performance gains, dynamic wiresizing also avoids *topological degeneracy,* a problem common in optimal routings. Topological degeneracy is an "uninteresting" topology, such as a star or straight line. Topologically degenerate routings are usually a poor choice because they either exhibit high maximum delay or cannot be practically implemented. Our algorithm typically avoids these degeneracies, at no additional cost, where other similar algorithms do not.

A Routing Algorithm to Aid in the Design of High-Performance VLSI Circuits

Todd D. Hodes

Undergraduate Thesis
Department of Computer Science
University of Virginia

1. Introduction

1.1 Purpose

This section introduces my undergraduate thesis project and explains the problem it solves. The references to "we" in the document refer to myself and two other members of the computer science department, Dr. Gabriel Robins and Bernard McCoy. Here I will explain the importance of the routing problem and exhibit the rationale behind our solution.

1.2 Summary

The product of my thesis project is an algorithm that produces *routings* for *VLSI circuits*. *Routing* is the process of finding a way to connect a group of pins on an integrated circuit. *VLSI circuits* are the circuits common in everyday electronic devices ranging from toasters to microcomputers. *VLSI* stands for Very Large Scale Integration. VLSI is the method used to pack millions of transistors into spaces on the order of a few square millimeters. *Routing algorithms* (or routers) allow a computer to aid in the design of these VLSI routings in two ways. Firstly, rather than require an human to design them, with the inherent possibility of error, the algorithm always outputs a faultless routing. Secondly, the algorithm automatically generates routings guaranteeing excellent performance characteristics. Thus, the product of this thesis project completes a procedure usually performed by humans automatically, while increasing speed, accuracy, and performance.

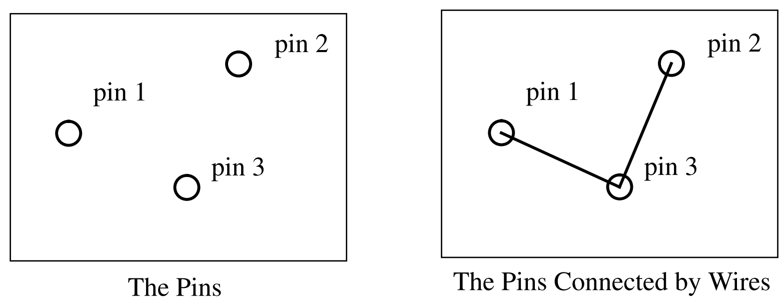
1.3 Problem Relevance

The *routing* of a chip largely determines the maximum clock speed of that chip. Increasing the clock speed of a chip such as a computer's processor will make the computer faster. Thus, much current research is directed toward increasing the clock speed of integrated circuits for use in high-performance computer equipment. Since routing is a major factor in determining the clock speed of integrated circuits, one possible way to increase this speed is to make the chip's routing better. Detailed herein is a new heuristic algorithm that produces chip routings better than the current state-of-the-art. The algorithm can be used to increase the clock speeds of both existing and forthcoming integrated circuits. By incorporating a new concept called *wiresizing* into the algorithm, results better than the finest previous routers were achieved. This new technique therefore can be utilized to increase the clock speed of circuits. This is an worthwhile goal because increasing computing power and speed allows members of society to solve problems that are currently too computationally prohibitive to be attempted.

1.4 Problem Definition

In order to solve the routing problem, we must abstract it away from the actual physical world it inhabits into the mathematical one. We abstract the pins of the integrated circuit which need to be connected into points in a plane, and the wires that would connect the pins into edges between the points. Thus, a circuit to be routed becomes a simple graph. In this abstracted sense, routing is the act of creating a graph over the set of points (pins) such that all the points are somehow connected by edges (wires). For example, given a three-pin integrated circuit to route, the abstracted problem, along with a possible routing, might look like Figure 1. Once the problem has been abstracted to a graph, we then note that graph theory can be used for a basis of analysis.

FIGURE 1. Abstracted routing problem for a three-pin integrated circuit



The abstraction is necessary in order to take the problem from the real world to a mathematical one

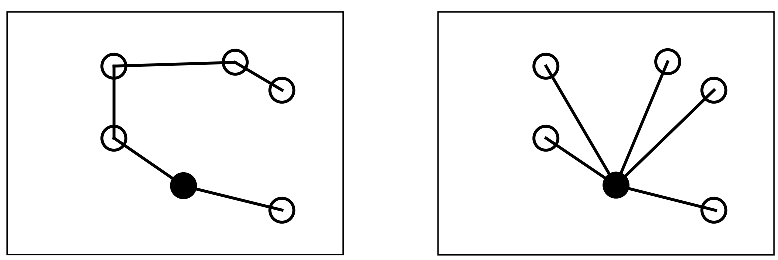
Routing a net is not done arbitrarily, of course. For our purposes, there is some main pin called the *source* from which a signal (an electron pulse) propagates, and other pins, called *sinks*, that receive this signal. The basis of "good" routing is to get the signal from the source to all the sinks in the least possible amount of time. In other words, a good routing minimizes the maximum *propagation delay* (sometimes simply called *delay*). The *propagation delay* is the amount of time that lapses between the instant that a sending pin (or *source* pin) transmits a signal and the instant that a receiving pin (or *sink* pin) has accepted enough of the signal to understand it. For our simple example above, if pin 1 were the source and pins 2 and 3 the sinks, the signal would travel first to pin 3 and then to pin 2, and thus the maximum delay would be to pin 2. Various methods of solving the routing problem in this abstract sense already exist. Descriptions of the best preforming current algorithms are presented in the Review of Relevant Literature, while well-known, more general methods are introduced in the Routing Overview, below.

