ON OPTIMAL INTERCONNECTIONS FOR VLSI

ON OPTIMAL INTERCONNECTIONS FOR VLSI

Andrew B. Kahng
University of California
Los Angeles, California, USA

Gabriel Robins
University of Virginia
Charlottesville, Virginia, USA

KLUWER ACADEMIC PUBLISHERS
Boston/London/Dordrecht

To the field of VLSI CAD

CONTENTS

LIST OF FIGURES

Chapter 1

Chapter 2

Appendix A

LIST OF TABLES

1

PRELIMINARIES

1.1 PREFACE

This book discusses problems of "optimal interconnection" and describes efficient algorithms for several basic formulations. Our domain of application is the computer-aided design (CAD) of very large-scale integrated (VLSI) circuits, wherein interconnection design is now one of the most actively studied areas. However, much of what we develop can be applied to other domains ranging from urban planning to the design of communication networks. Because most formulations that we study are intractable, the term "optimal" in some sense is a misnomer: rather, our focus is on the reasoned and principled development of good heuristics.

This book is an outgrowth of the 1992 Ph.D. dissertation of Gabriel Robins [203] at the UCLA Computer Science Department. As such, it retains a highly personal perspective: it gives a retrospective of our own research, and it is colored by our research interests and our background in discrete algorithms and optimization. Our treatment also attempts to convey a sense of history – how our field has co-evolved with an emerging "science of VLSI design". With recent years having seen VLSI designs become increasingly performance-dominated, and thus interconnect-dominated, VLSI interconnections are indeed a rich domain for this historical view. In particular, our research on interconnection design has spanned the field's rapid transition from purely geometric formulations to more "physically-motivated" formulations.

Although we do not attempt an encyclopedic treatment, we do describe key relevant works, and the discussion is largely self-contained. We envision that this book will be useful as a reference for researchers and CAD algorithm de-

velopers, or as reading for a seminar on VLSI CAD, heuristic algorithms, or geometric optimization. Our own codes, which are cited throughout the book, are freely available to interested parties; see our contact information below.

1.2 THE DOMAIN OF DISCOURSE: ROUTING IN VLSI PHYSICAL DESIGN

Let us first outline the context for our particular subfield of VLSI CAD, namely, the global routing phase of physical design. For more complete reviews of VLSI design, and physical design in particular, the reader is referred to [168, 182, 194, 216].

The goal of VLSI CAD is to transform a high-level system description into a set of mask geometries for fabrication. This is typically accomplished by the following sequence of stages (see Figure 1.1).

- **Design Specification:** Starting from a real-world requirement (e.g. "secure communication"), a high-level system description (e.g., the "DES" data encryption standard) is developed which includes such parameters as architecture, performance, area, power, cost and technology.
- **Functional Design:** The design is transformed into a behavioral specification which captures the system I/O behavior using mathematical equations, timing diagrams, instruction sets and other devices.
- **Logic Design:** The functional design is represented in logical form, typically via Boolean expressions which may be subsequently optimized to reduce the complexity of the system description.
- **Structural Design:** The logic design is represented as a circuit using components from an available library of modules (e.g., NAND and NOR gates, standard cells, or building-block macros); this may also involve technology mapping steps.
- **Physical Design:** The structural design is transformed into the mask geometry for fabrication while adhering to underlying design rules for the chosen technology.

The last stage in this process, **physical design**, contains our area of interest. Physical design consists of two major steps. First, the *placement* step maps

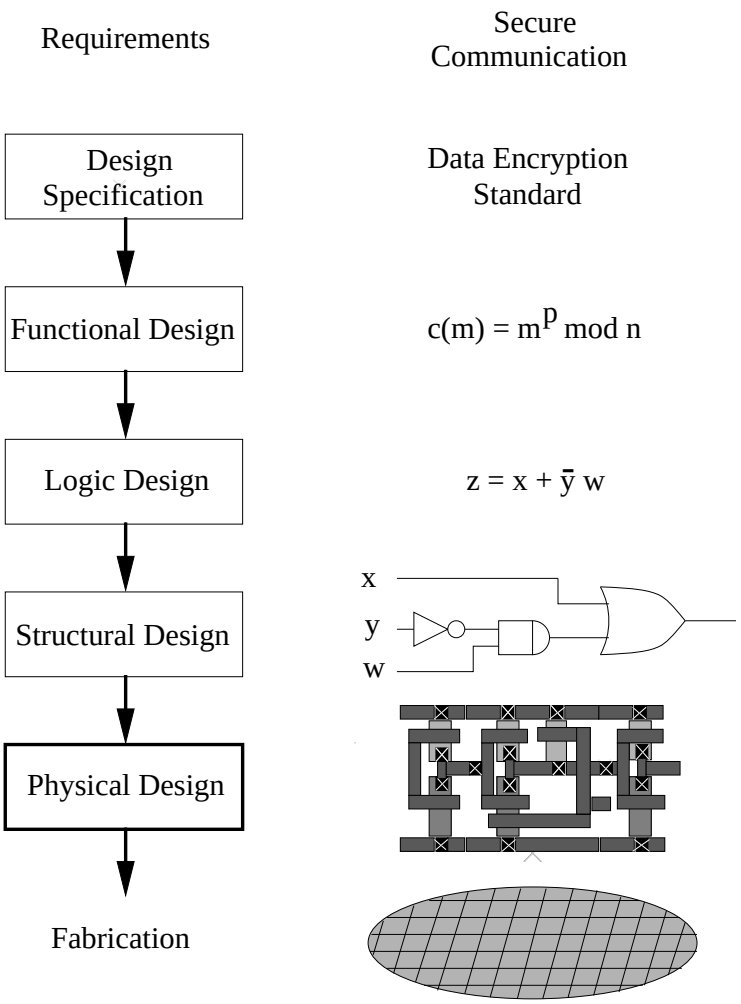


Figure 1.1 The VLSI design process.

functional units (modules) onto portions of a *layout region*, e.g., the surface of a chip. Second, the *routing* step interconnects specified sets of terminals, i.e., the *signal nets* of the design, by wiring within *routing regions* that lie between or over the functional units. (A signal net consists of a module output terminal

together with the various module input terminals to which the output signal must be delivered.)

Within the field of physical design, prevailing objectives have evolved over the years in response to advances in VLSI technology. When system operating frequencies were dominated by device switching speeds, placement and routing optimizations centered on reduction of total routing area. Subsequent advances in fabrication technology have increased packing densities, allowing more and faster devices to be placed on larger ICs. Leading-edge fabrication technology now goes well into submicron feature sizes, and circuit speeds are approaching gigahertz frequencies. The reduced feature size implies more resistive interconnects, and increased system complexity implies larger layout regions. Thus, minimization of interconnection delay has become the major concern in physical design.

In light of this trend, *performance-driven* physical design has seen much research activity within the past five years. Early works focused on performance-driven placement, with the standard objective being the close placement of modules belonging to timing-critical paths. However, performance-driven placement algorithms will achieve their intended effect only when the associated routing algorithms can realize the full potential of a high-quality placement. Thus, the emphasis in routing objectives has shifted from area minimization to delay minimization, and more recently to the *control* of interconnect delay (e.g., by limiting skews or delays at particular terminals). This range of routing objectives – area, delay, skew and beyond – defines the scope of this book.

Once an objective has been established, the actual routing of a given signal net can be decomposed into *global* and *detailed* routing. The *global routing* phase is a higher-level process during which the routing topologies of signal nets are defined over the available routing regions. Then, the *detailed routing* phase produces the actual geometries which realize the required connectivity on the fabricated chip. Our work applies to the global routing phase of physical design.[1]

We assume that during the global routing phase, all module and terminal locations have already been fixed in the plane, so that we need only ensure

[1]This traditional taxonomy may seem ambiguous. We do not address standard "detailed routing" topics such as switchbox routing or river routing. However, optimizing routing area and performance requires a concern with the specific geometry of the routing. In our discussion, we will define a routing topology by specifying for each edge its length and width, and the location of its endpoints; our work addresses "global routing" in that the particular detailed embedding of an edge between its endpoints does not matter.

electrical connectivity of the signal nets. With *standard-cell* or *gate-array* design methodologies, which have many small functional modules, global routing may be viewed as taking place in Manhattan geometry, i.e., distances between terminals are given by rectilinear distance. In other words, these design methodologies possess sufficiently high *porosity* that the routing problem can be formulated in the geometric plane. On the other hand, *building-block* design methodologies involve larger functional blocks or macro cells. Since these are often treated as obstacles, the routing problem is formulated with respect to a weighted *routing graph* that represents the available routing area. A standard model is the *channel intersection graph* (CIG), where each edge represents a *channel*(i.e., the empty rectangular space between adjacent modules) and each vertex corresponds to the intersection of two orthogonal channels [193] (see Figure 1.2). The edge weights of the CIG can be used to model channel width or congestion.

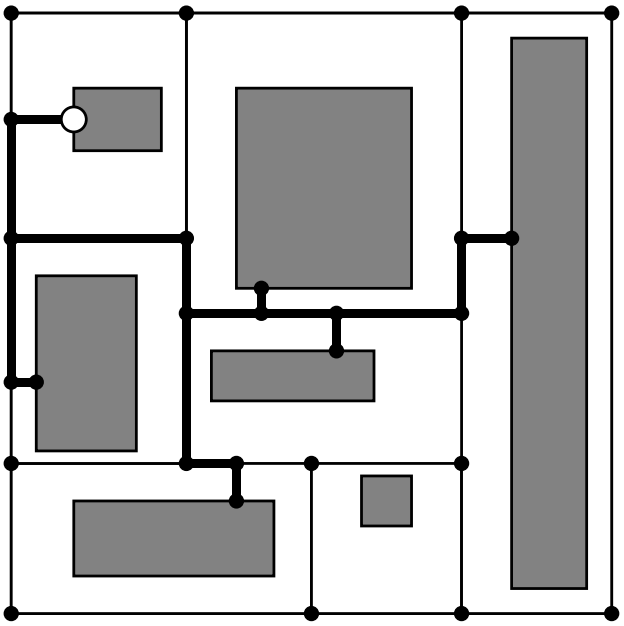


Figure 1.2 A channel intersection graph induced by a set of modules, and a routing tree that connects the highlighted terminals. The source is shown by a hollow dot.

A "true" global router processes multiple signal nets *simultaneously* using such techniques as simulated annealing, multicommodity flow or mathematical programming. However, many existing codes are *sequential*, or *"net-at-a-time"*, in that they establish a heuristic ordering of nets for routing and use ripup-and-retry techniques when the routing fails. (There are also even more fine-grain methods which route individual two-terminal subnets of signal nets.) With either type of global router, the key operation is to compute a good routing topology over a *single* signal net: hence, this book deals exclusively with methods that route a single net at a time.

As with previous routing constructions that have formed the basis of new global routers (e.g., "Steiner min-max trees"), each method that we develop can be transparently integrated into existing global routing approaches. In the mathematical programming approach, finding a routing solution for a given net generates a new entering basis column within a primal-dual iteration. In the sequential approach, routing solutions are found for the highest-priority nets first, leaving lower-priority nets to encounter more congestion and blockage. After each net is routed, the routing region costs (e.g., CIG edge weights) can be updated before the next net is processed.

We conclude this section with a review of basic conventions and terminology used throughout the book. We define a *terminal* to be a given location in the layout region. A signal *net* $S = \{s_0, s_1, s_2, \ldots, s_n\}$ is a set of $n+1$ terminals, with one terminal $s_0 \in S$ a designated *source* and the remaining terminals *sinks*. A *routing solution* is a set of wires that connects, i.e., spans, the terminals of a net so that a signal generated at the source will be propagated to all the sinks.

The rectilinear wiring technology implies an underlying "Manhattan" geometry, where the *distance* between points a and b is $d(a,b) = |a_x - b_x| + |a_y - b_y|$, i.e., the sum of the differences in their x- and y-coordinates. A *segment* is an uninterrupted horizontal or vertical wire, and any connection between two terminals will consist of one or more wire segments. VLSI and printed circuit board technologies admit multiple routing layers, where a preferred-direction routing methodology is used to facilitate design, manufacturability and reliability. In other words, the available wiring layers are partitioned, with horizontal wire segments preferentially routed on certain layers, and vertical wire segments routed on the other layers. A connection between two wire segments from different layers is called a *via*.

Sometimes it is convenient to embed S in an underlying *routing graph* $G = (V, E)$, consisting of a set of vertices V and a set of edges $E \subseteq V \times V$. Thus, the set of terminals is some $S \subseteq V$. A *subgraph* of G is a graph $G' = (V', E')$

with $V' \subseteq V$ and $E' \subseteq E$, and $E' \subseteq V' \times V'$. A routing solution is a subgraph of G that spans S. A *path* between two vertices $x, y \in V$ is a sequence of k edges of the form $(x, v_{i_1}), (v_{i_1}, v_{i_2}), \ldots, (v_{i_k}, y)$, where $(v_{i_m}, v_{i_{m+1}}) \in E$ for all $1 \leq m \leq k-1$. A graph is *connected* if there exists a path between each pair of vertices. A graph is a *tree* if it is connected but the removal of any edge will disconnect it. Since a tree topology uses the fewest edges of any spanning graph over the signal net, i.e., $|S| - 1 = n$ edges, routing formulations typically seek a tree topology.

A *weighted graph* has a non-negative real weight assigned to each of its edges. The *cost* of a weighted graph is the sum of its edge weights. A *shortest path* in G between two vertices $x, y \in V$, denoted by $minpath_G(x, y)$, is a minimum-cost path connecting x and y. In a tree T, $minpath_T(x, y)$ is simply the unique path between x and y. For a weighted graph G we use $dist_G(x, y)$ to denote the cost of $minpath_G(x, y)$. The distance from the source to a given sink s_i in a tree is denoted as $l_i = dist_T(s_0, s_i)$.

Because a signal net is inherently oriented from its source to its sinks, we use the special notation R_i to denote the cost of the shortest s_0-s_i path in G, i.e., $R_i = dist_G(s_0, s_i)$. We use R to denote the maximum R_i value over all sinks s_i, and say that R is the *radius* of the signal net. The radius of a routing tree T is $r(T) = \max_{1 \leq i \leq n} l_i$. Additional terminology will be developed throughout the following chapters, as needed. The reader is referred to, e.g., [67] or [92] for a more rigorous development of basic graph-theoretic concepts.

As noted at the outset, most problems encountered in VLSI CAD, including all of the interconnection formulations that we address, are intractable. While we resort to heuristic solutions, a basic precept in our work is to prove that our proposed heuristics perform well. For example, we often strive to show that the heuristic solution cost in the worst case (or average case) is no more than a constant factor from optimal. Since the practical relevance of a heuristic may hinge on issues beyond asymptotic time and space complexity, we also augment our performance bounds with empirical simulations using standard test cases from the literature, e.g., those maintained by ACM SIGDA (currently available by anonymous ftp to <mcnc.org>).

1.3 OVERVIEW OF THE BOOK

Beyond its sketch of our application domain of VLSI routing, the present chapter also surveys the main results contained in this book. Chapters 2, 3 and 4 are respectively entitled Area, Delay, and Skew. These form the core of the book, and address three fundamental routing objectives: (i) minimization of total wirelength, (ii) minimization of signal delay, and (iii) minimization of skew among signal arrival times. Chapter 5 provides new frameworks for the simultaneous optimization of multiple competing objectives; one such framework allows various unifications of the techniques developed in the preceding three chapters. The following subsections summarize the key developments of each chapter.

1.3.1 Minimum Area: The Steiner Minimal Tree Problem

VLSI design rules dictate a minimum separation between wires, and therefore the area occupied by the routing on a chip is roughly proportional to the total wirelength of the routing. Added wirelength generally increases signal delay and power consumption due to increased resistance and capacitance. Other system cost measures, e.g., those based on fabrication cost, yield and reliability, also increase with chip area. Thus, a fundamental objective is to minimize the total wirelength required to connect a prescribed set of points in the plane, i.e., the terminals of a given signal net. The subject of Chapter 2 is the *Steiner minimal tree* (SMT) problem, which for a given net S asks for a set S' of *Steiner points* such that the total edgelength of the minimum spanning tree (MST) over $S \cup S'$ is minimized. The main insight is that the points of S' will serve as internal nodes of the tree – "intermediate junction points" – which reduce the interconnection cost. Without introducing such points, the minimum-cost solution would simply be a minimum spanning tree over S.

The SMT problem is well-studied in combinatorial optimization and network design; see the monographs [138] and [139]. The geometry of VLSI, which usually allows only vertical and horizontal wiring directions, has motivated studies of the *rectilinear* version of the problem, typically for the wirelength estimation and global routing phases of layout design. With only a few highly constrained exceptions, existing variants of the SMT problem are NP-complete. Most SMT heuristics in the literature have analogies to classic minimum spanning tree constructions; this is in part due to the MST being a constant-factor approximation to the SMT, with performance ratio $\frac{3}{2}$ in the rectilinear metric. However, the first result of Chapter 2 defines a general class of "MST-based"

SMT heuristics, and shows that such methods cannot have performance ratio better than that of the simple MST approximation.

The focus of Chapter 2 lies in developing the Iterated 1-Steiner (I1S) heuristic, which iteratively finds optimal Steiner points that are added directly into the set S. The I1S construction thus avoids traditional analogies to minimum spanning tree solutions, and in practice achieves good performance even on inputs that are pathological for previous heuristics. For random 8-point planar instances, I1S solution costs are optimal for 90% of all instances, and average within 0.25% of optimal overall. (The I1S approach also applies to graph instances and higher-dimensional geometric instances.) The chapter describes a straightforward, efficient implementation of I1S, along with such enhancements as a parallel implementation that achieves near-linear speedup. Similarities between I1S and the recent method of Zelikovsky are also discussed.