1.5 Routing Overview

Certain algorithms for connecting a set of points are considered fundamental to graph theory, and therefore routing. Example include both Prim's algorithm for computing a

Minimum Spanning Tree and Dijkstra's algorithm for computing a Shortest Path Tree. Both of these algorithms create *spanning trees* (trees that include all the points in the graph under consideration) that maintain certain desirable characteristics. Since these algorithms are often at the core of more complex routers and are the basis of many intuitive routing solutions, they will be introduced before describing the routers in the literature. As mentioned, one of these fundamental spanning trees is the *shortest path tree* (SPT). A SPT is a spanning tree with the property that the maximum total edge length from a designated point (the *source*) to any other point (the *sinks*) is minimized. Another fundamental spanning tree is the *minimum spanning tree* (MST). The MST is a spanning tree with the property that the sum of the total edge length is minimized. These two simple geometric constructs form the basis for the router in Cong, et. al. (1993) [5]. Better routers, such as that in Barrera, et. al. [2], use more complex formulas to create the routing trees than these simple geometric constructs. See Figure 2 for examples of an MST and SPT over the same sample pointset.

FIGURE 2. Examples of a Minimum Spanning Tree and a Shortest Path Tree



A Minimum Spanning Tree A Shortest Path Tree

Note that the MST has less total *edge length, but that the SPT has less edge length between the source (in black) and any sink*

1.6 Solution Methodology

The key idea behind our new algorithm is rather intuitive; it is an idea that becomes obvious once explained, and is a natural and logical extension of existing routing methods. This new idea is the concept of *wiresizing*. *Wiresizing* is a term used to describe the process of allowing certain wires in an integrated circuit to be of different thicknesses. In practice, it can be implemented by either actually allowing the wire widths to vary, or by simply using multiple wires between two pins where necessary. Wiresizing betters circuit routings by reducing the maximum delay in the circuit. Why wiresizing works is more evident if we compare routing to a more natural occurrence. Say, for instance, that we are trying to supply water from a single feeder pipe to many houses. If each pipe leading off of the source pipe was of the same width, we would expect each house to get an equal amount of water pressure. But if one pipe lead to an apartment complex, we would need to *thicken* this pipe so that each apartment would receive the same water pressure the average house obtains. This thickening or thinning of pipes is analogous to the process of thickening or thinning wires–certain places in the net need more electrons flowing to them than others. Another simple analogy can be obtained by thinking of a living tree. It has thin branches where less water and nutrients need to travel, and a thick trunk where lots of water and nutrients must pass.

One might ask why wiresizing has not been adopted earlier, given its intuitive advantages. There are many reasons. Firstly, the technology has yet to be proven. Wiresizing is so young that it is still in its formative stages, and thus it will not be incorporated into current technologies until it has been sufficiently tested. Additionally, it is not yet obvious that the performance increase will outweigh the increase in cost. The economic issues involved warrant further study.

Given this concept of variable widths, we added wiresizing to current routing techniques from the literature and created our own technique that is optimized for the wiresizing. In addition to reducing overall maximum source-sink delay (making the routing better), as was expected, wiresizing also reduces the amount of *topological degeneracy* that currently occurs in many optimal solutions. A *topologically degenerate* solution is a solution that is “uninteresting,” such as simply connecting each sink to the source directly or connecting all the sinks in a line. Topologically degenerate routings are usually a poor choice because they either exhibit high maximum delay or cannot be practically implemented. To go back to the water pipe analogy, imagine that water pressure is required to be extremely high at each house. One solution is to run a separate pipe directly from the water works to each house. This common topological degeneracy (called a "star-like" topology for obvious reasons) would certainly maximize water pressure, but is clearly unrealistic because of the sheer length of pipe that would be required. In terms of routing, it is usually physically impossible to attach a large number of wires to a single source pin, although the design would certainly give excellent performance. The obvious solution is the one actually used with water pipes--use a few pipes of increased width near the source so much water can flow through them, then branch smaller pipes off of them where necessary. Thus, we discovered that utilizing wiresizing not only gives better routings, but also inherently avoids topological degeneracy.

1.7 Objectives Met

A sample of our objectives accomplished.

- added static wiresizing to previously existing unwiresized routing schemes
- developed of a novel wiresized algorithm to achieve better routings
- implementing these algorithms on research machines in the department of Computer Science, using the C programming language and SunOS Unix operating system
- produced and collected extensive data on the performance of these algorithms using SPICE, a finite-element circuit simulation package
- collected data comparing unwiresized algorithms to their *wiresized* equivalents
- produced a undergraduate thesis project to submit for degree candidacy
- incrementally improved the discipline of VLSI CAD Routing
- published and presented a document and talk based on this work at the 1994 International Symposium on Circuits And Systems (ISCAS '94) in London, England. A copy of the conference paper is available in Appendix 4.