Finally, Chapter 2 develops the result that any pointset in the Manhattan plane has an MST with maximum degree 4, and that in three-dimensional Manhattan space the maximum MST degree is 14 (the best previous bounds were 6 and 26, respectively); this improves I1S runtimes and is also of independent theoretical interest. The chapter concludes with a discussion of the Steiner problem in graphs.

1.3.2 Minimum Delay: Toward Optimal-Delay Routing Trees

Chapter 3 considers minimization of signal delay, which is synonymous with "performance-driven" system design. As VLSI technology scales to smaller feature sizes and larger layout areas, signal delays become interconnect-dominated, i.e., signal delay through interconnects increasingly dominates delay through devices. In leading-edge technologies, minimum-delay wiring topologies can differ substantially from minimum-area (SMT) wiring topologies.

The signal delay objective takes us from the *unoriented* pointset of the Steiner minimal tree problem to an *oriented* collection of terminals in the layout plane. Such a collection of terminals, which we call a *signal net*, has one identified *source* terminal; the remaining terminals are *sinks*. Typically, the source terminal is the output of a gate, and the sinks are the fanins for that output signal at inputs of other gates.

The discussion of Chapter 3 centers on four issues which have guided recent progress in minimum-delay routing heuristics. First, there is the issue

of *technology-dependence* in the routing construction, e.g., a simple analysis of Elmore delay in distributed RC trees shows that routing objectives should be dependent on parameters of the prevailing interconnect technology. We thus give a taxonomy of methods based on their tunability to specific technology parameters and signal net criticalities, and demonstrate the advantages of such tunable methods as the "Elmore routing tree" approach and the Prim-Dijkstra tradeoff.

Second, the chapter compares *"actual delay"*, versus *geometric*, routing objectives. To a first-order approximation, signal delay from the source to a given sink is proportional to the source-sink pathlength in the routing tree. This *linear delay* approximation suggests minimizing the maximum source-sink pathlength in the routing tree (i.e., a geometric "minimum-radius" criterion). On the other hand, reducing the total cost of the routing tree will reduce its lumped capacitance (i.e., a geometric "minimum-cost" criterion). We review how early works employed geometric criteria to achieve tractability in both the design and the analysis of routing heuristics. Of particular interest is a "bounded-radius, bounded-cost" (BRBC) approach which seeks a minimum-cost routing tree subject to a given bound on tree radius; we describe an algorithm which simultaneously minimizes both tree cost and tree radius to within constant factors of optimal. The BRBC approach and its analysis generalize to Steiner routing and to routing in arbitrary weighted graphs that capture the variation of routing costs over the layout region. The chapter gives details of recent methods, notably the "Elmore routing tree" variants which obtain reduced signal delays by optimizing higher-order delay estimates directly.

Third, we discuss minimization of *sink-dependent* delay, as opposed to *net-dependent* delay. Here, the key observation is that timing-driven placement and routing are typically iterated with static timing estimation, so that critical-path information is available during the routing tree construction. With this in mind, the traditional objective of minimizing maximum sink delay is "net-dependent" in that it ignores available path-dependent information. An approach which optimizes delay to identified critical sinks, such as that given in 1993 by Boese, Kahng and Robins [34], seems better matched to modern design methodologies. More recent work of Boese et al. provides an interesting addendum to the earlier SMT discussion: it generalizes Hanan's theorem to Elmore delay-optimal Steiner trees and gives a new "peeling" decomposition for optimal Steiner trees.

Finally, Chapter 3 addresses the issue of *demonstrable quality* for minimum-delay routing heuristics. Analogous to the empirical studies of the I1S SMT heuristic in Chapter 2, we present empirical studies showing near-optimality of a construction for minimum Elmore delay at prescribed critical sinks. The chap-

ter concludes with a review of two other recent advances in performance-driven interconnect design; these involve wiresizing and non-tree routing techniques. An Appendix provides the basic theory behind several efficient delay estimates, and also discusses measures of accuracy and fidelity for the linear, Elmore, and two-pole delay approximations.

1.3.3 Minimum Skew: The Zero-Skew Clock Routing Problem

In a high-performance VLSI design, circuit speed is limited not only by the signal propagation within and between circuit elements, but also by the *skew* between signal arrival times. The form of skew most often studied is *clock skew*, i.e., the difference between longest and shortest arrival times of a clock signal at synchronizing elements of the circuit. Clock skew minimization, and in particular the "zero-skew clock routing" problem, has become a central issue in the design of leading-edge systems. However, it should be noted that skew control for arbitrary signal nets is also of increasing importance, as are related problems of *prescribed-skew* or *bounded-skew* routing.

Chapter 4 discusses clock tree construction to minimize skew and wirelength as a combination of two processes: *topology generation*, and *geometric embedding* of the topology. We present methods which accomplish each of these processes using either the linear or Elmore delay model to guide the construction. Our discussion focuses on so-called "exact zero skew" clock routing constructions.

The first part of Chapter 4 uses the linear delay model to motivate a *pathlength-balanced tree* problem formulation, which seeks a minimum-cost tree with all source-sink pathlengths of equal length. We describe a simple approach, based on iterative geometric matching, for generating a clock tree topology while simultaneously embedding it in the layout region.

The second part of the chapter describes the Deferred-Merge Embedding (DME) algorithm, which embeds any prescribed connection topology (i.e., a binary tree with the clock sinks at the leaves), so as to create a clock tree with zero skew while minimizing total wirelength. The algorithm runs in linear time, and always yields *exact* zero skew trees with respect to a given monotone delay model such as linear or Elmore delay. The DME method achieves substantial cost reductions over earlier constructions, and can be combined with previous methods that concentrate on generation of the clock tree topology.

Finally, the third part of the chapter unifies the topology generation and geometric embedding of exact zero-skew clock trees. Under the linear delay model, the two phases of the DME algorithm (bottom-up identification of loci for "zero-skew balance points", followed by top-down selection of these balance points within a minimum-delay zero-skew embedding) can be replaced by a single top-down phase. Where DME would nominally require a prescribed topology as input, this top-down construction allows the clock tree topology to be determined dynamically and flexibly while being optimally embedded at the same time. A natural outgrowth is a DME-like algorithm for *single-layer*, exact zero-skew clock routing; such a construction is increasingly sought to minimize signal attenuation through vias, simplify buffering optimizations, and maximize process-variation independence.

Chapter 4 also describes extensions of these clock routing methods to "min-max" delay constraints and *bounded-skew* routing for general signal nets. The chapter concludes by noting additional issues and problem formulations, including optimal buffering hierarchies for minimum phase delay, and multiple-level clock trees for multi-chip module packaging.

1.3.4 Multiple Objectives

The last chapter of the book, Chapter 5, discusses frameworks and techniques which enable the simultaneous optimization of multiple competing objectives. Section 5.1 notes that beyond the nominal total wirelength, the grid-based structure of VLSI routing resources provides additional information for determining the impact of a given routing solution on layout area. The discussion explores a new *minimum density* objective for spanning and Steiner tree constructions, which seeks to balance the use of horizontal and vertical routing resources. We describe two heuristic constructions for low-density spanning trees whose outputs are within small constants of optimal with respect to both tree cost and density. (The proof techniques suggest a constructive lower bound scheme which affords tighter estimates of solution quality for a given problem instance.) Of particular interest is that the minimum density objective can be transparently combined with, e.g., minimum radius or minimum skew – without affecting asymptotic solution quality with respect to these competing objectives.

While previous chapters each focus on a fundamental routing criterion (i.e., area, delay or skew), many secondary objectives may exist, including congestion avoidance, jog minimization, reliability, etc. Section 5.2 develops a

general framework of *multi-weighted graphs*, in which multiple competing objectives can be simultaneously optimized. This is accomplished by assigning to each edge a *vector* of weights, corresponding to the various optimization criteria; graph searches are then guided by the weighted average of the edge weights according to designer-specified tradeoff parameters. This framework is applicable to graph-based routing regimes, such as building-block design and field-programmable gate array layout.

Finally, we describe optimization within the framework of a *continuously-weighted layout region*, which can be induced by the simultaneous consideration of multiple criteria (e.g., reliability, thermal density, and routing congestion). Within this framework, we consider a problem which has applications ranging from circuit board routing to vehicle navigation, namely, finding a minimum-cost prescribed-width path connecting a given source and destination [131]. Previous path routing approaches such as Dijkstra's algorithm implicitly assume that the path is of zero width, but this assumption is usually not realistic (e.g., consider routing a wide bus, or traces on a circuit board). Section 5.3 develops a network-flow based approach to prescribed-width routing in a continuously weighted region. Interestingly, the extension to higher dimensions can solve a discrete version of Plateau's problem, which seeks a minimum-area surface that spans a given closed curve [130].

1.4 ACKNOWLEDGMENTS

This book is the product of the research, suggestions, and technical assistance of many individuals. We first thank the students who have been so dedicated to the research that forms the basis of this book. In alphabetical order[2], they are: Mike Alexander, Charles J. Alpert, Kenneth D. Boese, Dennis Jen-Hsin Huang, Berni A. McCoy, Chung-Wen Albert Tsao and Tongtong Zhang. Any list of specific debts must begin with Ken Boese, who developed much of the core material in Chapters 3 and 4, including the characterization of delay-optimal routing trees and the results concerning the DME clock routing algorithm. The precise exposition in these sections is a product of Ken's efforts. Berni McCoy dedicated well over a year to investigations of accuracy and fidelity of delay estimates, near-optimality of the ERT construction, dynamic wiresizing and non-tree routing – these results appear throughout Chapter 3. Mike Alexander developed the graph generalization of I1S in Chapter 2, as well as the multi-

[2]Since early 1991, listing names in alphabetical order has been the "official" policy on all our publications as well.

weighted graphs framework of Chapter 5. Chuck Alpert and Dennis Huang pursued the Prim-Dijkstra tradeoff of Chapter 3 through its many incarnations; Chuck also contributed to the study of minimum-density routing trees in Section 5.1. Albert Tsao developed the Planar-DME algorithm which forms the capstone of Chapter 4. Ken and Chuck, along with Lars Hagen, provided many critical comments as this book took shape. Brett Coryell and Brian Robinson provided invaluable help throughout the final stages of writing. Certainly, it is our students who have always been our best critics, motivators, and colleagues.

The various research collaborations that form the basis of this book list a number of other coauthors: Tim Barrera, Ting-Hai Chao, Jim Cohoon, Jason Cong, Todd Hodes, Joseph Ganley, Jeff Griffith, Jan-Ming Ho, Yu-Chin Hsu, T. C. Hu, David Karger, Kwok-Shing Hardy Leung, Sally McKee, Sudhakar Muddu, Jeff Salowe, Majid Sarrafzadeh, C. K. Wong and Dian Zhou. David Karger suggested the second Prim-Dijkstra tradeoff of Chapter 3. Dian Zhou provided us with the original "Two-Pole" simulator code, while Sudhakar Muddu provided invaluable amendments to this code and the totality of our knowledge concerning delay analysis of interconnects. Jason Cong provided the geometric analysis of H-flipping cited in Chapter 4, as well as the bounded-radius minimum routing tree problem formulation in Chapter 3. Since 1990, Jason and his students have brought much energy to VLSI CAD at UCLA.

Others who have over the years provided advice, feedback, and/or use of their codes include: Jim Aylor, Marshall Bern, Stephen Brown, John Canny, Pak Chan, Kamal Chaudhary, Brett Coryell, Erik Cota-Robles, Wayne Wei-Ming Dai, Miloš Ercegovac, Eli Gafni, Basil Gordon, Sheila Greibach, Lars Hagen, Rajeev Jain, Kevin Karplus, Samir Khuller, Ernest S. Kuh, C. L. Liu, John Pfaltz, Sinai Robins, Brian Robinson, Jonathan Rose, Andy Schwab, Michael Shur, Ashok Vittal and Neal Young. Eli Gafni provided the key pointer to the shallow-light construction of Awerbuch, Baratz and Peleg that led to the BRBC algorithm in Chapter 3.

The dedication of this book, *"To the field of VLSI CAD"*, requires some elaboration. As newcomers to the field, we are grateful for the inspiration provided by the leading researchers who preceded us. Above all, Professor T. C. Hu of U.C. San Diego has been the one constant source of guidance, wisdom and research interaction in our academic careers. His influence predates our studies in VLSI CAD, and goes much deeper; he has truly shaped us both. Professor Ernest S. Kuh of U.C. Berkeley has profoundly influenced how the field of VLSI CAD is defined today. There is an ethic of quiet excellence in "kuhsgroup" alumni: Professors Chung-Kuan Cheng, Kwang-Ting (Tim) Cheng, Wayne Wei-Ming Dai and Malgorzata Marek-Sadowska, as well as Professor Kuh himself, will

long remain our models of collegiality, activity and impact. Professor C. L. Liu of the University of Illinois has for several years given wholeheartedly of his experience, advice, and support. He is a gifted teacher, scholar and raconteur, and it is always a rare pleasure to be in his company. His former students – Jason Cong and Martin Wong in particular – are of course models for all young faculty in the field. Further inspirations have derived from Majid Sarrafzadeh at Northwestern University; Daniel Gajski and his group at U.C. Irvine; Robert Brayton at U.C. Berkeley; Thomas Lengauer at GMD Bonn; Ralph Otten at Delft; and many others. The field that brings such individuals together truly deserves to flourish.

On a more personal level, Gabriel Robins would like to thank Bill Wulf and Anita Jones for all their support and sage advice, and for inspiring and nurturing so much of the shared vision that is unique to computer science at the University of Virginia. Randy Pausch has been a continuing source of inspiration, and a firm advocate of "the right culture". Together, the UVa Department of Computer Science and its Chair Jim Ortega deserve much credit for their support of young faculty development.

Our work was supported by National Science Foundation Young Investigator Awards MIP-9257982 and MIP-9457412, by National Science Foundation Research Initiation Award MIP-9110696, by Army Research Office grants DAAK-70-92-K-0001 and DAAL-03-92-G-0050, by setup funds provided by the UCLA School of Engineering and Applied Science during 1989-1991, by research initiation funds provided by the University of Virginia School of Engineering during 1992-1993, and by an IBM Graduate Fellowship. Part of this book was written during a Spring 1993 sabbatical that was hosted by Professor Ernest S. Kuh and his research group. Finally, this book would not exist without the incredible patience of Carl Harris at Kluwer Academic Publishers – a debt that goes beyond any possible statement of thanks.

Andrew B. Kahng
Department of Computer Science
University of California, Los Angeles
Los Angeles, CA 90024-1596
<abk@cs.ucla.edu>

Gabriel Robins
Department of Computer Science
University of Virginia
Charlottesville, VA 22903-2442
<robins@cs.virginia.edu>

2

AREA

Overview

To achieve a minimum-area layout, circuit interconnections should in general be realized with minimum total wirelength. This chapter discusses the corresponding *Steiner minimal tree* (SMT) problem, which seeks to connect a given set of points in the plane using the minimum amount of wiring. The SMT problem is central to VLSI global routing and wiring estimation; it also arises in such non-VLSI applications as communication network design. Recent reference books treat the Steiner problem in detail [138, 139]. Thus, in this chapter we will limit our discussion to the *rectilinear* SMT formulation, which reflects the Manhattan geometry of VLSI layout. The discussion focuses on an iterative construction, called Iterated 1-Steiner, that eschews traditional analogies to minimum spanning tree solutions. Practical implementation issues are discussed as well.

Our development will be as follows. We first demonstrate that many existing SMT heuristics have a performance ratio of $\frac{3}{2}$ in the Manhattan plane, which is the same bound achieved by the minimum spanning tree (MST) construction. We then develop the Iterated 1-Steiner (I1S) heuristic, an iterative construction that can achieve good performance even on inputs that are pathological for previous methods. For uniform distributions of 8-point instances in the plane, I1S obtains solution costs that are optimal for 90% of uniformly distributed instances, and average within 0.25% of optimal overall. (The I1S approach also applies to graph instances and higher-dimensional geometric instances.) We present a straightforward implementation of I1S, along with a parallel implementation that achieves near-linear speedup. Similarities between I1S and the recent method of Zelikovsky are also discussed. Finally, we show that any

pointset in the Manhattan plane has an MST with maximum degree 4, and that in three-dimensional Manhattan space the maximum MST degree is 14 (the best previous bounds were 6 and 26, respectively): this result improves I1S runtimes and is of independent theoretical interest. The chapter concludes with a discussion of the Steiner problem in graphs.

2.1 INTRODUCTION

In the *Manhattan*, or L_1, plane, the distance between points (a_x, a_y) and (b_x, b_y) is given by $|a_x - b_x| + |a_y - b_y|$. This is also known as *rectilinear* distance, and reflects the cost of wiring between two points in a VLSI layout.[1] Given a set P of n points in the plane, we often wish to connect these points using as little wire as possible. This objective arises in minimum-area VLSI global routing (since minimum-spacing design rules imply a roughly linear relationship between wirelength and wiring area), with P corresponding to the set of terminals in a signal net. In succeeding chapters, each terminal in the signal net will be distinguished as either a "source" or "sink", i.e., the interconnecting wire will have an implicit orientation. However, in this chapter we cast our description in terms of generic points in the plane since a solution to the problem of minimum-wirelength interconnection is inherently unoriented.

When all wires are "point-to-point", with no intermediate junctions other than points of P, the optimum solution is a minimum spanning tree (MST) over P, denoted as $MST(P)$. However, in VLSI routing it is possible to introduce intermediate junctions – called *Steiner points* – in connecting the points of P. The resulting planar *Steiner minimal tree* (SMT) problem is the subject of this chapter.