1.8 Overview of Remaining Contents

The remaining sections of the report include the following:

- a review of the relevant literature to date in Chapter 2
- an explanation of current routing algorithms and their wiresized cousins in Chapter 3
- a discussion of wiresizing in Chapter 4

- a presentation of our new *dynamically wiresized* routing construction, the Dynamically-Wiresized Steiner Elmore Routing Tree, also in Chapter 4
- the results of head-to-head testing of the algorithms in Chapter 5
- conclusions in Chapter 6

2. Review of Relevant Literature

2.1 Introduction

The purpose of the literature search is to gain a working knowledge of general routing information, the historical context, specific routing techniques, and the theoretical grounding. Herein is an overview of the routing work that preceded our own. The particular constructions that we compared to our new algorithm and to each other are introduced here but will be explained in more detail in separate sections of Chapter 3, Existing Routing Algorithms.

2.2 Field Overview

Interconnect delay has recently become a dominant concern in the design of complex, high-performance circuits, due to the *scaling* of *VLSI technology.* Although the distance between features keep shrinking rapidly, interconnect delays are not. Thus, interconnect delay is quickly becoming the dominate portion of the clock cycle, creating a bottleneck in improving performance. Performance-driven layout design has thus become an active area of research over the past several years, where for a given signal net, the typical goal of performance-driven routing is to minimize average or maximum source-sink delay, as described in Kahng and Robins [8]. Much early work implicitly equates *optimal routing* with minimum-cost *Steiner* routing. A Steiner routing is a routing where additional points can be added to the pointset, changing the *topology* and increasing performance. For example, Prastjutrakul and Kubitz [9] have given approaches which are tuned to building-block layout and allow prescribed upper bounds on individual source-sink delays. Recently it became increasingly apparent that for leading-edge technologies, delay minimization and wire length minimization are far from synonymous. For example, Cohoon and Randall [3] proposed a heuristic which simultaneously considered both the *cost* (total edge length) and the *radius* (longest source-sink path length) of the routing tree. A more general formulation was given by Cong, et. al. in [4], wherein a parameter guides the trade-off between cost and radius minimization in a “provably good” BRBC (bounded-radius, bounded-cost) algorithm that simultaneously maintains both cost and radius within constant factors of optimal. The cost-radius trade-off may also be viewed as one between competing minimum spanning tree (MST) (or minimum-cost Steiner tree) and shortest path tree (SPT) constructions. Using this perspective, Alpert, et. al. [1] recently proposed the AHHK algorithm, which achieves a direct MST-SPT trade-off. Finally, Cong, et. al. [5] have recently proposed the use of rectilinear Steiner arborescences, or A-trees, which are essentially minimum-cost SPTs with Steiner points allowed. Other recent Steiner algorithms include the Iterated One-Steiner (I1S) of Kahng and Robins [8] and the Steiner Elmore Routing Tree (SERT) of Cong, et. al. [4]. The algorithms that we implemented, tested, and compared (in addition to our own) include the A-Tree algorithm, the Iterated One-Steiner algorithm, the Steiner Elmore Routing Tree algorithm, and the *statically wiresized* version of each of the previous three. The algorithms will be formalized in the section on existing routing algorithms (Chapter 3), and a discussion of static wiresizing occurs in the section on wiresizing (Chapter 4).

3. Existing Routing Algorithms

3.1 Introduction

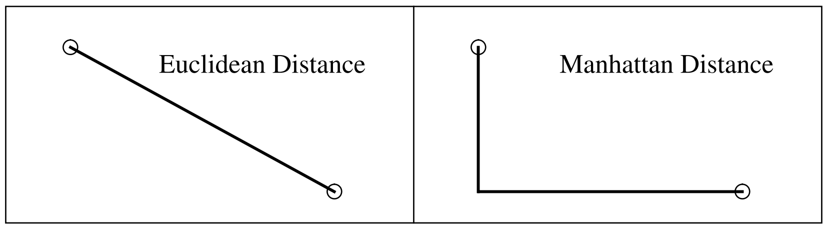
This section describes the best routing algorithms in the literature in detail. These algorithms, and their wiresized cousins, were compared to each other and our new construction in order to gauge both the effectiveness of wiresizing and the efficacy of our algorithm.

3.2 Definitions and Notation

First of all, let us define our terms. All distances involved in routing are assumed to be *Manhattan distances*, rather than the usual *Euclidean distances* measured by the Pythagorean theorem. Manhattan distance is measured by summing the differences between the x and y values directly, rather than by summing their squares and then taking the square root. See Figure 3 for a graphical depiction of the difference.

Additionally, the *Manhattan plane* is defined to be a plane where distances are measured using the Manhattan distance formula rather than the Euclidean distance formula. All routing takes place in the Manhattan plane. Note that this is not an arbitrary decision. Wires can only run in the x or y direction on circuit boards, and thus Manhattan distances are the actual ones encountered in practice.

FIGURE 3. Distance metrics



$$\text{Euclidean distance} = \sqrt{(\Delta x)^2 + (\Delta y)^2}$$

$$\text{Manhattan distance} = \Delta x + \Delta y$$

Next, define a *signal net* $N = \{n_0, n_1, \ldots, n_k\}$ as a fixed set of pins in the Manhattan plane to be connected by a *routing tree* $T = (N, E)$, where $E \subseteq N \times N$. Pin $n_0 \in N$ is the *source* (the point where the signal originates), and the remaining pins are *sinks* (points where the signal propagates to). Each edge $e_{ij} \in E$ has an associated edge cost, d_{ij}, equal to the Manhattan distance between its two endpoints n_i and n_j. The *cost* of T is the sum of its edge costs. $t(n_i)$ is used to denote the *signal propagation delay* from the source to pin n_i. The goal of a routing algorithm is to construct a spanning tree over the net that minimizes the maximum source-sink delay, or in other words minimizes the largest $t(n_i)$ value. This problem is formalized as the Optimal Routing Tree (ORT) Problem, or the Optimal Steiner Routing Tree (OSRT) Problem if Steiner points are allowed:

Optimal Steiner Routing Problem:

Given a signal net $N = \{n_0, n_1, \ldots, n_k\}$ with source n_0, find a set of points S and construct a routing tree $T = (N \cup S, E)$ with $E \subseteq (N \cup S) \times (N \cup S)$ such that $t(T) = max_{\forall i}\, t(n_i)$ is minimized.

Finally, define $MST(N)$ to be a minimum spanning tree over the set of pins N, and $\Delta MST(N, S)$ to be the change in cost that occurs when adding the Steiner points in S. Thus, $\Delta MST(N, S) = cost(MST(N)) - cost(MST(N \cup S))$.

3.3 Iterated One Steiner (I1S)

The first algorithm explained here is the Iterated One Steiner (I1S) algorithm of Kahng and Robins [8]. The I1S algorithm is one of the best performing purely geometric algorithms, producing near-optimal Steiner trees. It does not take into account the actual delay between two points when adding edges, but instead assumes that minimizing total wire length will lead to efficient routings. The the I1S algorithm returns solutions on average less than 0.25% away from the optimal Steiner tree solution and produces optimal Steiner solutions 90% of the time, as reported in Barrera, et al [2]. Unfortunately, it is still hindered by the assumption that propagation delay will be accurately predicted by geometry. This had been shown to be an invalid assumption because propagation delay is a non-linear phenomenon, as was reported in Elmore [7], meaning that “simple” (linear) predictors like total wire length or longest source-sink path length can’t be used to predict actual performance. The I1S algorithm's performance numbers are used as normalizing the results in the data section. The I1S algorithm is defined in Table 1.

TABLE 1. The I1S algorithm

1.	**Input** N = a set of points to be routed with source n_0
2.	$S = 0$
3.	**While** (Steiner points can be added to reduce the total cost of the tree)
4.	T = the set of Steiner points that can reduce the total cost of the tree
5.	**Find** $x \in T$ with maximum $\Delta MST(N \cup S, \{x\})$
6.	$S = S \cup \{x\}$
7.	**Remove** any Steiner points that are redundant
8.	**Output** the resulting tree $MST(N \cup S)$

3.4 Steiner Elmore Routing Tree (SERT)

The second algorithm explained here is the Steiner Elmore routing tree algorithm of Cong, et. al. (1992) [4]. Again, since delay is a non-linear phenomenon, knowing path length information alone is not enough to predict actual performance. Many previous routers (such as I1S) have ignored this fact and have either implicitly or explicitly assumed that delay is proportional to source-sink path length. Thus, such methods only attempt to *heuristically* capture the goal of "high-performance." It is therefore not surprising that when trees produced by these methods are tested by simulation, their performances often prove disappointing. The SERT construction avoids the level of abstraction inherent in such previous objectives as "minimum cost," "shortest path," or "bounded radius," and instead *directly* optimizes *Elmore delay* during tree construction. *Elmore delay* is a non-linear predictor that can be calculated rather quickly.

The SERT algorithm is analogous to Prim's minimum spanning tree construction, and works as follows. Staring with the trivial tree consisting of only the source pin, we grow the tree at each step by finding a new pin to connect to the tree such that the maximum *Elmore delay* to any leaf is minimized. But rather than requiring that the new pin connect directly to a pin already in the tree, we allow the new pin to connect to any tree edge, thus inducing a *Steiner point*. The algorithm terminates when the resulting Steiner tree spans the entire net. The SERT algorithm is defined more formally in Table 2.

TABLE 2. The SERT algorithm

1.	**Input** N = a set of points to be routed with source n_0
2.	$T = (V, E) = (\{n_0\}, 0)$
3.	$M = N - \{n_0\}$
4.	**While** $(M \neq 0)$ [there are still pins to be attached]
5.	**Find** $u \in M$, $(v, v') \in E$, and a new point w which minimizes the maximum Elmore delay from n_0 to any leaf in the new tree formed†
6.	$V = V \cup \{u, w\}$
7.	$E = (E - \{(v, v')\}) \cup \{(v, w), (w, v')\}$
8.	$M = M - \{u\}$
9.	**Output** the resulting Steiner tree $T = (V, E)$

† where the new tree formed is $(V \cup \{u, w\}, (E - \{(v, v')\}) \cup \{(v, w), (w, v')\})$

3.5 A-Tree

The final algorithm explained here is the A-Tree algorithm of Cong, et. al. (1993) [5]. The A-Tree can be a preferred interconnect topology because it minimizes total wire length while maintaining shortest paths between the source and every sink. Although this is a purely geometric construction, it performs excellently, in many cases better than even the delay-based SERT. An A-Tree is based on rectilinear (Manhattan) Steiner *arborescences*, as described in Rao, et. al. [10]. *Arborescences* are a special type of shortest path tree with very little total wire. The algorithm begins with a forest of n single-node arborescences and proceeds by applying a sequence of either “optimal” or “heuristic” moves. The moves either extend an existing arborescence or combine two arborescences. The process terminates when only one arborescence remains, and this spanning tree is output. The A-Tree generating algorithm is formalized in Table 3.