The Steiner Minimal Tree (SMT) Problem: Given a set P of n points in the plane, determine a set S of Steiner points such that the MST over $P \cup S$ has minimum cost.

An optimal solution to this problem is referred to as an SMT over P, or $SMT(P)$. Here, an edge in a tree T has *cost* equal to the distance between its endpoints; the cost of T itself is the sum of its edge costs, and is denoted by $cost(T)$.

[1] More generally, the distance between two points in the L_p plane is given by $\sqrt[p]{(\Delta x)^p + (\Delta y)^p}$. Thus, $p = 1$, $p = 2$ and $p = \infty$ define the Manhattan, Euclidean and Chebyshev norms.

We will focus on the *rectilinear Steiner minimal tree* problem, where every edge is embedded in the plane using a path of one or more alternating horizontal and vertical segments between its endpoints. Where no confusion is possible, we will overload the two concepts of a graph edge and a "physical" (i.e., embedded in the plane) edge, for example, when we speak of "connecting a point to an edge". Implicitly, we also assume that only a shortest-possible path of segments can be used to embed a given edge. Thus, an edge is embedded using some *monotone*, or "staircase", path between its endpoints. The *bounding box* of a pointset P denotes the minimum rectangle which contains all points of P and whose sides are oriented parallel to the coordinate axes. If an edge is embedded with minimum cost, its routing will remain within the bounding box of its endpoints.

Beyond its application to VLSI global routing, the rectilinear SMT problem also arises in wirelength estimation for circuit layout. Figure 2.1 shows an MST and an SMT for the same pointset in the Manhattan plane.

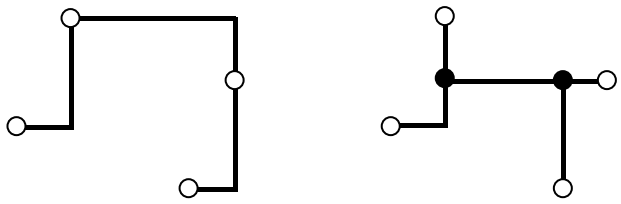


Figure 2.1 An MST (left) and an SMT (right) for a pointset with $n = 4$; hollow dots represent the original pointset P, and solid dots represent the set S of Steiner points.

Three results have greatly influenced the progress of research on the SMT problem. First, consider the set $H(P)$ of intersection points that are obtained when horizontal and vertical gridlines are drawn through every point of P. Hanan [116] showed that there exists an SMT whose Steiner points S are all chosen from $H(P)$, which we call the *Steiner candidate set* or the *set of Hanan points* (see Figure 2.2).[2] Snyder [222] has generalized Hanan's result to all higher-dimensional Manhattan geometries, and extensions to certain allowed-angle geometries [210] seem possible.

[2] Hanan's proof relies on a perturbative argument: if an edge of an SMT does not lie in the "Hanan grid", it can always be shifted onto a gridline without increasing the tree cost. Similar arguments have been applied by Chiang et al. [53] to prove a Hanan-like result for the SMT problem in a planar layout with varying routing region costs.

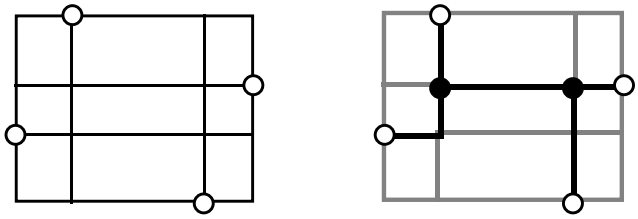


Figure 2.2 Hanan's theorem: there exists an SMT with all Steiner points chosen from the intersection points of horizontal and vertical lines drawn through points of P.

Second, Garey and Johnson showed that despite this restriction on the set of possible Steiner points, the rectilinear SMT problem is NP-complete [106]. Only a very few special cases have been solved optimally, e.g., a linear-time solution exists when all points of P lie on the boundary of a rectangle [2, 59], and pseudopolynomial algorithms have been proposed for the case when there are a limited number of rectilinear obstacles in the plane [52, 184]. Many heuristics have been proposed for the general problem, as surveyed in [138, 139].

In attacking intractable problems, a standard goal is to achieve a "provably good" heuristic, typically in the sense of having constant-factor performance ratio.[3] In light of the intractability of the rectilinear SMT problem, a third fundamental result is that of Hwang [135], who showed that the MST over P is a fairly good approximation to the SMT, with performance ratio $\frac{3}{2}$, i.e., $\frac{cost(MST(P))}{cost(SMT(P))} \leq \frac{3}{2}$ (or equivalently, $\frac{cost(SMT(P))}{cost(MST(P))} \geq \frac{2}{3}$) for any pointset P. Because the proof of this result is not trivial, and because several details will later prove useful, we first digress to sketch Hwang's proof.

Theorem 2.1.1 *(Hwang, 1976)* *For any pointset P, $\frac{cost(SMT(P))}{cost(MST(P))} \geq \frac{2}{3}$.*

Proof: The proof is by induction on the size of P. Given pointset $P = \{p_0, p_1, \ldots, p_{n-1}\}$, let M be the set of all SMTs over P. Partition M into M_1 and M_2, where an SMT m is in M_1 exactly when all nodes of P have degree

[3] The *performance ratio* of a heuristic is its asymptotic worst-case error from optimal. Let I denote an instance of a problem with optimal solution cost $opt(I)$, and let $H(I)$ denote the cost of the solution returned by heuristic H on instance I. Then, the performance ratio of H is $\lim_{n\to\infty} \sup_{|I|=n} \frac{H(I)}{opt(I)}$.

$= 1$ in m (i.e., each node of P is a leaf in m; such a topology has been termed a *full Steiner topology* in the literature). All other SMTs are in M_2. For any $m \in M_2$, we can split m into two components at a node having degree ≥ 2, and apply the induction hypothesis to each component separately. Thus, we need only prove the theorem for $m \in M_1$. For any SMT $m \in M_1$, observe that all Steiner points of m lie on a straight line, except perhaps the last one (see Figure 2.3).

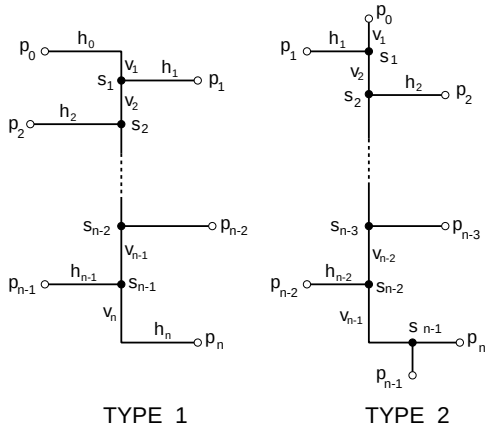


Figure 2.3 Two types of SMTs in M_1: Type 1 has all of its Steiner points on a line; Type 2 has all but one of its Steiner points on a line.

The strategy is to split m at some Steiner point s_q to yield subtrees m_1 and m_2, with m_1 being the induced subgraph over $\{p_0, p_1, \ldots, p_{q-1}\}$ plus the edge (s_{q-1}, s_q). Consider the subtree m_1. Assume that we can construct a path X_1 which visits the points $\{p_0, p_1, \ldots, p_{q-1}\}$ in sorted order such that

$$\frac{2}{3} \cdot cost(X_1) \leq cost(m_1).$$

Referring to Figure 2.3, this is equivalent to the existence of some k, $1 \leq k \leq n-2$, for which

$$\frac{2}{3} \cdot \left(\sum_{i=0}^{k-1} h_i + \sum_{i=1}^{k} h_i + \sum_{i=1}^{k} v_i \right) \;\leq\; \sum_{i=0}^{k} h_i + \sum_{i=1}^{k} v_i \tag{2.1}$$

Note that the terms on the left side of (2.1) represent the "zig-zag" path X_1 from p_0 to p_{k-1}, and that this path is one possible spanning tree. Then, we are done since $\frac{2}{3} \cdot cost(MST(m_2)) \leq cost(m_2)$ by the induction hypothesis.

If there is no such Steiner point s_q, then (2.1) does not hold for any $k, 1 \leq k \leq n-2$. Manipulating the sum of the resulting inequalities yields

$$\sum_{i=1}^{k} v_i \;\leq\; \sum_{i=1}^{k} h_i + (h_k - h_0), \qquad 1 \leq k \leq n-2. \tag{2.2}$$

Next, observe that we can assume the existence of some index j such that $h_i > h_{i-2}$ for all $i = 2, \ldots, j-1$, and $h_j \leq h_{j-2}$. In other words, the splitting point of the tree can be chosen so that *some* initial portion of m_1 looks like a "Christmas tree", as shown in Figure 2.4. This special structure of the h_i values, $0 \leq i \leq j-1$, allows us to set $q = j$ and connect $(p_j, p_{j-2}, p_{j-4}, \ldots, p_0, \ldots, p_{j-3}, p_{j-1}, p_j)$ in that order to yield a new tour t. The cost of tour t is equal to the perimeter R of the bounding box of points $p_0, \ldots, p_j$.

A path over $p_0, \ldots, p_j$ can be obtained by deleting an edge of the tour t. Observe that the four edges $p_{j-4} \to p_{j-2} \to p_j \to p_{j-1} \to p_{j-3}$ in t have total cost given by

$$R - h_{j-3} - h_{j-4} - \sum_{i=1}^{j-4} v_i - \sum_{i=1}^{j-3} v_i,$$

where the negative terms represent the cost of the subtour from p_{j-3} to p_0 and from p_0 to p_{j-4}. By (2.2), this quantity is

$$\begin{aligned}
&\geq\; R - h_{j-3} - h_{j-4} - \left(\sum_{i=1}^{j-3} h_i + (h_{j-3} - h_0) \right) - \left(\sum_{i=1}^{j-4} h_i + (h_{j-4} - h_0) \right) \\
&\geq\; R - 4 \cdot \sum_{i=0}^{j-3} h_i \\
&=\; R \cdot (1 - 4\theta)
\end{aligned}$$

where $\theta = (\sum_{i=0}^{j-3} h_i)/R$. If we delete from t the edge with maximum cost among these four edges, we obtain a path (i.e., a spanning tree) X_2 over the

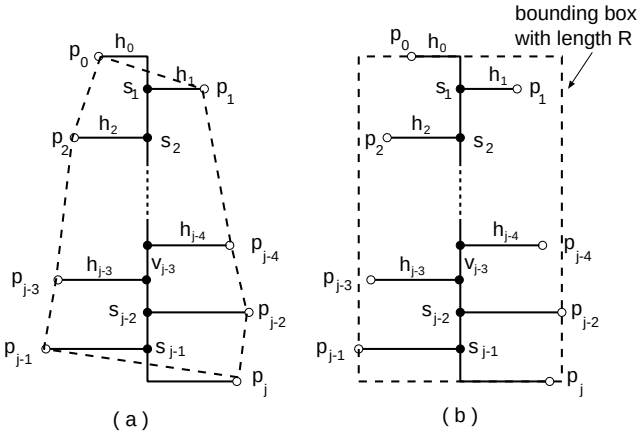


Figure 2.4 Cost of tour t in (a) is equal to the perimeter R of the bounding box of points $p_0, \ldots, p_j$ in (b).

points $p_0, \ldots, p_j$ with

$$cost(X_2) \leq R - \frac{1}{4}R \cdot (1 - 4\theta) = R \cdot (\frac{3}{4} + \theta).$$

The cost of the SMT m_1 is

$$\sum_{i=0}^{j-1} h_i + \sum_{i=1}^{j} v_i = \sum_{i=0}^{j-3} h_i + \frac{1}{2}R = R \cdot (\frac{1}{2} + \theta)$$

and we conclude that

$$\frac{2}{3}cost(X_2) \leq cost(m_1) \qquad \square$$

Hwang's result implies that any approach which improves upon an initial MST solution will have performance ratio at most $\frac{3}{2}$. Thus, many SMT heuristics in the literature resemble, or are otherwise based on, classic MST constructions [138]; we call such heuristics *MST-based strategies*. A leading example is the SMT heuristic of Ho, Vijayan and Wong [124], which exploits flexibility in the embedding of each rectilinear MST edge. Recall that in general, an edge between two points in the Manhattan plane will have many minimum-cost

embeddings; in the example of Figure 2.1, simply choosing the alternate "L" embedding for two of the three MST edges will cause the maximum possible overlapping of edges, and result in the SMT solution when redundant (overlapped) wire is removed. The authors of [124] give a linear-time construction for the *optimal* rectilinear Steiner tree derivable from a given MST in the sense of being embedded within the union of the bounding boxes of the MST edges. A second MST-based strategy due to Hasan, Vijayan and Wong [118] also begins with an MST topology, and iteratively adds as many "locally independent" Steiner points as possible to reduce the tree cost.

For over 15 years after the publication of [135], the fundamental open problem was to find a heuristic with (worst-case) performance ratio strictly less than $\frac{3}{2}$. A complementary research goal has been to find new heuristics with improved average-case performance. In practice, most SMT heuristics – including MST-based strategies – exhibit very similar performance. The standard experimental testbed consists of uniformly random instances (n points chosen from a uniform distribution in the unit square), which reflects observed terminal distributions from actual VLSI placements.[4] On such instances, heuristic Steiner tree costs usually average between 7% and 9% less than corresponding MST costs [138]. Results of Steele [229] establish the theoretical result that the average ratio $\frac{cost(SMT)}{cost(MST)}$ for random pointsets should converge to a constant as n grows large.[5] Bern and de Carvalho [27] estimated the average value of the ratio $\frac{cost(SMT)}{cost(MST)}$ to be 0.88; more recently, Salowe [208] has given an empirical estimate of this average ratio for n up to 100, using the most efficient known branch-and-bound code currently available (see Section 2.8 below).

[4] At least, such has been the claim throughout the literature. Optimization of abutments, vertical cell alignments, use of feedthroughs, and other criteria in module placement can result in highly non-random terminal placements for signal nets. For example, vertical alignment and feedthrough reduction will often cause the bounding box of a signal net to have very large *aspect ratio*, that is to say, ratio of the length of the larger side to the length of the smaller side.

[5] A more oblique motivation for MST-based approaches follows from asymptotic behavior of subadditive functionals of uniformly random pointsets in the Manhattan plane [23, 229]. Such functionals include the MST cost and the SMT cost, as well as the optimal traveling salesman tour cost, the optimal matching cost, etc. Steele [229] has shown that optimal solutions to random n-point instances of these problems have expected cost $\beta\sqrt{n}$, where the constant β depends on both the problem, e.g., SMT versus MST, and the underlying L_p norm. Thus, we expect the average MST cost and the average SMT cost to differ by a constant factor. (Of course, this result does not apply on an instance-by-instance basis.) The theory of subadditive functionals can have other implications for VLSI CAD optimizations. For example, VLSI layout engines (e.g., TimberWolfSC [212]) often use the semiperimeter of the pointset bounding box as a fast estimate of SMT cost. The $\sqrt{n}$ growth rate implies that this estimate can be refined by using a $\Theta(\sqrt{n})$ scaling factor at negligible added CPU cost; see the related work of Chung and Graham [55].

The worst-case bound of Hwang and the average-case analysis of Steele [229] together provide strong motivation for MST-based strategies. However, there are also reasons to consider alternative approaches. Section 2.2 shows that the $\frac{3}{2}$ bound is tight for any of a wide range of MST-based strategies [152], i.e., the MST for such instances is essentially unimprovable. This suggests that MST-based heuristics are unlikely to achieve performance ratio strictly less than $\frac{3}{2}$. Section 2.3 introduces the focal point of the chapter – the *Iterated 1-Steiner* (I1S) heuristic – whose simple iterative scheme avoids analogies to classic minimum spanning tree solutions. Key developments in the remainder of the chapter include:

- Bounds on the I1S performance ratio. In particular, the method has performance ratio $\frac{4}{3}$ on all "difficult" instances for which $\frac{cost(MST)}{cost(SMT)} = \frac{3}{2}$. We also contrast I1S with the recent breakthrough due to Zelikovsky, Berman and coauthors, namely, a heuristic which achieves performance ratio of $\frac{11}{8}$ for the rectilinear SMT problem.

- Performance enhancements to the I1S method, including a "batched" strategy, a perturbative strategy, and a randomization scheme for tie-breaking. Together, these bring I1S performance to within a small fraction of one percent from optimal for typical instance sizes. Tradeoffs between runtime and solution quality are also discussed.

- Practical implementation options, notably an implementation of the batched I1S variant that runs within time $O(n^3)$ per "round". This method is based on a dynamic MST update scheme, and is simple to code and considerably faster than the naive implementation. We also describe a parallel version of I1S that achieves near-linear speedup within a prototypical CAD environment consisting of a network of workstations.

- Extensions of I1S and its variants to three dimensions, and to the "two and one-half dimensional" case where all the terminals lie on L parallel planes (see, e.g., three-dimensional VLSI technology [117] and the design of buildings [221]).

- Two new bounds on the maximum node degree in an MST under the Manhattan metric. Specifically: (i) every two-dimensional pointset has an MST with maximum degree at most 4; and (ii) every three-dimensional pointset has an MST with maximum degree at most 14. (The best previous bounds were 6 and 26, respectively.) These degree bounds allow speedup of the I1S implementation and are of independent interest in algorithmic complexity theory.

- Extension of the I1S construction to arbitrary weighted graphs. A general methodology for increasing the power of heuristics using iterated constructions is described.
- Experimental results for I1S and its variants.

2.2 PERFORMANCE BOUNDS FOR MST-BASED STRATEGIES

Recall that the $\frac{3}{2}$ performance ratio of the MST approximation motivates *MST-based strategies*, which improve an initial MST solution by various means. Such strategies are enhanced by the efficiency of the MST computation for a planar pointset [114].