TABLE 3. The A-Tree algorithm

1.	**Input** N = a set of points to be routed with source n_0
2.	**While** (There is more than one arborescence)
3.	**If** an optimal move exists
4.	**Then** perform an optimal move
5.	**Else** perform a heuristic move
6.	**Output** the resulting Steiner tree

3.6 Summary

The Algorithms presented above are all excellent high-performance routing techniques. The I1S is interesting in that it produces optimal or near-optimal Steiner trees, which are intriguing in their own right. Unfortunately, *Steiner* trees are less-than-optimal *routing* trees. The A-Tree is a non-Steiner approach, and achieves much better routings, indicating that SPTs with close to MST wire length are very good routing trees. The SERT algorithm takes a more markedly dissimilar approach by using a non-linear estimator as its basis, and also achieves better-than-Steiner routings, but it is limited by some topological degeneracy.

4. Wiresizing

4.1 Introduction

Wiresizing is the process of allowing a routing's edges to have width in addition to length in an attempt to decrease maximum delay. It can improve signal propagation delay by trading increased capacitance for decreased resistance. When a wire width is increased, additional capacitance in induced, but overall source-sink resistance may decrease. Generally, the lower the resistance goes the lower the delay falls. Thus, the idea behind successful wiresizing is to find wire segments in the routing where an increase in capacitance is more than compensated for by the corresponding decrease in resistance, thus improving the maximum source-sink routing signal delay. More formally, if we define $w(e_i)$ as the width of edge e_i and allow the range of edge widths to be $\{w_1, w_2, \ldots, w_j\}$ with $w_1 < w_2 < \ldots < w_j$, then we can extend the problem of finding the Optimal Steiner Routing Tree to the problem of finding the Optimal Wiresized Steiner Routing Tree (OWSRT) as follows:

Optimal Wiresized Steiner Routing Problem:

Given a signal net $N = \{n_0, n_1, \ldots, n_k\}$ with source n_0, find a set of points S and construct a routing tree $T = (N \cup S, E)$ with edges $E \subseteq (N \cup S) \times (N \cup S)$ of varying width $w(e_i)$ such that $t(T) = max_{\forall i}\ t(n_i)$ is minimized.

4.2 Static Wiresizing

Static wiresizing is a procedure to find the optimal size of each wire in a net *given a fixed topology.* These fixed topologies can come from sources such as the routing algorithms discussed in Section 3. The static wiresizing scheme of Cong, et. al. (1993) [5] optimally wiresizes by recursively wiresizing each subtree of the source. As long as overall maximum tree delay improvement is possible, edges connecting the root to a subtree are widened. Our static wiresizing scheme generalizes the scheme of Cong, et. al. (1993) [5] in that allows *any* arbitrary delay calculation to be used to drive the wiresizing. The salient issue here is that this method is *static*, meaning that the topology of the tree is determined before wiresizing commences and does not change during the wiresizing process. The generalized Static Wiresizing (SW) scheme is formalized in Table 4.

TABLE 4. **The Static Wiresizing (SW) algorithm**

1.	**Input** T = (V, E), a tree to be wiresized
2.	**For** each node $n_i \in V$ such that $e = (n_0, n_i) \in E$
3.	**Recursively Call** SW on the subtree routed at n_i
4.	**Repeat**
5.	$delay_{old} = t(T)$
6.	**Increase** w_r to w_{r+1} of edge e
7.	**Until** $delay_{old} < t(T)$
8.	**Decrease** w_r to w_{r-1} of edge e
9.	**Output** the resulting wiresized tree T

4.3 Dynamic Wiresizing

While static wiresizing provides a near-optimal wiresizing for a *given* topology, as described in Cong, et. al. (1993) [5], the wiresizing process is largely constrained to the fixed input topology. We do not want to completely dissociate the topology construction from the wiresizing phase (as was implicit in Cong, et. al. (1993) [5]), since this strategy will not benefit from a possible synergy between the two issues. Ideally we would like to compute the *optimal* combination of routing topology *and* wiresizing; unfortunately, this not computationally feasible. With this in mind, we have developed a *dynamic* wiresizing algorithm that hybridizes the routing topology construction with the wiresizing process. Our new construction combines the Elmore routing tree method of Section 3.4 with the static wiresizing of Section 4.2. The overall structure of this new algorithm, the Dynamically Wiresized Steiner Elmore routing tree (DWSERT) construction, is similar to that of SERT, except that when we select a new edge to add to the growing topology, rather than minimize the Elmore delay in the resulting topology, we instead minimize the Elmore delay in the *wiresized* current topology. In other words, in each step of the SERT construction we invoke the static wiresizing routine once for each candidate edge, and select to add the edge that yields the best wiresized tree.

Note, however, that during the execution of DWSERT, a partial topology is never *actually* wiresized, but is instead left unwiresized. Rather than wiresizing during the construction, wiresizing considerations are used as a *guide* to drive the edge-selection process. Once the topology spans all the pins in the net, we invoke the static wiresizing algorithm one final time and return the resulting *wiresized* tree. The DWSERT algorithm is formalized in Table 5.

TABLE 5. The DWSERT algorithm

1.	**Input** N = a set of points to be routed with source n_0
2.	$T = (V, E) = (\{n_0\}, 0)$
3.	$M = N - \{n_0\}$
4.	**While** ($M \neq 0$) [there are still pins to be attached]
5.	**Find** $u \in M$, $(v, v') \in E$, and a new point w which minimizes the maximum Elmore delay from n_0 to any leaf in the *wiresized* tree formed†
6.	$V = V \cup \{u, w\}$
7.	$E = (E - \{(v, v')\}) \cup \{(v, w), (w, v')\}$
8.	$M = M - \{u\}$
9.	**Output** the resulting wiresized Steiner tree $SW(T = (V, E))$

† where the new tree formed is SW($V \cup \{u, w\}, (E - \{(v, v')\}) \cup \{(v, w), (w, v')\}$)

5. Experimental Results

We have implemented the I1S algorithm, the SERT algorithm, the A-Tree algorithm, and their statically wiresized versions (called WI1S, WSERT, and WA-Tree, respectively), along with our dynamically wiresized construction (DWSERT) on the research machines in the department of Computer Science, using the C programming language and SunOS Unix operating system. We tested these algorithms on two hundred fifty random nets with five to twenty-five pins uniformly distributed in a $100{,}000\mu \times 100{,}000\mu$ grid. (μ is the symbol for a micron, a unit equal to 10^{-6} meters.) One of the pins is chosen at random to be the source, and the rest act as sinks. Our *technology parameters*, such as grid size, correspond to a typical multichip module (MCM) interconnect parameters, provided by the Microelectronics division of AT&T. *Technology parameters* are a set of statistics that exactly specify the type of microchips that are being designed. They vary widely based on product, company, and purpose. Our technology parameters are summarized in Table 6.

TABLE 6. MCM technology parameters

Parameter	**Value**
driver resistance	25 Ω
wire resistance	0.008 Ω / μm
wire capacitance	0.06 *fF* / μm
wire inductance	380 *f*H / μm
sink loading capacitance	1000 *fF*
layout area	10^4 mm^2

Table 7 gives the average percent SPICE delay improvement in maximum source-sink delay relative to the corresponding I1S value. In other words, each entry in the table represents the average percent improvement in maximum delay as compared to the maximum delay for the I1S routing over the same nets. Since I1S produces near-optimal Steiner routings, it was chosen as the reference point and normalizing algorithm.

TABLE 7. Average percent maximum SPICE delay decrease over I1S

net size =	**5**	**10**	**15**	**20**	**25**
I1S	0.0	0.0	0.0	0.0	0.0
WI1S	28.6	34.4	38.2	36.8	38.4
A-Tree	7.4	23.4	27.7	43.8	43.7
WA-Tree	38.2	53.6	56.0	67.5	68.3
SERT	27.2	52.2	61.9	63.9	67.9
WSERT	31.6	54.4	63.7	65.4	69.8
DWSERT	32.2	56.2	65.9	68.4	72.7

Examples of the WI1S, WA-Tree, and DWSERT routings produced for the same fifteen point signal net are shown in Figures 3, 4, and 5, below. The hollow dot is the source pin, the black ones sinks. A comparison of the performance characteristics of these particular routings gives insight into Table 7, above. The statically wiresized Iterated One Steiner tree has a maximum propagation delay of 4.05 ns. The non-wiresized I1S tree has a maximum propagation delay of 6.05 ns. The statically wiresized A-Tree has a maximum propagation delay of 3.00 ns. The non-wiresized A-Tree tree has a maximum propagation delay of 4.05 ns. The dynamically wiresized Steiner Elmore routing tree has a maximum propagation delay of 2.55 ns., a 15% improvement over the next best contender, the statically wiresized A-Tree. Thus, we see that at least for a typical fifteen point case, our dynamically wiresized construction indeed betters all statically wiresized ones.