Trivially, an MST-based Steiner tree construction which has cost no greater than the MST cost will have performance ratio at most $\frac{3}{2}$. However, the actual performance ratio for many MST-based methods has remained unknown. It was believed that certain methods might be provably better than the simple MST approximation (cf. [137]), with the algorithms of Bern [26] and Ho, Vijayan and Wong [124] being two examples.

This section shows that any Steiner tree heuristic in a general class C of greedy MST-based methods will have worst-case performance ratio arbitrarily close to $\frac{3}{2}$, i.e., the same bound as the MST approximation. By "arbitrarily close", we mean performance ratio $> \frac{3}{2} - \epsilon \quad \forall \epsilon > 0$. Performance ratios are thus resolved for a number of heuristics in the literature with previously unknown worst-case behavior [26, 27, 103, 118, 124, 137, 202, 215] since they can be shown to belong to the class C. The enabling construction also serves to correct a claim in [124] and establish a lower bound of $\frac{3}{2}$ on performance ratios for some heuristics not in C, e.g., [137, 164, 220]. Analogous constructions in d-dimensional Manhattan geometry, with $d > 2$, show that all of these heuristics have performance ratio at least $\frac{2d-1}{d}$ [97].

2.2.1 Counterexamples in Two Dimensions

We now describe two prototypical heuristic approaches, called *MST-Overlap* and *Kruskal-Steiner*, for the rectilinear SMT problem. We then unify these approaches under a general template for greedy MST-based strategies. The

first approach starts with a rectilinear MST and obtains a Steiner tree by overlapping edge embeddings. In other words, a monotone (staircase) embedding is selected for each MST edge, and then all superposed segments are merged since they represent redundant wiring. Alternatively, we may view this approach as starting with an MST over P, then determining the minimum-cost Steiner tree which lies completely within the union of bounding boxes of the MST edges. Figure 2.5 illustrates this strategy with respect to the original example from Figure 2.1. The resulting Steiner tree has cost no greater than the MST cost.

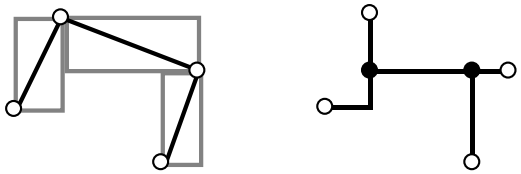


Figure 2.5 Optimal overlap of MST edges within their bounding boxes.

This approach has been studied by Hasan, Vijayan and Wong [118], Ho, Vijayan and Wong [124], Hwang [136], Lee, Bose and Hwang [164], and Lee and Sechen [165]. Ho, Vijayan and Wong [124] have given the best-possible result, namely, a linear-time algorithm for computing the *optimal* rectilinear Steiner tree derivable in this fashion. Their construction requires that no two edge bounding boxes of the MST intersect or overlap, unless the edges are adjacent. This property of the MST, known as *separability*, enables a dynamic programming approach. A method which determines a separable MST for any pointset P was given in [124].

Since the algorithm of Ho et al. dominates all other algorithms that share the goal of overlapping MST edges within the union of bounding boxes, we will treat it synonymously with the general approach itself, and use the name **MST-Overlap** to indicate either. It was conjectured that the worst-case performance ratio of MST-Overlap is less than $\frac{3}{2}$.

The example of Figure 2.6 results in an MST-Overlap performance ratio of exactly $\frac{3}{2}$. However, this example is not separable. If the starting MST is separable, a performance ratio arbitrarily close to $\frac{3}{2}$ can still result: Figure 2.7(a) shows a separable MST over a pointset for which the strict equality $\frac{cost(MST)}{cost(SMT)} = \frac{3}{2}$ holds; Figure 2.7(b) shows a perturbation of the point loca-

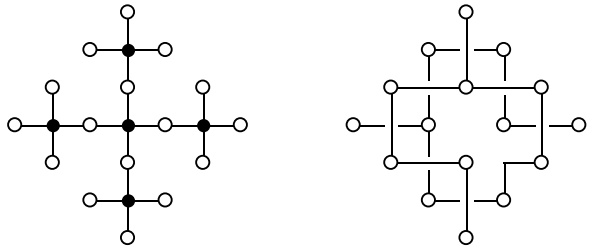


Figure 2.6 Example with strict equality $\frac{cost(MST-Overlap)}{cost(SMT)} = \frac{3}{2}$. On the left is the SMT ($cost = 20$); any Steiner tree derived from the MST on the right will have $cost = 30$.

tions such that the MST is unique; and Figure 2.7(c) shows the optimal SMT topology for both cases.

The second type of MST-based strategy builds a Steiner tree by emulating the standard MST constructions of Kruskal [160] or Prim [196], with connections to new Steiner points replacing direct connections between points in P. Examples of this strategy are discussed by Bern [26], Bern and Carvalho [27], Richards [202] and Servit [215]. Typically, embeddings of edges within their bounding boxes are left unresolved for as long as possible during the construction, which allows the greatest possible freedom to make a short connection.

We call this second MST-based strategy the **Kruskal-Steiner** approach. It begins with a spanning forest of n isolated components (the points of P) and repeatedly connects the closest pair of components in the spanning forest until only one component (the Steiner tree) remains. Richards [202] characterizes Kruskal-Steiner and its variants as a "folklore" heuristic; the method has also been ascribed to Thomborson by Bern [26, 27]. Variants in the literature differ primarily in their definition of the "closest pair" of components, but the example of Figure 2.7(b) is immune to these distinctions. When any variant of Kruskal-Steiner is executed on the pointset of Figure 2.7(b), it will start at the leftmost points and alternate among the middle, top, and bottom rows, adding a single horizontal to each in turn. The ϵ perturbations in Figure 2.7(b) force the alternation between rows and make the construction completely deterministic. The resulting Steiner tree will consist entirely of horizontal segments except at the left end, and its cost will be arbitrarily close to $\frac{3}{2}$ times optimal. Hwang et al. [138] note that for random instances, results are similar to those

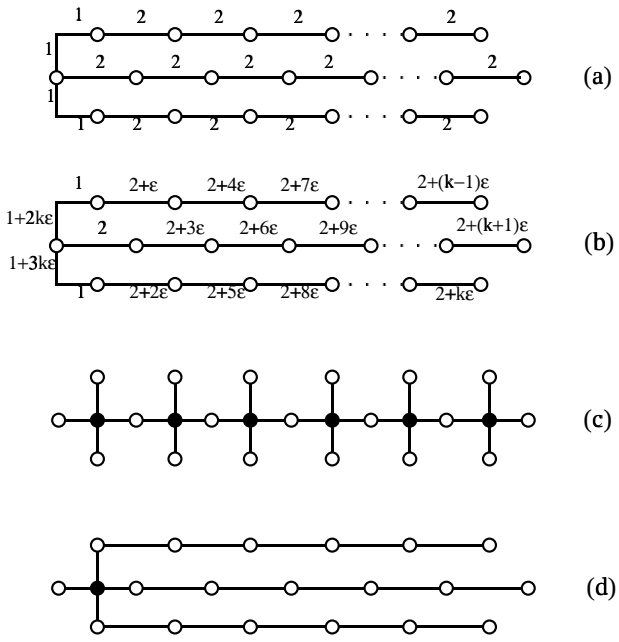


Figure 2.7 A separable MST for which $\frac{cost(MST-Overlap)}{cost(SMT)}$ is arbitrarily close to $\frac{3}{2}$. For n points, any Steiner tree derivable from the separable MSTs of (a) or (b) will have cost $2(n-2)$, while the SMT (c) has cost $\frac{4}{3}(n-1)$, yielding a performance ratio arbitrarily close to $\frac{3}{2}$ for large enough n. In (d), we show the best possible rectilinear Steiner tree that can be produced by any MST-Overlap or Kruskal-Steiner heuristic.

obtained by MST-Overlap variants, i.e., the heuristic Steiner tree cost averages between 7% and 9% less than the MST cost.

An algorithm is said to be *greedy* if it constructs a solution by iteratively selecting the best among all remaining alternatives [189]. We now show that MST-Overlap and Kruskal-Steiner belong to a general class of greedy Steiner tree heuristics, and that the example of Figure 2.7 is pathological for this class. Recall that without loss of generality, a Steiner tree may be viewed as a minimum spanning tree over $P \cup S$, where P is the input pointset and S is the added set of Steiner points. We are interested in Steiner tree constructions

which induce new edges, and possibly new Steiner points, using the following types of *connections* within an existing spanning forest over $P \cup S$: (i) *point-point* connections between two points of P; (ii) *point-edge* connections between a point of P and an edge, which may induce up to one new Steiner point in S; and (iii) *edge-edge* connections between two edges, which may induce up to two new Steiner points in S. To reflect the fact that the embedding of a given edge is indeterminate, we say that any edge between two points of $P \cup S$ can be arbitrarily *re-embedded* by the Steiner tree construction. Figure 2.8 defines a class of Steiner tree heuristics which we call C. All heuristics $H \in C$ are greedy with respect to Manhattan edge length.

Heuristic $H \in C$: greedy Steiner tree construction
Input: n isolated components (points of P) **Output:** Rectilinear Steiner tree over P

While there is more than one connected component **Do**
  Select a connection type $\tau \in$ { point-point, point-edge, edge-edge }
  Connect the *closest* pair of components greedily with respect to τ
  Optionally at any time, **Re-embed** any edge within its bounding box
  Optionally at any time, **Remove** redundant (overlapped) edge segments
Output the single remaining component

Figure 2.8 The class C of greedy Steiner tree heuristics.

Theorem 2.2.1 *Every $H \in C$ has performance ratio arbitrarily close to $\frac{3}{2}$.*

Proof: The MST of the pointset depicted in Figure 2.7(b) is unique since all interpoint distances < 3 are unique. Thus, all connections in the MST are horizontal point-point connections except for exactly two connections, one from the top row to the middle row and one from the middle row to the bottom row. The greedy routing of every edge but these two is unique since all edges except these two have degenerate bounding boxes. No improvement is possible by edge re-embedding within these degenerate bounding boxes. Therefore, no heuristic in C can do better than the result in Figure 2.7(d). The optional re-embedding within the two non-degenerate bounding boxes is negligible as n grows large, hence the performance ratio is arbitrarily close to $\frac{3}{2}$. □

There are many heuristics in the literature with previously unknown performance ratio, which by Theorem 2.2.1 have performance ratio arbitrarily close to $\frac{3}{2}$. Greedy Kruskal-like constructions include the methods of [136] and [165],

in addition to the methods described by Bern [26, 27], Gadre et al. [103], Richards [202] and Servit [215]. Algorithms which start with an initial MST and then overlap edges within their bounding boxes, such as those of [118] and [124], also belong to C: an MST can be constructed using only point-point connections, and the optional re-embedding is then used to induce edge overlaps. Exponential-time methods can also belong to the class C, notably the suboptimal branch-and-bound method of Yang and Wing [250]. Theorem 2.2.1 implies that all of these methods have the same worst-case error bound as the simple MST.

The counterexample of Figure 2.7 also establishes lower bounds arbitrarily close to $\frac{3}{2}$ for the performance ratios of several heuristics not in C, such as the three-point connection methods of Hwang [137], Lee, Bose and Hwang [164], and the Delaunay triangulation-based method of Smith, Lee and Liebman [220]. This is easy to verify using the pointset in Figure 2.7(b): as with the heuristics in C, these latter methods are severely constrained by the nature of the unique minimum spanning tree. Finally, we note that De Souza and Ribiero [72] construct an instance similar to that of Figure 2.7 and also discuss the worst-case performance of several rectilinear Steiner tree heuristics. Shute [218] gives a somewhat less general construction, also with the goal of showing a $\frac{3}{2}$ performance ratio for MST-like heuristics.

2.2.2 Counterexamples in Higher Dimensions

The rectilinear SMT problem remains well-defined when the points of P are located in d-dimensional Manhattan space with $d > 2$. Most heuristics, including those in the class C defined above, readily extend to higher dimensions. However, the construction of Figure 2.7 also extends to d dimensions, where it again provides a lower bound for the performance ratio of heuristics in C. In d dimensions, the Figure 2.7 construction generalizes to $n = k(2d-1)+1$ points, for any given k. As Figure 2.9 illustrates for $d = 3$, the cost of the optimal Steiner tree is at most $\frac{2d(n-1)}{2d-1}$; the cost of the (unique, separable) MST is $2(n-1)$; and the cost of the best Steiner tree obtainable from the MST by edge-overlapping is $2(n-d)$. Thus, in d dimensions the performance ratio of a heuristic in class C will be arbitrarily close to $\frac{2d-1}{d}$. This slightly improves on the previous lower bound of $\frac{2(d-1)}{d}$ given by Foulds [97] for the performance ratio of the MST approximation in d dimensions.

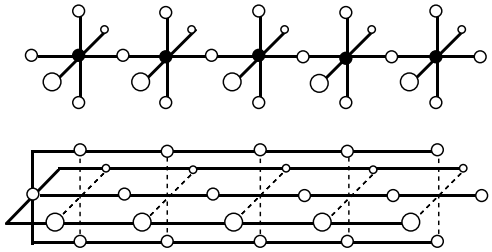


Figure 2.9 For $d = 3$, the SMT (top) has cost $\frac{6}{5}(n-1)$, while any Steiner tree derivable from the MST by re-embedding edges (bottom) has cost $2(n-3)$, yielding performance ratio arbitrarily close to $\frac{5}{3}$ as n grows large.

2.3 ITERATED 1-STEINER (I1S)

We now develop an effective SMT heuristic that avoids analogies to traditional MST constructions. The approach is greedy: we iteratively find optimum *single* Steiner points for inclusion into the pointset.

Given two pointsets A and B, we define the *MST savings of B with respect to A* as

$$\Delta MST(A,B) = cost(MST(A)) - cost(MST(A \cup B)).$$

Recall that $H(P)$ denotes the Steiner candidate set, i.e., the set of intersection points of all horizontal and vertical lines passing through points of P. For any pointset P, a *1-Steiner point of P* is a point $x \in H(P)$ which maximizes $\Delta MST(P, \{x\}) > 0$. Starting with a pointset P and a set $S = \emptyset$ of Steiner points, the *Iterated 1-Steiner* (I1S) method repeatedly finds a 1-Steiner point x of $P \cup S$ and sets $S \leftarrow S \cup \{x\}$. Note that the stated initial conditions of the algorithm imply that the Steiner candidate set $H(P \cup S)$ at each iteration will be identical to $H(P)$. The cost of $MST(P \cup S)$ will decrease with each added point, and the construction terminates when there no longer exists any point x with $\Delta MST(P \cup S, \{x\}) > 0$.

While there is always an optimal Steiner tree with at most $n-2$ Steiner points (this follows from simple degree arguments [109]), I1S can add more than $n-2$ Steiner points. Therefore, at each step we eliminate any extraneous Steiner points which have degree ≤ 2 in the MST over $P \cup S$. Figure 2.10 describes the algorithm formally, and Figure 2.11 illustrates a sample execution. This

method was first described in [150, 151, 203]. Minoux [183] has independently described an algorithm similar to I1S for the Steiner problem in graphs.

Algorithm Iterated 1-Steiner (I1S)
Input: A set P of n points **Output:** A rectilinear Steiner tree over P

$S = \emptyset$
While Cand_set $= \{x \in H(P \cup S) \mid \Delta MST(P \cup S, \{x\}) > 0\} \neq \emptyset$ **Do**
  Find $x \in$ Cand_set which maximizes $\Delta MST(P \cup S, \{x\})$
  $S = S \cup \{x\}$
  Remove points in S which have degree ≤ 2 in $MST(P \cup S)$
Output MST$(P \cup S)$

Figure 2.10 The Iterated 1-Steiner (I1S) algorithm.

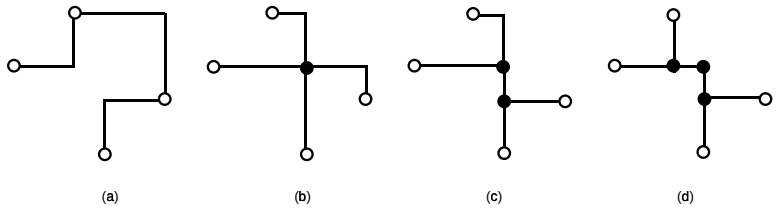


Figure 2.11 Example of the execution of Iterated 1-Steiner (I1S). Note that in step (d) a degree-2 Steiner point results; I1S will eliminate this point from the topology.

To find a 1-Steiner point, it suffices to construct an MST over $|P \cup S| + 1$ points for each of the $O(n^2)$ members of the Steiner candidate set, and then pick a candidate which minimizes the MST cost. This follows from a perturbative argument similar to that used by Hanan. Each MST computation can be performed in $O(n \log n)$ time [195], yielding an $O(n^3 \log n)$ time method to find a single 1-Steiner point. A more efficient algorithm presented in the next section finds a new 1-Steiner point in $O(n^2)$ time. A linear number of Steiner points can therefore be found in $O(n^3)$ time, and solutions with a bounded number of $\leq k$ Steiner points require $O(kn^2)$ time.

2.3.1 Finding 1-Steiner Points Efficiently

Georgakopoulos and Papadimitriou [107] give an $O(n^2)$ method for computing a 1-Steiner point in the Euclidean plane. Their method can be adapted to Manhattan geometry, via the following sequence of observations (see [107] for a more detailed account).