FIGURE 4. Example of fifteen point WI1S routing

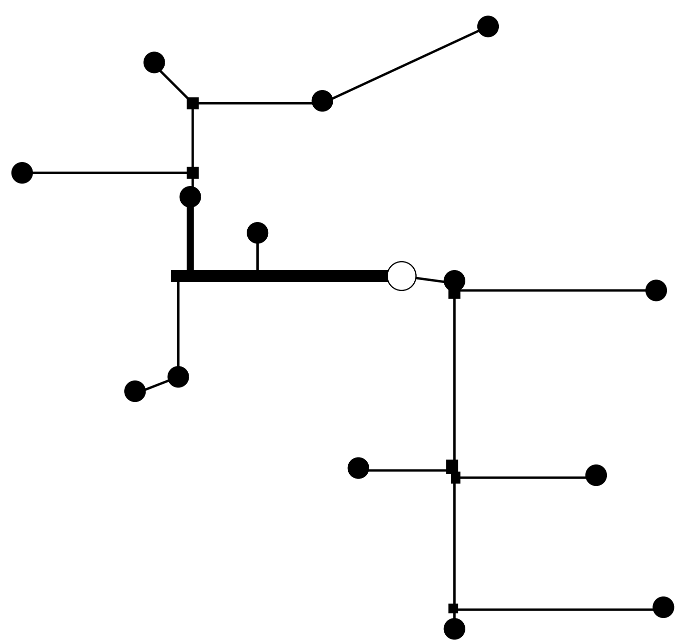


FIGURE 5. Example of fifteen point WA-Tree routing

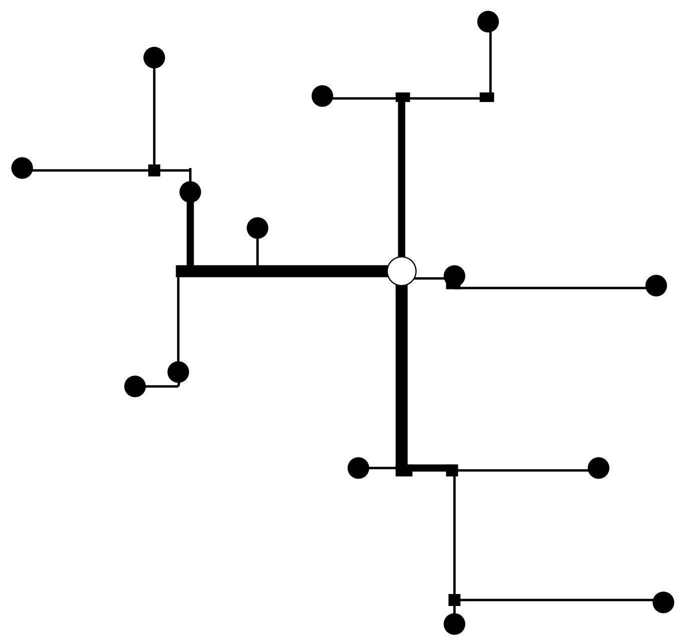
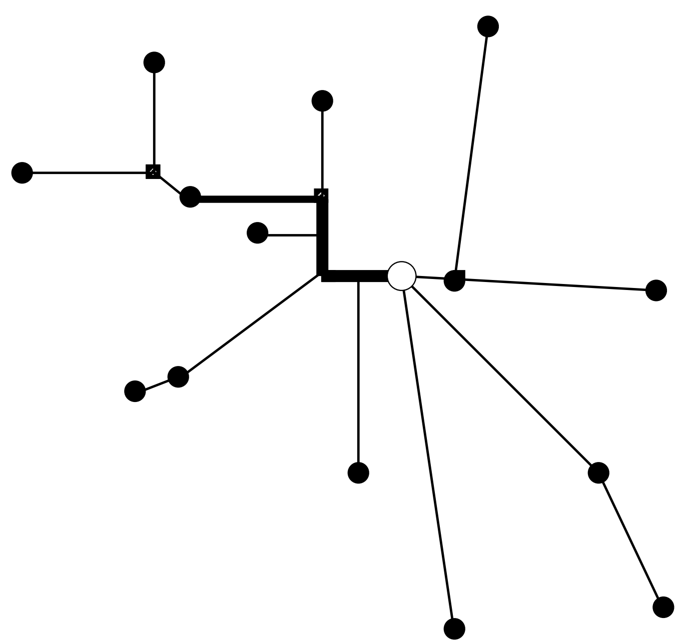


FIGURE 6. Example of fifteen point DWSERT routing



6. Conclusions

6.1 Summary and Interpretation

Upon inspection of the table of SPICE delay vales (Table 7, above), we first note that wiresizing in general, either static or dynamic, betters the performance of existing algorithms. The increase varies greatly, from as little as 1.5% when statically wiresizing the twenty-pin SERT routings to as great as 38.4% when statically wiresizing the twenty-five pin I1S routings. This verifies the intuition that wiresizing will indeed help almost any routing construction. It is important, though, to understand that non-wiresized constructions should not be compared directly to wiresized ones. There is a significant increase in total wire length that occurs when routings are wiresized, and this translates into higher production costs. Thus, wiresizing should be used only when performance is paramount or when the cost-performance trade-off is acceptable.

Interestingly, though, note that only a slight improvement occurs when the SERT routings are statically wiresized, unlike the more profound change that occurs when the I1S and A-Tree algorithms are wiresized. This occurs because the near-optimal SERT topologies tend to be star-like, with most sinks being directly connected to the source. Thus, the lower resistance of a wider edge does not compensate for the increase in capacitance that would occur with more complex tree structures. On the other hand, DWSERT not only betters the SERT delay numbers by as much as 5%, but also avoids yielding these degenerate topologies. For example, the SERT solution to the fifteen-pin signal net routed in Figures 3, 4, and 5 is degenerately star-like, which would make it impractical to implement. Similarly, because static wiresizing inherently maintains the input topology, the WSERT routings are also generally star-like. But the DWSERT algorithm avoids this degenerate behavior completely: in Figure 5 there are only four wires emanating from the source. Thus, while the wonderfully high numbers for both the SERT and WSERT algorithms over all net sizes are somewhat illusory, the even better numbers for DWSERT are not. DWSERT seems to be both high-performance *and* practical. The delay-based DWSERT improves on the best geometric router (the A-Tree) for most net sizes, and thus should be an excellent general-purpose algorithm. But note that for five pin nets, the WA-Tree handily beats all other wiresized contenders. This occurs because smaller A-Trees are nearly optimal in both path length and edge length. Thus, the WA-Tree construction might be a better choice for the routing of smaller nets. The loser among the bunch seems to be the I1S algorithm and its wiresized cousin; both are significantly out-performed by the other constructions for all net size. This occurs because the single constraint of minimum wire length does not capture the goal of "high-performance" or "low-delay" accurately.

6.2 Applications and Implications

In terms of immediate application, chip designers can use the new DWSERT routing algorithm to speed up current or forthcoming chips. Although computers seemingly get faster by the week, it is only through the design of new production methods that such advances actually occur. Just as the exponential increase in computing power over the last two decades allowed computers to be put on the desktop of the average citizen, additional breakthroughs during the next twenty years could affect another similar paradigm shift. It is through incremental work such as this that the computer revolution continues.