- Observe that a point $p \in P$ cannot have two neighbors in $MST(P)$ which lie in the same octant of the plane with respect to p. (The octants of the plane with respect to p are defined by passing lines through p with slope 0, 1, ∞ and -1.)
- Observe that two directions θ_1 and θ_2 in the plane, together with a point location x, define a cone $C(x, \theta_1, \theta_2)$. For any $p \in P$, the set of all x such that p is the closest point to x in the set $P \cap C(x, \theta_1, \theta_2)$ forms a (possibly unbounded) polygon known as an *oriented Dirichlet cell*. For fixed θ_1 and θ_2, the oriented Dirichlet cells over all points of P will partition the region of the plane that lies "in front of" the pointset P with respect to the directions θ_1 and θ_2 (see Figure 2.12). The eight pairs of directions θ_1, θ_2 that define the octants of the plane will define eight plane partitions.

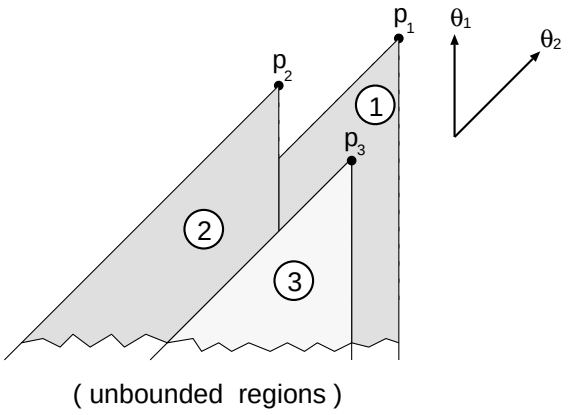


Figure 2.12 The oriented Dirichlet cells with respect to directions θ_1 and θ_2 for three points. In this example, all three regions of the planar partition are unbounded.

- These eight plane partitions can be computed and superposed to yield a "common partition" of the plane within $O(n^2)$ time. It can be shown that the $O(n^2)$ regions of the common partition possess the so-called *isodendral* property: the topology of $MST(P \cup \{x\})$ is constant for all points x within any given region. However, we need only know that for x in any given region, the common partition indicates the set of (≤ 8) possible neighbors of x in $MST(P \cup \{x\})$.

- $MST(P)$ can be constructed in $O(n^2)$ time, and by performing $O(n^2)$ preprocessing we can update the MST to include any new point $x \notin P$ in *constant* time. This is accomplished by precomputing, for every edge $e \in \{P \times P\}$ not in $MST(P)$, the shortest edge in the unique cycle formed when e is added into the tree. When x is added into the spanning tree, it will effectively introduce an "edge" between each pair of its neighbors; the precomputation allows edges to be deleted from $MST(P)$ as appropriate.

- Finally, the essence of the method is as follows. (1) If we know that the new Steiner point x is to be located in a given region of the common partition, we already know the (≤ 8) possible neighbors of x in $MST(P \cup \{x\})$. (2) Notice that *some* subset of these possible neighbors will actually be adjacent to x in $MST(P \cup \{x\})$, and there are $O(1)$ such subsets. (3) We simply try every subset of possible neighbors: for each, we can find the optimal location of x in $O(1)$ time (since this is a Steiner instance of bounded size), and we can also check the resulting cost savings when x is added to the MST in $O(1)$ time by virtue of the preprocessing. (4) Recalling that there are only $O(n^2)$ regions in the common partition, we can return the lowest-cost MST over the points in $P \cup \{x\}$, using a total of $O(n^2)$ time. Thus, the total time for all phases is $O(n^2)$.

A linear number of iterations will imply $O(n^3)$ overall time complexity. In practice, for uniformly random pointsets the number of iterations performed by I1S averages less than $\frac{n}{2}$.

2.3.2 The I1S Performance Ratio

In this section, we first completely characterize the class of instances having $\frac{cost(MST)}{cost(SMT)} = \frac{3}{2}$, and then show that I1S will always find a 1-Steiner point for such instances. Thus, the output of I1S can never be as bad as $\frac{3}{2}$ times optimal. We also show that for this class of "difficult" instances, I1S has performance ratio $\leq \frac{4}{3}$.

Lemma 2.3.1 *Any pointset P with $|P| \leq 3$ has $\frac{cost(MST)}{cost(SMT)} \leq \frac{4}{3}$.*

Proof: For $|P| = 2$, $\frac{cost(MST(P))}{cost(SMT(P))} = 1$. For $|P| = 3$ we have $cost(SMT(P)) = \frac{R}{2}$, where R is the perimeter of the bounding box of P. On the other hand, the pigeonhole principle implies $cost(MST(P)) \leq \frac{2}{3}R$. If two points of P lie on the rectangle that defines the bounding box, then $cost(MST(P)) = cost(SMT(P))$. If three points of P lie on this rectangle, then removing the largest segment of the bounding box perimeter that lies between two points of P will leave a spanning tree over P having cost at most $\frac{2}{3}R$. It follows that $\frac{cost(MST(P))}{cost(SMT(P))} \leq \frac{\frac{2}{3}R}{\frac{R}{2}} = \frac{4}{3}$. □

Definition: A *plus* is an SMT over 4 points $\{(x-r, y), (x+r, y), (x, y-r), (x, y+r)\}$ with exactly one Steiner point at the *center* (x, y) of the plus.

Lemma 2.3.2 *For $|P| = 4$ and $|S| = 1$, a plus is the only configuration that achieves a performance ratio $\frac{cost(MST)}{cost(SMT)}$ of* exactly $\frac{3}{2}$.

Proof: If $SMT(P)$ has one degree-three Steiner point, then we have the performance ratio $cost(MST(P))/cost(SMT(P)) < 3/2$. Thus, $SMT(P)$ must have the same *topology* as a plus. Since the possibility of overlapping wire would imply at least two Steiner points, the pointset must have coordinates of form $P = \{(x - h_1, y), (x + h_2, y), (x, y - v_1), (x, y + v_2)\}$. Let R again denote the perimeter of P's bounding box. $SMT(P)$ has cost $\frac{R}{2}$, while a pigeonhole argument implies that $MST(P)$ has cost $\leq R - \frac{1}{4}R$ (we obtain a spanning tree by deleting the longest of the four edges comprising the bounding box). This implies that $\frac{cost(MST(P))}{cost(SMT(P))} \leq \frac{3}{2}$ with equality holding only when the longest edge length around the bounding box is not greater than $\frac{1}{4}R$, i.e., all four edges around the bounding box have equal length. Therefore, $h_1 = h_2$ and $v_1 = v_2$. We write $h = h_1 = h_2$ and $v = v_1 = v_2$, and without loss of generality assume that $h \leq v$. Then:

$$\frac{cost(MST(P))}{cost(SMT(P))} = \frac{2(v + h) + 2h}{2(v + h)} = 1 + \frac{h}{v + h} \leq \frac{3}{2}$$

with equality holding when $h = v$. □

Definition: A *union of pluses* is a Steiner tree with $|S| = k$ and $|P| = 3k + 1$, and with exactly four edges of equal length incident to any Steiner point.

Theorem 2.3.3 *Any planar pointset having* $\frac{cost(MST)}{cost(SMT)} = \frac{3}{2}$ *has an SMT which is a union of pluses.*

Proof: Recall from the proof of Theorem 2.1.1 that any pointset P has an SMT that is composed of connected components, each of which has all its Steiner points forming a chain. Recall also that all the Steiner points on any such chain are collinear, with the possible exception of the Steiner point at the end of the chain (Figure 2.3). Using the same upper bound for MST cost and the exact expression for SMT cost as in the Theorem 2.1.1 proof, we can equate expressions for $\frac{2}{3} \cdot cost(MST)$ and $cost(SMT)$ for the points of any chain:

$$R \cdot (\frac{1}{2} + \frac{2}{3} \cdot \Theta) = R \cdot (\frac{1}{2} + \Theta)$$

where R is the perimeter of the bounding box of the points in the chain, and Θ is defined such that $R \cdot \Theta$ is equal to the sum of the distances from all (except the last) points of P to their adjacent Steiner points in the chain. The above equality implies that $\Theta = 0$, and hence all but one of the original points have the same coordinates as their adjacent Steiner points, a contradiction unless there is only one Steiner point (i.e., the last) in this chain. From Lemma 2.3.2, any chain which has only one Steiner point and which exactly achieves the $\frac{3}{2}$ ratio must be a plus. Therefore, any SMT which exactly achieves the $\frac{3}{2}$ ratio is decomposable as a union of pluses. □

Theorem 2.3.3 completely characterizes the pointsets for which $\frac{cost(MST)}{cost(SMT)}$ is *exactly* equal to $\frac{3}{2}$.

Theorem 2.3.4 *The performance ratio of I1S is* $< \frac{3}{2}$.

Proof: If $\frac{cost(MST(P))}{cost(SMT(P))} < \frac{3}{2}$, then even if I1S does not find any Steiner points, it will have performance ratio $< \frac{3}{2}$. From Theorem 2.3.3, any P for which $\frac{cost(MST(P))}{cost(SMT(P))} = \frac{3}{2}$ will have $SMT(P)$ that is a union of pluses; in such a case I1S will select and add the center of some plus at the first iteration, yielding performance ratio strictly less than $\frac{3}{2}$. To see this, note that a spanning tree with cost $\frac{3}{2} \cdot cost(SMT(P))$ is obtained simply by replacing each plus in $SMT(P)$ by an arbitrary spanning tree over the four points of P in the plus (see Figure 2.13).

(The center of the plus is one of the Steiner candidates considered during the first iteration of I1S. Even if there are other Steiner candidates within the

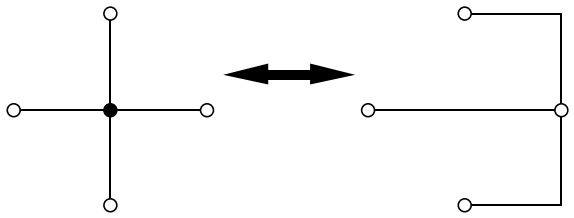


Figure 2.13 Locally replacing each plus (left) with an MST over the four points in P (right).

convex hull of the four points of the plus, the center gives the greatest possible cost savings of exactly one-third.) □

Theorem 2.3.5 *For pointsets P with $SMT(P)$ a union of pluses, the performance ratio of I1S $\leq \frac{4}{3}$.*

Proof: When I1S selects the center of a plus as a 1-Steiner point, at most three centers of other pluses are excluded from future selection. By the greedy selection rule of I1S, any center that is excluded belongs to a plus that induces less cost savings than the selected plus.[6] Thus, even if I1S selects a plus that is not in $SMT(P)$, the cost savings will be at least as great as the savings that would have been realized by selecting the largest of the (up to three) pluses that are now excluded due to topological constraints (see Figure 2.14).

Each plus represents a savings of $\frac{1}{3}$ of the MST cost over the points of P in the plus, so even if we use simple MST edges to connect the remaining affected points to the selected plus, the total heuristic cost is no more than $cost(MST) - \frac{1}{3} \cdot \frac{1}{3} \cdot cost(MST) = \frac{8}{9} \cdot cost(MST)$. Therefore, the performance ratio of I1S is no greater than $\frac{\frac{8}{9} \cdot cost(MST)}{\frac{2}{3} \cdot cost(MST)} = \frac{4}{3}$. □

This bound can likely be tightened by more exhaustive case analysis. Since most signal nets in VLSI designs have six or fewer terminals, we briefly discuss known I1S performance bounds for small values of $|P|$.

Theorem 2.3.6 *I1S is optimal for $|P| \leq 4$ points.*

[6] The cost savings of a plus are with respect to the MST over the four points in the plus. These savings are proportional to the "size" of the plus: larger pluses induce greater savings.

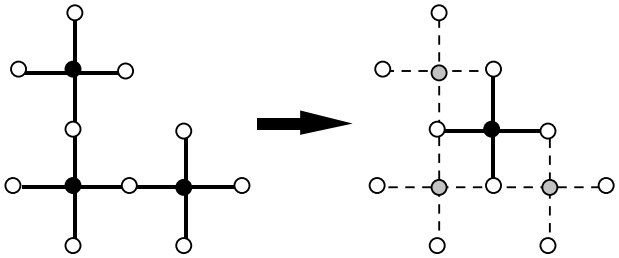


Figure 2.14 Each selected 1-Steiner point may exclude at most three potential 1-Steiner points from future selection; thus at least $\frac{1}{3}$ of the maximum possible savings is achieved.

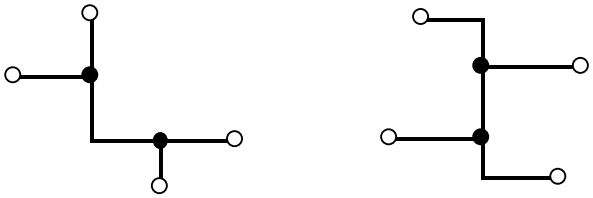


Figure 2.15 The two possible Steiner tree topologies on 4 points.

Figure 2.16 A 5-point example where the I1S performance ratio is $\frac{7}{6}$. The optimal SMT (left) has cost 6, while the (possible) heuristic output (right) has cost 7.

Proof: When $SMT(P)$ has less than two Steiner points, I1S is optimal since it examines all candidates. For $|P| = 3$, there can be at most one Steiner point. For $|P| = 4$ and $|S| = 2$, Hwang [135] showed that an SMT must have one of the two topologies shown in Figure 2.15. A case analysis shows that I1S always selects both Steiner points. □

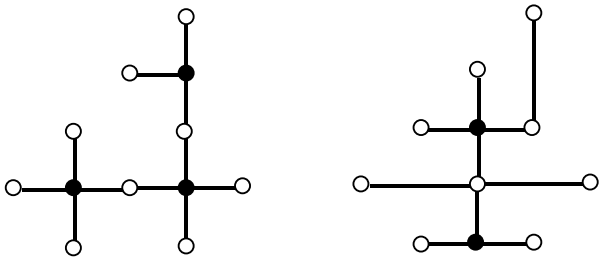


Figure 2.17 A 9-point example where the I1S performance ratio is $\frac{13}{11}$; the optimal SMT (left) has cost 11, while the (possible) heuristic output (right) has cost 13.

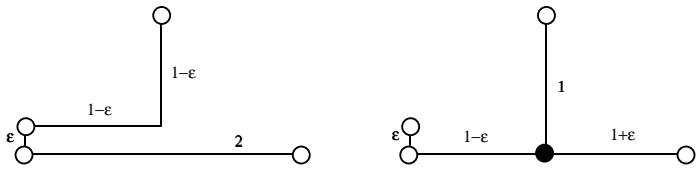


Figure 2.18 A 4-point instance on which MST-based heuristics perform arbitrarily close to $\frac{4}{3}$ times optimal (left); the (optimal) I1S solution is also shown (right).

In contrast to I1S, MST-based methods are generally not optimal for $|P| = 4$; Figure 2.18 shows that performance ratios approaching $\frac{4}{3}$ are possible. As shown in Figure 2.16, the worst-case performance ratio of I1S for $|P| \leq 5$ is conjectured to be $\frac{7}{6}$. Figure 2.17 shows a 9-point instance on which the I1S tree cost is $\frac{13}{11}$ times optimal.

Finally, an elegant iterated construction due to Berman, Fössmeier Karpinski, Kaufmann and Zelikovsky [24, 255] shows that the performance ratio of I1S has a lower bound of 1.3. Figure 2.19 reproduces the construction, which has point coordinates as follows:

- $a_i = (4 \cdot 4^i, 0), i = 0, ..., k$
- $a'_i = (0, 4 \cdot 4^i), i = 0, ..., k$

- $b_i = (2 \cdot 4^i, -4^{i-1}), i = 1, ..., k$
- $b'_i = (-4^{i-1}, 2 \cdot 4^i), i = 1, ..., k$
- $c_i = (3 \cdot 4^i, 4^{i-1}), i = 1, ..., k$
- $c'_i = (4^{i-1}, 3 \cdot 4^i), i = 1, ..., k$

For any value of k, this construction yields an instance whose SMT consists of conjoined Steiner minimal trees over the sets $\{a_{i-1}, b_i, c_i, a_i\}$ and $\{a'_{i-1}, b'_i, c'_i, a_i\}$, along with the edge (a_0, a'_0); this SMT has cost $10\frac{4^k-1}{3} + 2$. I1S will return 1-Steiner points that are adjacent to triples of points $\{a_{i-1}, c_i, c'_i\}$, which implies cost $13\frac{4^k-1}{3} + 2$. Thus, the construction establishes the lower bound on performance ratio of

$$\lim_{k \to \infty} \frac{13\frac{4^k-1}{3} + 2}{10\frac{4^k-1}{3} + 2} = 1.3.$$

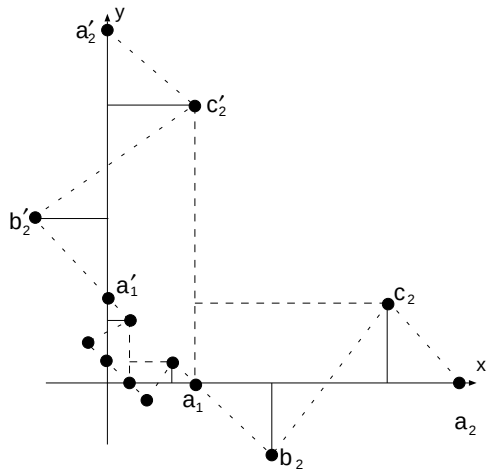


Figure 2.19 The construction of Berman et al. which establishes a lower bound of 1.3 on the I1S performance ratio.

2.3.3 The Method of Zelikovsky

Berman and Ramaiyer [25], and Zelikovsky and coauthors [24, 95, 253, 255], have recently developed several SMT heuristics that are similar to I1S, and have performance ratios substantially less than $\frac{3}{2}$. These methods derive from a breakthrough technique developed by Zelikovsky for the SMT problem in graphs [254]. The results of [25, 253] in 1992 settled in the affirmative the long-standing open question of whether there exists a polynomial-time rectilinear SMT heuristic with performance ratio $< \frac{3}{2}$.[7] Here, we briefly review key ideas in this sequence of works, following the discussion of [24].

Given a Steiner tree T over pointset P, any subtree T' is a *full Steiner component* if every point of P in T' has degree one. As noted in the earlier discussion of Hwang's theorem, any Steiner tree T can be partitioned into edge-disjoint full Steiner components. (Recall that Figure 2.3 showed the two possible types of full Steiner components.) A Steiner tree T is *k-restricted* if each of its full Steiner components has at most k leaves. Thus, for example, an optimal 2-restricted tree over P is exactly an MST over P. We may use t_k to denote the cost of the minimum-cost k-restricted Steiner tree over P; thus, for example, $t_2 = cost(MST(P))$ and $t_n = cost(SMT(P))$. Hwang's theorem states that $t_2 \leq \frac{3}{2}t_n$; Zelikovsky [253] showed that $t_3 \leq \frac{5}{4}t_n$; and Berman and Ramaiyer [25] showed that $t_k \leq \frac{2k-1}{2k-2}t_n$. Zelikovsky pioneered the approach of approximating the optimal k-restricted Steiner tree for some small value of k, as opposed to approximating the SMT itself.

For expository reasons, we will begin by describing the heuristic of Berman and Ramaiyer, which is called $\mathcal{A}_k$; the time complexity and performance ratio of $\mathcal{A}_k$ depends on the value of k. For $k = 3$, $\mathcal{A}_k$ has performance ratio $\frac{11}{8}$ and time complexity $O(n^{3.5})$. The heuristic $\mathcal{A}_k$ begins with some $MST(P)$, then considers all optimal Steiner trees over subsets of P of size k or less. $\mathcal{A}_k$ is similar to I1S, in that it will consider adding Steiner points one at a time from the Hanan candidate set, and in that it uses some measure of cost improvement to evaluate the utility of each candidate Steiner point. However, instead of adding a new Steiner point into the tree, $\mathcal{A}_3$ replaces two edges from the current MST with two "abstract edges" having the same endpoints but reduced cost. The cost of each new abstract edge is equal to the cost of the

[7] Interestingly, we conjectured in [150, 151] that I1S has performance ratio strictly less than $\frac{3}{2}$, but could not prove this. There are clear similarities between the "batched" I1S variant that we discuss below and the method of [253], suggesting that "batched" I1S has performance ratio at most $\frac{11}{8} = 1.375$. Recently, Zelikovsky [252] has stated that I1S actually has performance ratio upper-bounded by 1.3125 (and lower-bounded by 1.3 per the construction of Figure 2.19).

edge it replaces, minus the cost improvement, or "gain", that would be achieved by inserting the new Steiner node.

Conceptually, $\mathcal{A}_3$ merges the current $MST(P)$ with the optimal Steiner tree over the $k = 3$ points of P, and updates the MST over P within the resulting graph. More specifically, let τ be the optimal Steiner tree over a given three points of P, and consider the graph formed by the union of edges in τ and $MST(P)$. This graph will contain two cycles C_1 and C_2, with the longest edge in each cycle respectively being c_1 and c_2. Removing c_1 and c_2 yields a new minimum spanning tree T' over the graph. The "gain" associated with τ is given by $gain(\tau) = cost(MST(P)) - cost(T')$. For $i = 1, 2$ an abstract edge is inserted between the two leaves of τ through which cycle C_i passed; the cost of the abstract edge is $cost(c_i) - gain(\tau)$.

In Berman and Ramaiyer's algorithm, the new abstract edges are added only if the gain value is greater than zero. Because the gain is subtracted from *both* new edges, however, the cost of the new MST is optimistically small. Beyond the consideration of candidate Steiner points in arbitrary order, this concept of "optimistic gain" is the main difference between $\mathcal{A}_3$ and I1S.

$\mathcal{A}_3$ works in two main phases: in the "evaluation" phase all $\binom{|P|}{3}$ triples of points from P are considered in arbitrary order. If adding the Steiner node for a triple τ would reduce the cost of the current MST (i.e., $gain(\tau)$ is positive), then two abstract edges are added as we have described. In the "selection" phase, triples with positive gain are considered in reverse order. If the abstract edges of a triple are still used in an MST over P in which abstract edges from *all* triples with positive gain are considered, then the Steiner point for that triple is included in the output construction.

Berman and Ramaiyer prove that in algorithm $\mathcal{A}_k$, the MST containing abstract edges has cost less than the optimal k-restricted tree. In $\mathcal{A}_3$, the cost reduction from the abstract edges is at most twice the cost reduction obtained by actually adding the new Steiner points. Hence, the performance improvement for $\mathcal{A}_3$ versus MST is at least one-half the performance improvement of the optimal 3-restricted tree. Berman and Ramaiyer establish a performance ratio r_k for the optimal k-restricted Steiner tree: $r_k \leq 1 + 1/(2k-2)$. (To prove this bound, they show how to construct $2k-2$ k-restricted Steiner trees over S such that their total cost is at most $2k-1$ times that of the minimum-cost Steiner tree.) This gives a $5/4$ performance ratio for the optimal 3-restricted tree, and an $\frac{11}{8}$ performance ratio for $\mathcal{A}_3$.

Zelikovsky's method [253] is greedier than that of Berman and Ramaiyer, and is extremely similar to the "batched" variant of I1S that we discuss below. Zelikovsky's method finds and incorporates the triple τ with largest $gain(\tau)$, adding three zero-cost edges between pairs of leaves of τ into the graph noted above. The largest-gain triple is found in the new graph, and the process terminates when there is no remaining triple with positive gain. A performance ratio of $\frac{11}{8}$ was shown in [253].

Subsequent work has improved on the $O(n^{3.5})$ time complexity of $\mathcal{A}_3$. An $\frac{11}{8}$ performance bound with an $O(n^{1.5})$ implementation was achieved by Fössmeier et al. [95], who show that only a linear number of triples need to be considered in $\mathcal{A}_3$. More recently, the five authors of [24] and [255] have together shown that Zelikovsky's algorithm has performance ratio between 1.3 and 1.3125, and that Berman and Ramaiyer's algorithm has performance ratio at most 1.271; the latter algorithm can also be implemented to run in $O(n \log^2 n)$ time.

2.4 ENHANCING I1S PERFORMANCE

In this section, we discuss variations of the I1S approach that can yield lower-cost solutions or runtime reductions in practice. These variations include an amortization of the 1-Steiner point computation via addition of an entire set of "independent" or "non-interfering" 1-Steiner points in a single iteration, as well as a perturbative variant.

2.4.1 A Batched Variant

Although a single 1-Steiner point may be found in $O(n^2)$ time, the required computational geometry techniques have large hidden constants in their time complexities and are difficult to implement. We now describe a *batched* I1S variant which amortizes some of the computational expense by adding an entire set of "independent" Steiner points in a single *round*.

The *Batched 1-Steiner* (B1S) variant computes $\Delta MST(P, \{x\})$ for each candidate Steiner point $x \in H(P)$. Two candidate Steiner points x and y are *independent* if

$$\Delta MST(P, \{x\}) + \Delta MST(P, \{y\}) \leq \Delta MST(P, \{x, y\}),$$

i.e., introducing one does not reduce the potential MST cost savings of the other. Given pointset P and a set of Steiner points S, each round of B1S greedily selects a maximal independent set of Steiner points, then adds this set to S. The algorithm terminates when a round fails to produce a new Steiner point. A single round of B1S is described as follows:

- In $O(n \log n)$ time, compute both $MST(P)$ and the Delaunay triangulation [219] over P (the Delaunay triangulation is the geometric dual of the Voronoi tesselation of the plane).
- Compute the $O(n^2)$ isodendral regions over P, and for each region determine the $O(1)$ potential neighboring points in the MST as in [107]. This requires a total of $O(n^2)$ time.
- Using $O(n^2 \log n)$ time and $O(n^2 \log n)$ space, preprocess the $O(n^2)$ isodendral regions, now treated as a planar subdivision, so that determining the region in which a given point lies may be performed in $O(\log n)$ time. This is the problem of *planar subdivision search* [195].
- For each candidate Steiner point x, compute $\Delta MST(P, \{x\})$. Determine the isodendral region to which x belongs via $O(\log n)$ time planar subdivision search, and let X be the set of potential MST neighbors of x. For each subset $Y \subseteq X$, add the weighted edge set $\{(x, y) \mid y \in Y\}$ to the graph G. The MST of a planar weighted graph can be maintained dynamically using $O(\log n)$ time per addition/insertion of a point or edge [88]. Since $|X| = O(1)$ and therefore $|Y| = O(1)$, we can determine in $O(\log n)$ time the MST cost savings for each candidate Steiner point. Since there are $O(n^2)$ candidate Steiner points, the total time for this step is $O(n^2 \log n)$.
- Sort the $O(n^2)$ Hanan candidates in order of decreasing MST cost savings; this requires $O(n^2 \log n)$ time using any efficient sorting algorithm.
- Determine a maximal set S of independent candidate Steiner points to be added during this round, by greedily adding candidates in order of decreasing MST cost savings as long as each added Steiner point is independent of all Steiner points previously added during this round. In other words, for an original pointset P, a set of already added candidate points S, and a new candidate x, add x to S if and only if $\Delta MST(P, \{x\}) \leq \Delta MST(P \cup S, \{x\})$. Again, MST cost savings due to the addition or deletion of a single point can be determined in time $O(\log n)$ [88], bringing the total time for this entire step to $O(n^2 \log n)$.

The total time required for each round is $O(n^2 \log n)$. The resulting B1S algorithm is summarized in Figure 2.21. Empirical data indicates that the number of rounds required grows much more slowly than the number of Steiner points produced. For example, on pointsets of size 300, B1S produces an average of 138 Steiner points (with a maximum of 145), while the average number of rounds is only 2.5 (with a maximum of 4); see Section 2.8 for more details. We conjecture that the number of rounds grows sub-linearly with the number of points.

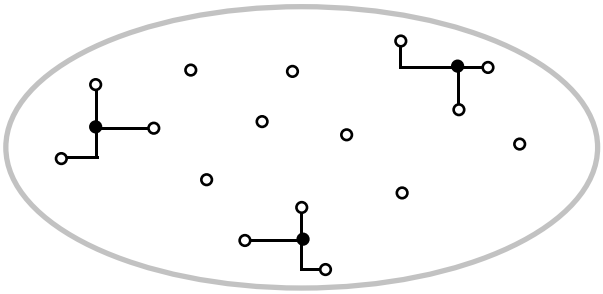


Figure 2.20 The Batched 1-Steiner heuristic: selecting a maximal independent set of candidate Steiner points in one round.

Algorithm Batched 1-Steiner (B1S)

Input: A set P of n points
Output: A rectilinear Steiner tree over P

While $T = \{x \in H(P) | \Delta MST(P, \{x\}) > 0\} \neq \emptyset$ **Do**
  $S = \emptyset$
  For $x \in \{T$ in order of non-increasing $\Delta MST\}$ **Do**
    If $\Delta MST(P \cup S, \{x\}) \geq \Delta MST(P, \{x\})$ **Then** $S = S \cup \{x\}$
  $P = P \cup S$
  Remove from P Steiner points with degree ≤ 2 in MST(P)
Output MST(P)

Figure 2.21 The Batched 1-Steiner (B1S) algorithm.

Empirical studies indicate that only a small fraction of the Hanan candidates have positive MST savings in a given B1S round. Furthermore, candidates with positive MST savings in an earlier round are more likely to produce positive MST savings in subsequent rounds. Therefore, rather than examine the MST

savings of all Hanan candidates in a given round, subsequent rounds may consider only the candidates that produced positive savings in the previous round. In practice, this strategy significantly reduces the time spent during each round without substantially affecting the solution quality.

2.4.2 A Perturbative Variant

At each iteration, I1S selects a 1-Steiner point greedily. This may preclude additional savings in subsequent iterations. Suboptimalities may also occur due to tie-breaking among 1-Steiner points that induce equal savings. The examples of Figures 2.16 and 2.17 show that an unfortunate choice of a 1-Steiner point can result in a suboptimal solution.

Empirical tests indicate that multiple 1-Steiner points (i.e., points in $H(P)$ with equal MST savings) occur quite often. To avoid a deterministic tie-breaking rule that could preclude possible future savings, we may *randomly* select one of the 1-Steiner candidates and proceed with the algorithm execution. It is reasonable to then run this randomized I1S variant m times on a given input, where m is a user-defined parameter, and select the best of the m solutions.

To further avoid possible shortcomings of a deterministic greedy strategy, we also propose a mechanism that allows I1S to select as the 1-Steiner point any $x \in H(P)$ whose MST cost savings is within δ of the best candidate's cost savings; again, δ is a user-supplied parameter. This strategy would enable a slightly suboptimal choice which could perhaps enable greater overall savings in future iterations.

Finally, performance may be improved if instead of looking for 1-Steiner points, we search for *pairs* of Steiner candidates that offer maximum savings with respect to other candidates or pairs of candidates. Such a 2-*Steiner algorithm* would optimally solve the pointset of Figure 2.16. In general, a k-Steiner algorithm will search for sets of k candidate Steiner points which maximize the MST cost savings.

Combining the three techniques of (i) non-deterministic tie-breaking, (ii) near-greedy search, and (iii) k-Steiner selection, we obtain a Perturbative Iterated k-Steiner algorithm (PIkS), as detailed in Figure 2.22. Note that I1S is equivalent to PIkS with $k = 1$, $m = 1$, and $\delta = 0$. The PIkS scheme can be further extended using an "independence" criterion as in Section 2.4.1 to yield a Per-

turbative Batched k-Steiner algorithm (PBkS), where a maximal number of Steiner points are added during each round.

Algorithm Perturbative Iterated k-Steiner (PIkS)
Input: A set P of n points, integer parameters $\delta \geq 0$, $k \geq 1$, and $m \geq 1$ **Output:** A rectilinear Steiner tree over P

$T = MST(P)$
Do m times
  $S = \emptyset$
  While $C = \{X \subseteq H(P) \mid \|X\| \leq k, \Delta MST(P \cup S, X) > 0\} \neq \emptyset$ **Do**
  Find $Y \in C$ with maximum $\Delta MST(P \cup S, Y)$
  Randomly select $Z \in C$ with $\Delta MST(P \cup S, Z) > \Delta MST(P \cup S, Y) - \delta$
  $S = S \cup Z$
  Remove from S points with degree ≤ 2 in $MST(P \cup S)$
  If $cost(MST(P \cup S)) < cost(T)$ **Then** $T = MST(P \cup S)$
Output T

Figure 2.22 The Perturbative Iterated k-Steiner (PIkS) method.

For applications to multi-layer routing and three-dimensional VLSI structures, PIkS extends to the case of points lying on L parallel planes. The general three-dimensional SMT problem corresponds to $L \to \infty$, and the planar formulation corresponds to $L = 1$. The different costs of routing between layers and routing on a given fixed layer may be modeled by varying the distance between the parallel planes.

In three dimensions, PIkS exploits the generalization of Hanan's theorem to higher dimensions [223], namely, that there always exists an optimal Steiner tree whose Steiner points are chosen from the $O(n^3)$ intersections of all planes that are orthogonal to some coordinate axis and pass through a point of P. The three-dimensional analog of Hwang's result suggests that the Steiner ratio, i.e. the maximum $\frac{cost(MST)}{cost(SMT)}$ ratio for three dimensions is at most $\frac{5}{3}$; however, there is no known proof of this. An example consisting of six points located in the middle of the faces of a rectilinear cube establishes that $\frac{5}{3}$ is a lower bound for the Steiner ratio in three dimensions.

2.4.3 Parallel Implementation

The I1S and B1S algorithms are highly parallelizable since each one of p processors can compute the MST savings of $O(\frac{n^2}{p})$ candidate Steiner points. We have undertaken a parallel implementation of I1S, where all processors send their best candidate to a master processor, which selects the best of these candidates for inclusion into the pointset. This procedure is iterated until no improving candidates can be found. The Parallel Virtual Machine (PVM) system [230] was used for initiating processes on other machines and for controlling synchronization and communication among processes.[8]

2.5 PRACTICAL IMPLEMENTATION OPTIONS FOR I1S

This section describes practical ways to reduce the time complexity of an I1S implementation. We present three techniques: (i) an incremental MST update scheme, (ii) distribution of the computation over, e.g., a network of workstations, and (iii) use of tighter bounds on the maximum rectilinear MST degree in both two and three dimensions.

2.5.1 Incremental MST Updates in Batched 1-Steiner

In computing the MST savings of each of the $O(n^2)$ Steiner candidates, a key fact is that once we have computed an MST over the pointset P, the addition of a single new point x into P induces only a constant number of changes between the topologies of $MST(P)$ and $MST(P \cup \{x\})$. This follows from the observation that each point can have at most eight neighbors in a rectilinear planar MST, i.e. at most one per octant [124]. Thus, to update an MST with respect to a newly added point x, it suffices to consider only the closest point to x in each of the eight plane octants with respect to x (below, we refine this result and show that for each point it suffices to examine at most four potential candidates for connection in the MST).

[8] Initially, the "master" processor sends equal-sized subsets of the Steiner candidate set to the available processors, and the computation/response time of each processor is tracked. If any individual processor is determined to be considerably slower than the rest, it is henceforth given smaller tasks to perform. If a processor does not complete its task within a reasonable time, it is sent an abort message, and the task is reassigned to the fastest idle processor available. This prevents individual slow (or crashed) processors from seriously impeding the overall computation. Empirical result on this parallel implementation are given in Section 2.8.