More specifically, the applications most likely to benefit from algorithms such as DWSERT are those where the foremost goal is to minimize delay, regardless of wire length and total chip cost. DWSERT's appropriateness for applications where the trade-off between cost and performance is a grey area is questionable. It should be up to the human designer using DWSERT to compare its output to other routing options, and asses the cost/benefit ratio. If wiresized routings become the norm in the near future, unwiresized routers will become outdated and will need to be updated with at least a minimal post-processing static wiresizing scheme. This will also drive designers to reconsider the placement phase of VLSI design so that wiresizing can be used more effectively.

6.3 Closing

Herein, we have analyzed the impact of wiresizing on the performance of both geometric and delay-based routing constructions. Whereas previous wiresizing schemes are static, in that they wiresize an existing topology, we introduced a new *dynamic* delay-based wiresizing technique. The new technique uses wiresizing considerations to drive the routing construction itself. SPICE simulations indicate that dynamic wiresizing affords improved performance over static wiresizing, and additionally avoids degenerate topologies. Moreover, dynamically-wiresized Elmore-based constructions seem to significantly outperform all previous methods in terms of maximum source-sink SPICE delay, affording up to 73% average maximum delay improvement over traditional Steiner routings.

Appendix 1: Works Cited

[1] C. J. Alpert, T. C. Hu, J. H. Huang, and A. B. Kahng, *A Direct Combination of the Prim and Dijkstra Constructions for Improved Performance-Driven Global Routing*, Tech. Rep. CSD-TR-920051, Computer Science Department, UCLA, 1992.

[2] T. Barrera, J. Griffith, S. A. McKee, G. Robins, and T. Zhang, *Narrowing the Gap: Near-Optimal Steiner Trees in Polynomial Time*, in Proc. IEEE Intl. ASIC Conf., Rochester, NY, September 1993, pp. 87-90.

[3] J. Cohoon and J. Randall, *Critical Net Routing*, in Proc. IEEE Intl. Conf. Computer Design, Cambridge, MA, October 1991, pp. 174-177.

[4] J. Cong, A. B. Kahng, G. Robins, M Sarrafzadeh, and C. K. Wong, *Provably Good Performance-Driven Global Routing*, in IEEE Trans. Computer-Aided Design, 11(6), 1992, pp. 739-752.

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

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

[7] W. C. Elmore, *The Transient Response of Damped Linear Networks with Particular Regard to Wide-Band Amplifiers*, in J. Appl. Phys., 19(1):53-63, 1948.

[8] A. B. Kahng and G. Robins, *A New Class of Iterative Steiner Tree Heuristics With Good Performance*, IEEE Trans. Computer-Aided Design, 11 (1992), pp. 1462-1465.

[9] S. Prasitjutrakul and W. J. Kubitz, *A Timing-Driven Global Router for Custom Chip Design*, Proc. IEEE Intl. Conf. Computer-Aided Design, Santa Clara, CA, November, 1990, pp. 48-51.

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

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

Appendix 2: Elmore Delay Formula

This section explains in further detail exactly how the Elmore delay formula, from Elmore [7], is defined.

Given routing tree T rooted at n_0, let e_i denote the edge from n_i to its parent. The resistance and capacitance of edge e_i are denoted by r(e_i) and c(e_i), respectively. Let T_i denote the subtree of T rooted at n_i, and let c_i denote the sink capacitance of n_i. We use C_i to denote the *tree capacitance* of T_i, namely the sum of sink and edge capacitances in T_i. Using this notation, the Elmore delay along edge e_i is equal to r(e_i) (c(e_i)/2 + C_i). Let r_d denote the output driver resistance at the net's source. The Elmore delay $t_{ED}(e_i)$ at sink n_i is:

$$t_{\mathrm{ED}}(n_i) = r_d C_{n_0} + \sum_{e_j \in \mathrm{path}(n_0,\, n_i)} r(e_j)\,(c(e_j)/2 + C_j)$$

Elmore delay has a compact definition and can be quickly evaluated at *all* sinks in $O(k)$ time. The calculation uses two depth-first traversals: one to compute the delay along each edge and one to sum up the delays along each source-sink path. Thus, efficient implementation is possible, which is important if it is to be used internally in a routing construction, as it is in the SERT, WSERT, and DWSERT constructions.

Appendix 3: Biographical Sketch of Student

I am a fourth-year student at the University of Virginia majoring in both Computer Science and Applied Mathematics. My current research interests include VLSI CAD, Networks, Graphics, Language Theory, and Algorithms. I plan on going to graduate school next Fall in computer science at the University of California, Berkeley. I have taken a rigorous academic curriculum, including graduate classes and some independent study. Additionally, I have taken great pains to keep my education broad by taking classes that may not be worth degree credit, but are certainly worth taking, such as English, pure Math, Philosophy, and Music. My greatest interest outside of academia would probably be music, in both recorded and live form. I enjoy performing jazz and rock on piano and guitar, and I enjoy experimenting with the fusion of musical instruments and computers. The thesis project is important to me because it gave me a chance to exhibit my ability to do research, and because the resulting publication was instrumental in getting me into one of the top graduate schools in the country--an immediate goal of my undergraduate education. Born in Miami, Florida, I am 22 years of age, and currently reside in Atlanta, Georgia. I plan to return to Atlanta after obtaining my M. S. and Ph. D. in Computer Science from Berkeley, where I will either join academia or start a small company (Got any venture capital?).

Appendix 4: ISCAS '94 Publication

Attached is our conference publication for the 1994 International Symposium on Circuits and Systems in London, England. The full paper is available upon request from either myself or the contact author, professor Gabriel Robins of the department of Computer Science.