These observations suggest the following linear-time algorithm for dynamic MST maintenance: connect the new point x to each of its $O(1)$ potential neighbors (i.e, the closest point to x in each of the octants around x), and delete the longest edge on any resulting cycle. Using this dynamic MST maintenance scheme, the MST savings of each Hanan candidate can be computed in linear time, and therefore the MST savings of all $O(n^2)$ Hanan candidates may be computed in time $O(n^3)$. This method was first described in [20].

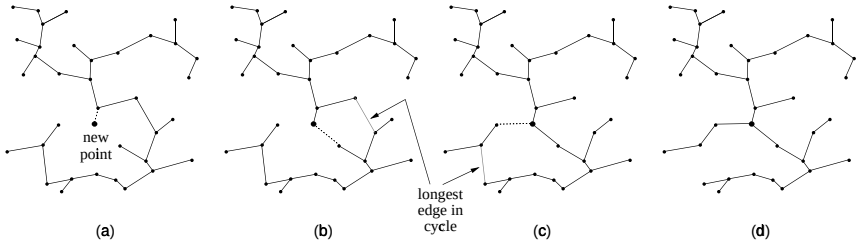


Figure 2.23 Dynamic MST maintenance: adding a point to an existing MST entails connecting the point to its closest neighbor in each octant, and deleting the longest edge on each resulting cycle. The Euclidean metric has been used for clarity in this example.

During each round of B1S we: (1) compute in $O(n^3)$ time the MST savings of all Hanan candidates, (2) sort them by decreasing MST savings in time $O(n^2 \log n)$, and (3) march down the sorted list and add into the pointset those candidates with "non-interfering" MST savings (at linear time per candidate according to our dynamic MST maintenance scheme described above). Thus, an entire round of B1S can be implemented in this straight-forward manner in time $O(n^3)$. An execution example is shown in Figure 2.23, and Figure 2.24 describes the algorithm formally. A similar method was used in [251] to obtain a sub-quadratic MST algorithm in higher dimensions, but no attempt was made to optimize the number of necessary regions.[9]

[9]Frederickson [98] has given a sublinear-time algorithm for dynamic MST maintenance, but we prefer the linear-time scheme above due to its simplicity and ease of implementation.

Dynamic MST Maintenance
Input: A set P of n points, $MST(P)$, a new point x **Output:** $MST(P \cup \{x\})$
$T = MST(P)$ **For** $i = 1$ to #regions **Do** **Find** in region $R_i(x)$ the point $p \in P$ closest to x **Add** edge (p, x) to T **If** T contains a cycle **Then** remove from T the longest edge on the cycle **Output** T

Figure 2.24 Linear-time dynamic MST maintenance.

2.5.2 MST Degree Bounds

The complexity of dynamic MST maintenance, and thus the complexity of B1S, improves when we observe that only four regions suffice for dynamic MST maintenance in the Manhattan plane. These four regions are defined by the two lines oriented at +45 and -45 degrees (Figure 2.25(a)); we call this division of the plane the *diagonal partition*. A key property for regions and partitions in dynamic MST maintenance is the *uniqueness property* [113] [204]:

The Uniqueness Property: Given a point p in d-dimensional space, a region R has the *uniqueness property* with respect to p if for every pair of points $u, w \in R$, either $d(w, u) \leq d(w, p)$ or $d(u, w) \leq d(u, p)$, where $d(u, w)$ is the distance between u and w.

A partition is said to satisfy the uniqueness property if each of its regions satisfies the uniqueness property. Any partition having the uniqueness property is useful for dynamic MST maintenance, since each region will contain at most one candidate for connection in the MST (recall the earlier use of "oriented Dirichlet cells" in the construction of Georgeakopoulos and Papadimitriou). We can show that the diagonal partition enjoys the uniqueness property.

Lemma 2.5.1 *For any point p in the Manhattan plane, the diagonal partition with respect to p has the uniqueness property.*

Proof: The two diagonal lines through p partition the plane into four disjoint regions R_1 through R_4 (Figure 2.25(a)). Points on the boundary between two

neighboring regions may be arbitrarily assigned to either region. Consider any of the four regions, say R_1, and points $u, w \in R_1$ (Figure 2.25(b)). Without loss of generality assume that $d(u,p) \leq d(w,p)$. Consider the diamond D in R_1 with one corner at p, and with u on the boundary of D (see Figure 2.25(c)). Let c be the center of D, so that c is equidistant from all points of D, and let the ray from p through w intersect the boundary of D at b. By the triangle inequality, $d(w,u) \leq d(w,b) + d(b,c) + d(c,u) = d(w,b) + d(b,c) + d(c,p) = d(w,p)$. Thus, w is not closer to p than to u, and the region has the uniqueness property. It follows that the diagonal partition has the uniqueness property. □

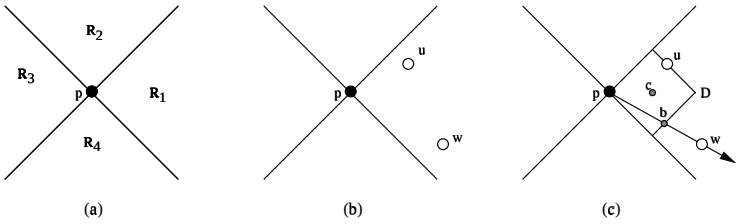


Figure 2.25 The *diagonal partition* of the plane (a) into four regions with respect to a point p has the *uniqueness property*: for every two points u and w that lie in the same region (b), either $d(w,u) \leq d(w,p)$ or else $d(u,w) \leq d(u,p)$ (c).

For any given dimension and metric, it is natural to seek an optimal partitioning scheme, i.e., one with the smallest possible number of regions. The set of five points consisting of the origin and the four corners of the diamond forces the MST to have degree four in the Manhattan plane. Thus, the diagonal partition is optimal.

Even in three dimensions, the addition of a single new point p into P can induce at most a constant number of topological changes between $MST(P)$ and $MST(P \cup \{p\})$. This follows from the fact that in any fixed dimension, each point can have at most $O(1)$ neighbors in a rectilinear MST. Therefore, the MST savings in three dimensions can be efficiently calculated by partitioning the space into $O(1)$ mutually disjoint regions R_i such that each has the uniqueness property. This would enable a linear-time procedure to compute the MST savings of a given Steiner candidate.

Using insights similar to those which led to Lemma 2.5.1, we can exhibit a partition of three-dimensional Manhattan space into 14 regions, with each re-

gion having the uniqueness property. This partition corresponds to the faces of the solid that is obtained by chopping off the corners of a cube to yield six square faces and eight equilateral triangular faces (Figure 2.26(a-b)). This solid is known as a "cuboctahedron" [177]. The 14 regions of this partition are induced by the 14 faces of the cuboctahedron, and consist of six pyramids with square cross-section (Figure 2.26(c)) and eight pyramids with triangular cross-section (Figure 2.26(d)). Again, points located on region boundaries may be arbitrarily assigned to either adjacent region. We call this partition the *cuboctahedral partition*, and refer to the two types of induced regions as *square pyramids* and *triangular pyramids*. The following theorem implies that for any given pointset P and new point p in three-dimensional Manhattan space, there exists some MST over $P \cup \{p\}$ in which p has degree ≤ 14.

Theorem 2.5.2 *Given a point p in three-dimensional Manhattan space, each of the* 14 *regions in the cuboctahedral partition with respect to p has the uniqueness property.*

Proof: Consider any of the square pyramids R with respect to p (Figure 2.26(c)), and let $u, w \in R$. Assume without loss of generality that $d(u,p) \leq d(w,p)$. Consider the locus of points $D \subset R$ at distance $d(u,p)$ from p (Figure 2.26(e)); D is the upper half of the boundary of an octahedron. Let c be the center of the octahedron determined by D, so that c is equidistant from all points of D. Let b be the intersection of the surface of D with a ray from p that passes through w. By the triangle inequality, $d(w,u) \leq d(w,b) + d(b,c) + d(c,u) = d(w,b) + d(b,c) + d(c,p) = d(w,p)$. Thus, w is not closer to p than to u, and the region R has the uniqueness property.

Next, consider any of the triangular pyramids R with respect to p (Figure 2.26(d)), and let $u, w \in R$. Assume without loss of generality that $d(u,p) \leq d(w,p)$. Consider the set of points $D \subset R$ at distance $d(u,p)$ from p (Figure 2.26(f)). Let b be the intersection of D with the ray from p that passes through w. By the triangle inequality, $d(w,u) \leq d(w,b) + d(b,u) \leq d(w,b) + d(b,p) = d(w,p)$. Thus, w is not closer to p than to u, and the region R has the uniqueness property. □

Theorem 2.5.3 *There are three-dimensional pointsets for which the maximum degree of* any *MST is at least* 13.

Proof: Consider the pointset P= {(0,0,0), (±100,0,0), (0,± 100,0), (0,0,±100), (47,-4,49), (-6,-49,45), (-49,8,43), (-4,47,-49), (-49,-6,-45), (8,-49,-43), (49,49,2)}.

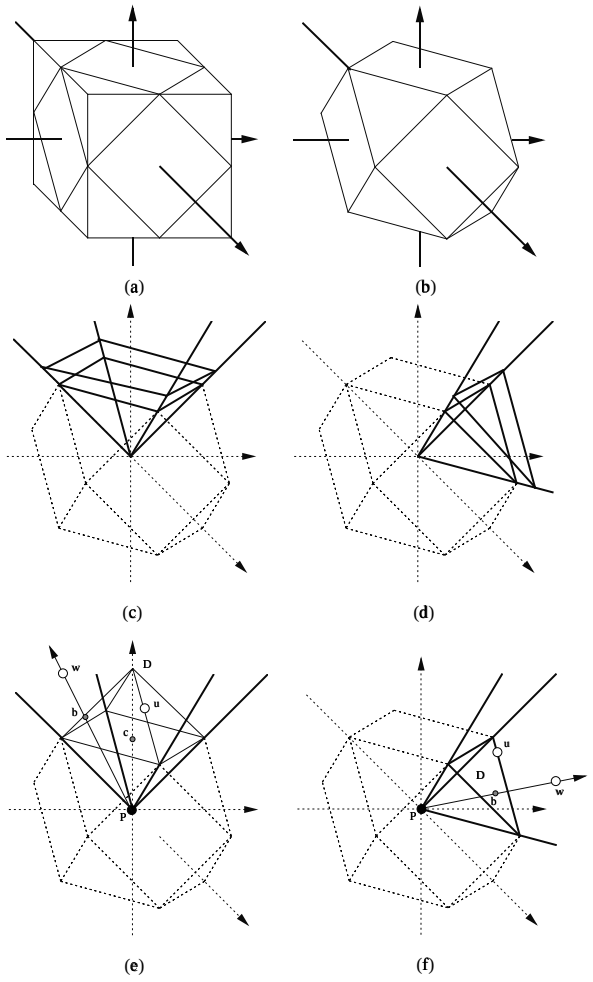


Figure 2.26 A truncated cube (a-b) induces a three-dimensional cuboctahedral space partition where each region has the uniqueness property. The 14 regions consist of six square pyramids (c), and eight triangular pyramids (d). Using the triangle inequality, the uniqueness property may be shown to hold for each region (e-f).

The distance between every point and the origin is exactly 100 units, but the distance between any two non-origin points is strictly greater than 100 units.

Therefore, the MST over P is unique (i.e., all 13 points must connect to the origin in the MST) and the origin point will have degree 13 in the MST. □

Given that each point can connect to at most 14 neighbors in the MST, linear-time dynamic MST maintenance in three dimensions is accomplished by connecting the new point in turn to each of ≤ 14 potential neighbors, then deleting the longest edge on each resulting cycle. This method was first described in [21]. It is still an open question whether for three dimensions the cuboctahedral partition is optimal (i.e., whether there exists a partition of space into 13 regions having the uniqueness property).

2.6 ON THE MAXIMUM MST DEGREE

Although the I1S algorithm described in Section 2.3.1 runs within time $O(n^3)$, the constant hidden in this asymptotic notation is *exponential* in the maximum MST degree. In this section we show that every pointset in the Manhattan plane has an MST with maximum degree ≤ 4. This result reduces the running time of the I1S implementation, and is of independent theoretical interest [204].

Even though the degree of any single node in a rectilinear MST can be made ≤ 4, Theorem 2.5.1 does not imply that the degrees of *all* nodes can be made ≤ 4 *simultaneously*. For example, decreasing the degree of one node may increase the degree of neighboring nodes. It turns out, however, that ties for connection during MST construction can always be broken appropriately so as to keep the maximum MST degree low. We begin by defining a strict version of the uniqueness property:

The Strict Uniqueness Property: Given a point p in d-dimensional space, a region R has the *strict uniqueness property* with respect to p if for every pair of points $u, w \in R$, either $d(w,u) < d(w,p)$ or $d(u,w) < d(u,p)$.

Each d-dimensional region that satisfies the strict uniqueness property may contribute at most one to the MST degree at p. Using a perturbative argument, we can prove that by breaking ties judiciously, the maximum MST degree is no larger than the number of d-dimensional regions in a partition having the strict uniqueness property.

Theorem 2.6.1 *If there exists a partition of d-dimensional space into r regions, with $r' \leq r$ of these regions being d-dimensional and satisfying the strict uniqueness property (the rest of the $r - r'$ regions are lower-dimensional, and are not required to satisfy the uniqueness property), then the maximum MST degree of any pointset in this space is r' or less.*

Proof: Given a pointset P, perturb the coordinates of each point by a tiny amount so that the lower-dimensional regions with respect to each point do not contain any other points. This is always possible to do, and yields a new perturbed pointset P'. Because interpoint distances in P' differ by only a tiny amount from the corresponding interpoint distances in P, the cost of the MSTs over P' and P will differ by only a similarly tiny amount which we can make arbitrarily small. But the MST over P' has maximum degree r', since only the r' d-dimensional regions of the partition are nonempty with respect to the points of P', and these regions satisfy the strict uniqueness property. We now use the topology of the MST for P' to connect the corresponding points of P, inducing an MST over P having maximum degree r'. □

Applications of this technique to two and three dimensions are immediate:

Corollary 2.6.2 *Every pointset P in the Manhattan plane has an MST with maximum degree ≤ 4.*

Proof: Modify the diagonal partition into a *strict* diagonal partition having a total of eight regions incident to each point of P: four two-dimensional open wedges (i.e., not containing any of their own boundary points), and four one-dimensional rays (i.e., the boundaries between the wedges). By arguments similar to those of Theorem 2.5.1, each of the open wedges possesses the strict uniqueness property, and thus by Theorem 2.6.1 points lying on the boundaries between wedges can be perturbed into the interiors of the wedges themselves, leaving the one-dimensional regions empty of points. The maximum MST degree given such a partitioning scheme is ≤ 4. □

This bound is tight, e.g., for the center and vertices of a diamond. The best previous bound was that the maximum MST degree in the Manhattan plane is ≤ 6 [124].

Corollary 2.6.3 *Every pointset in three-dimensional Manhattan space has an MST with maximum degree ≤ 14.*

Proof: Modify the cuboctahedral partition into a *strict* cuboctahedral partition having a total of 38 regions incident to each point of P: 14 three-dimensional open pyramids (i.e., eight triangular pyramids and six square pyramids, each not containing any of their own boundary points), and 24 two-dimensional regions (i.e., corresponding to all the boundaries between the pyramids). By arguments identical to those of Theorem 2.5.2, each of the open pyramids possesses the strict uniqueness property, and thus by Theorem 2.6.1, points lying on the boundaries between the 14 pyramids can be perturbed into the interiors of these pyramids. The maximum MST degree given such a partitioning scheme is ≤ 14. □

The best previous bound for the maximum MST degree in three-dimensional Manhattan space was 26 [69, 238]. it is still open whether there exists an example which forces a node in the MST to have degree 14. Interestingly, Corollaries 2.6.2 and 2.6.3 also settle some open questions in complexity theory. It is known that the problem of finding a degree-bounded MST is NP-complete, even when the degree bound is fixed at two (yielding the Traveling Salesman Problem), or at three [190]. Corollary 2.6.2 implies that the degree-bounded MST problem in the Manhattan plane is solvable in polynomial time when the degree bound is fixed at five or at four, since we have shown how to find an MST that meets these maximum degree constraints. Similarly, Corollary 2.6.3 implies that the degree-bounded MST problem in three-dimensional Manhattan space is solvable in polynomial time when the degree bound is ≥ 14. The work of Robins and Salowe [204] investigates the maximum MST degree for higher dimensions and other L_p norms, and relates the maximum MST degree to the so-called "Hadwiger" numbers.

2.7 STEINER TREES IN GRAPHS

Given a weighted graph $G = (V, E)$, $E \subseteq V \times V$, and $N \subseteq V$, the graph version of the SMT problem seeks a minimum-cost tree in G that spans N [48, 91, 166]. Any node in $V - N$ is a potential Steiner point. Each graph edge e_{ij} has a weight w_{ij}, and the *cost* of a tree (or any subgraph) is the sum of the weights of its edges. The graph Steiner problem arises when we wish to route a signal net in the presence of obstacles and congestion [104], or in the presence of variable-cost routing resources, as are present in field-programmable gate arrays [5, 7, 8].

The Graph Steiner Minimal Tree (GSMT) problem: Given a weighted graph $G = (V, E)$, and a set of nodes $N \subseteq V$, find a minimum-cost tree $T = (V', E')$ with $N \subseteq V' \subseteq V$ and $E' \subseteq E$.

The GSMT problem is NP-complete, since the geometric SMT problem is a special case. The heuristic of Kou, Markowsky and Berman (KMB) [159] solves the GSMT problem in polynomial time with performance ratio ≤ 2, using the following three basic steps (see Figure 2.27).

- Construct the complete graph G' over N with the weight of each edge e_{ij} equal to the cost of the corresponding shortest path in G between n_i and n_j. We call this shortest path $path(n_i, n_j)$, and its cost is denoted $dist_G(n_i, n_j)$.
- Compute MST(G'), the minimum spanning tree of G', and expand each edge e_{ij} of MST(G') into the corresponding $path(n_i, n_j)$ to yield subgraph G'' that spans N.
- Finally, compute the minimum spanning tree MST(G''), and delete pendant edges from MST(G'') until all leaves are members of N.

The resulting tree is an approximation to the GSMT that has cost no more than $2 \cdot (1 - \frac{1}{L})$ times optimal, where L is the minimum number of leaves in any optimal Steiner tree solution [159].

The Kou, Markowsky and Berman (KMB) Algorithm
Input: A graph $G = (V, E)$ with edge weights w_{ij} and a set $N \subseteq V$ **Output:** A low-cost tree $T' = (V', E')$ spanning N (i.e. $N \subseteq V'$ and $E' \subseteq E$)
$G' = (N, N \times N)$, with edge weights $w'_{ij} = dist_G(n_i, n_j)$ **Compute** $T = (N, E'') = \text{MST}(G')$ $G'' = \cup_{e_{ij} \in E''} path_G(n_i, n_j)$ **Compute** $T' = \text{MST}(G'')$ **Delete** pendant edges from T' until all leaf nodes are in N **Output** T'

Figure 2.27 The KMB heuristic for the GSMT problem.

The Iterated 1-Steiner approach can be generalized to solve the Steiner problem in arbitrary weighted graphs, by combining the geometric I1S heuristic with the KMB graph algorithm. The resulting hybrid method inherits the good

average performance of the Iterated 1-Steiner method, while also enjoying the error-bounded performance of the KMB algorithm. We refer to this hybrid method as the *Graph Iterated* 1-*Steiner* (GI1S) algorithm. The GI1S method is essentially an adaptation of I1S to graphs, where the "MST" in the inner loop is replaced with the KMB construction. That is, instead of using an "MST" subroutine to determine the "savings" of a candidate Steiner point/node, we use the KMB algorithm for this purpose. Thus, given a graph $G = (V, E)$, a set $N \subseteq V$, and a set S of potential Steiner points, we define the following:

$$\Delta\text{KMB}(N, S) = cost(\text{KMB}(N)) - cost(\text{KMB}(N \cup S))$$

The GI1S algorithm (Figure 2.28) repeatedly finds Steiner node candidates that reduce the overall KMB cost and includes them into the growing set of Steiner nodes S. The cost of the KMB tree over $N \cup S$ will decrease with each added node, and the construction terminates when there is no $x \in V$ with $\Delta\text{KMB}(N \cup S, \{x\}) > 0$.

Graph Iterated 1-Steiner (GI1S) Algorithm
Input: A weighted graph $G = (V, E)$ and a set $N \subseteq V$ **Output:** A low-cost tree $T' = (V', E')$ spanning N (i.e. $N \subseteq V' \subseteq V$ and $E' \subseteq E$)

$S = \emptyset$
While $T = \{x \in V - N \mid \Delta\text{KMB}(N \cup S, \{x\}) > 0\} \neq \emptyset$ **Do**
  Find $x \in T$ with maximum $\Delta\text{KMB}(N \cup S, \{x\})$
  $S = S \cup \{x\}$
Return $\text{KMB}(N \cup S)$

Figure 2.28 The Graph Iterated 1-Steiner algorithm (GI1S).

Given a weighted graph and an arbitrary set of nodes N, a performance ratio for GI1S of $2 \cdot (1 - \frac{1}{L})$ follows from the KMB bound and the fact that the cost of the GI1S construction cannot exceed that of the KMB construction. If $|N| \leq 3$ (e.g., a VLSI signal net with three or fewer terminals), GI1S is guaranteed to find an optimal solution. Although the worst-case performance ratio of GI1S is the same as that of KMB, in practice GI1S significantly outperforms KMB [7, 8]. Given a faster implementation of the KMB method [249], the GI1S algorithm can be implemented within time $O(|N| \cdot |G| + |N|^4 \log |N|)$, where $|N| \leq |V|$ is the number of nodes to be spanned and $|G| = |V| + |E|$ is the size of the graph. Other works that address Steiner routing in a graph include [48, 104, 166].

Note that the GI1S template above can be viewed as an *Iterated KMB* (IKMB) construction, and that KMB inside the inner loop may be replaced with any other graph Steiner heuristic, such as that of Zelikovsky (ZEL) [254], yielding an *Iterated Zelikovsky* (IZEL) heuristic. IZEL enjoys the same theoretical performance bound as ZEL, namely $\frac{11}{6}$. Experiments have shown that these heuristics in order of increasing empirical average performance are: KMB, ZEL, IKMB, and IZEL [9]. Thus, iterating a given Steiner heuristic is an effective general mechanism to improve empirical performance without sacrificing theoretical performance bounds.

2.8 EXPERIMENTAL RESULTS

We have implemented both serial and parallel versions of the I1S, B1S, and PI2S algorithms, using C in the Sun workstation environment. We compared these with the fastest known optimal Steiner tree algorithm (OPT) of Salowe and Warme [208] on up to 10000 pointsets of various cardinalities. Random instances were generated by choosing the coordinates of each point independently from a uniform distribution in a 10000×10000 grid; such instances are statistically similar to the terminal locations of actual VLSI nets and are a standard testbed for Steiner tree heuristics [138]. Both I1S and B1S have very similar average performance, approaching 11% improvement over MST cost (Figure 2.30(a)).[10] The average number of rounds for B1S is 2.5 for sets of 300 points, and was never observed to be more than 5 on any instance (Figure 2.30(b)); over 95% of the total improvement occurs in the first round, and over 99% of the improvement occurs in the first two rounds. The average number of Steiner points generated by B1S grows linearly with the number of points (Figure 2.30(c)). An example of the output of B1S on a random 300-point set is shown in Figure 2.29.

Figure 2.31(a) shows the performance comparison of B1S, PI2S, and OPT on small pointsets. We observe that the average performance of PI2S is nearly optimal: for $n = 8$, PI2S is on average only about 0.11% away from optimal, and solutions are optimal in about 90% of the cases (Figure 2.31(b)). Even for $n = 30$, B1S is only about 0.30% away from optimal, and yields optimal solutions in about one quarter of the cases.

[10] Recently, other Steiner heuristics with performance approaching that of I1S were proposed by Borah et al. [36], Chao and Hsu [43], and Lewis et al. [171].

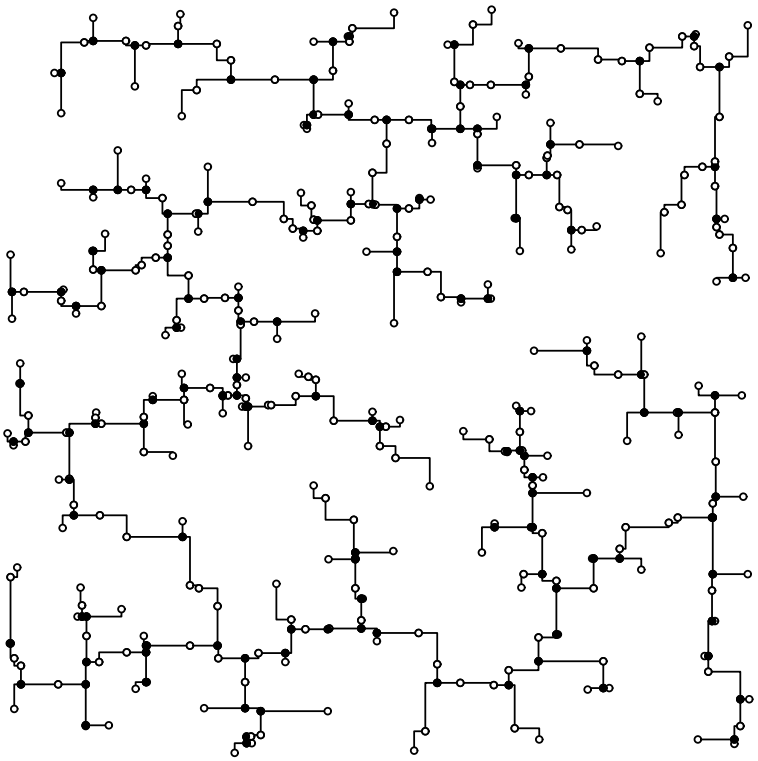


Figure 2.29 An example of the output of B1S on a random set of 300 points (hollow dots). The Steiner points produced by B1S are denoted by solid dots.

We timed the execution of the serial and parallel versions of B1S, using both a naive $O(n^4 \log n)$ implementation and the $O(n^3)$ implementation which incorporates the efficient, dynamic MST maintenance as described in Section 2.5. The parallel implementation (see 2.4.3used nine Sun 4/40 SPARC1 workstations, with a Sun 4/75 SPARC2 as the master processor. For $n = 100$, the fast serial implementation is 247 times faster than the naive implementation, and the parallel implementation running on 10 processors is 1163 times faster (Figure 2.30(d)). Even for small pointsets, the enhanced implementation is considerably faster than the naive one: for $n = 5$, the serial B1S is on average more than twice as fast as the naive implementation, while for $n = 10$ the serial speedup factor approaches 7. Notice that the serial speedup increases with

problem size; the parallel speedup (defined as the parallel time divided by the serial time) also increases with problem size, reaching about 7.2 for $n = 250$ using 10 processors.

The average running times of the algorithms for various pointset cardinalities are compared in Figure 2.30(d). The most time-efficient of the heuristics is B1S, requiring an average of 0.009 CPU seconds for $n = 8$, and an average of 0.375 seconds for $n = 30$. Using PIkS (or PBkS) with $k > 2$ improves the performance, but slows down the algorithm. Recall that for arbitrary k, PIkS (PBkS) always yields optimal solutions for $\leq k + 2$ points, but has time complexity greater by a factor of $n^{2(k-1)}$ than PI1S (PB1S). While this enables a smooth tradeoff between performance and efficiency, the performance of the PBkS algorithm with $k = 2$ is already so close to optimal that use of $k > 2$ is not likely to justify the resulting time penalty in most applications.

In three dimensions, we observed that the limit when the number of planes L approaches ∞, the average performance of PB1S approaches 15% improvement over MST cost, and the performance increases with L (Figure 2.31(c)). Recall that the average savings over MST cost in three dimensions is expected to be higher than in two dimensions, since the worst-case performance ratio is higher also (i.e., $\frac{5}{3}$ for three dimensions vs. $\frac{3}{2}$ for two dimensions). The number of added Steiner points in three dimensions grows linearly with the pointset cardinality (Figure 2.31(d)). In all cases, the L parallel planes were uniformly spaced in the unit cube (i.e., they were separated by $\frac{G}{L}$ units apart, where $G = 10000$ is the gridsize). The OPT algorithm of Salowe and Warme [208] does not generalize to higher dimensions, and thus we were not able to compare the three-dimensional version of PB1S against optimal solutions. As in two dimensions, the average number of rounds for B1S is very small.

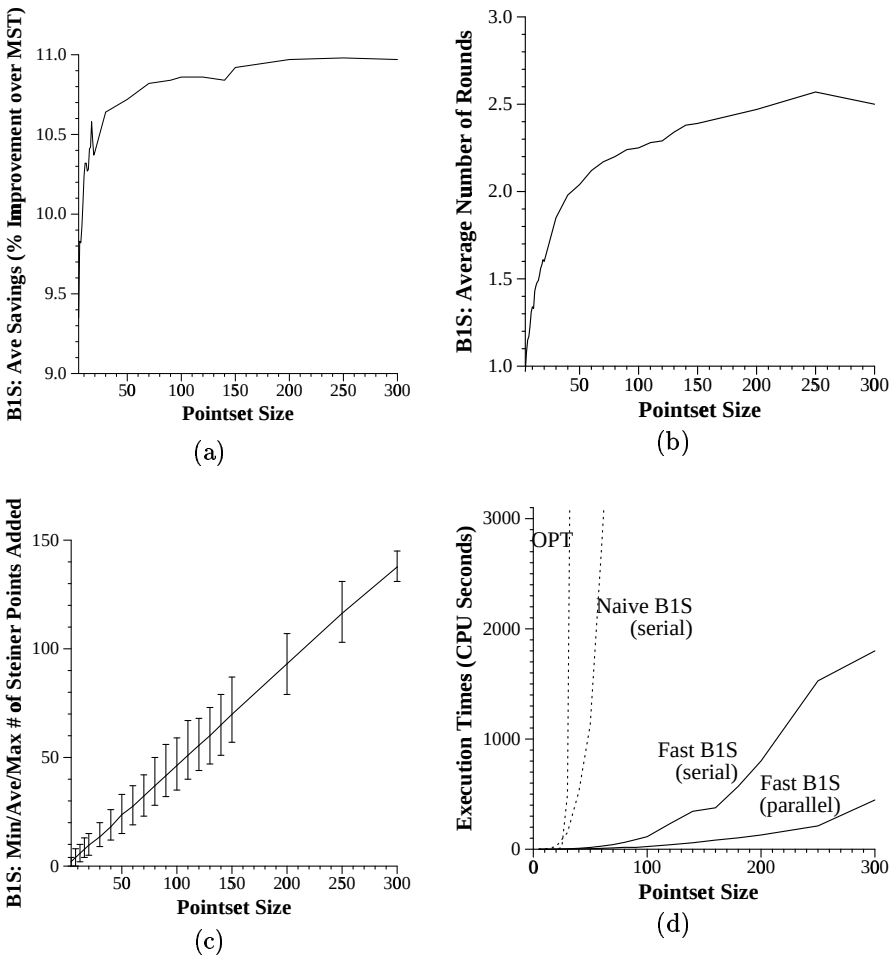


Figure 2.30 (a) Average performance of B1S, shown as percent improvement over MST cost. (b) Average number of rounds for B1S. (c) Average number of Steiner points induced by B1S (vertical bars indicate the range of the minimum and maximum number of Steiner points added) (d) Average execution times (in CPU seconds) for B1S, for both the naive implementation, as well as the "fast" B1S which uses the incremental MST maintenance scheme (also shown are OPT and the parallel version of B1S).

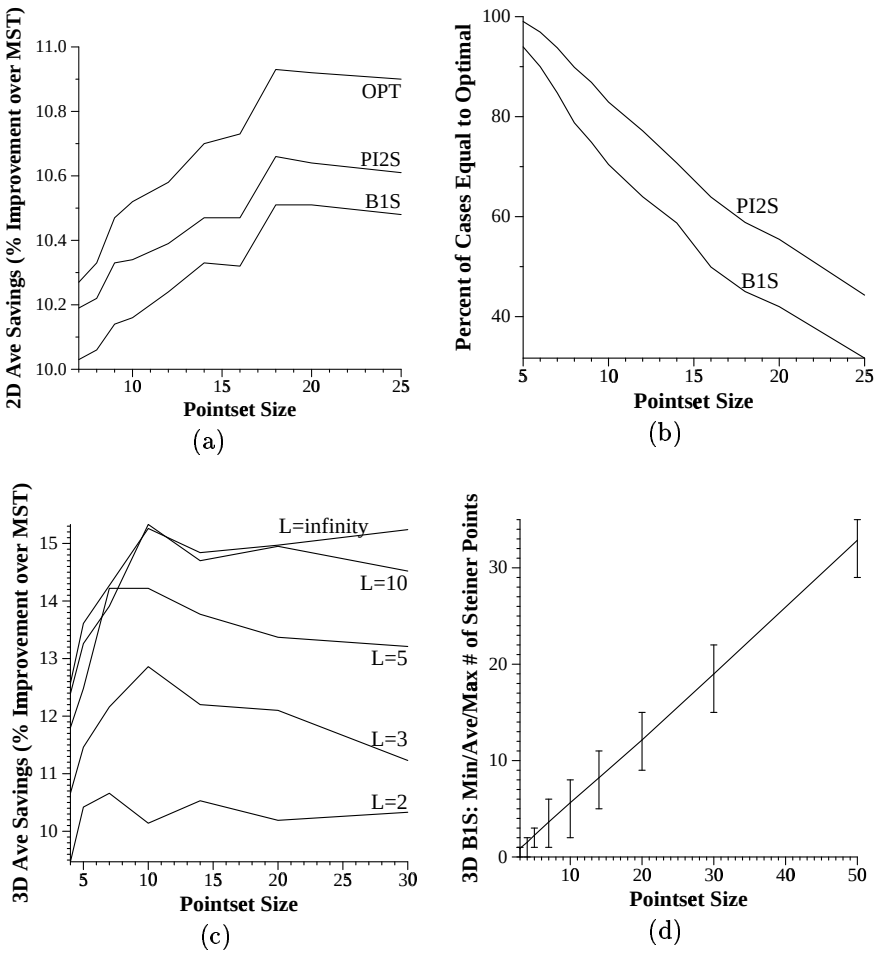


Figure 2.31 (a) Average performance in two dimensions of PI2S, B1S, and OPT; note that PI2S is only 0.25% (or less) away from optimal. (b) Percentage of all cases when the heuristics find the optimal solution (note that PI2S yields optimal solutions a large percentage of the time). (c) Average performance of PB1S in three dimensions for various values of L = number of parallel planes. (d) Average number of Steiner points added by B1S in three dimensions for $L = \infty$.

3

DELAY

Overview of the Chapter

This chapter considers the problem of minimizing *signal delay* for performance-driven system design. The signal delay objective moves us from the *unoriented* pointset P of the Steiner problem to an *oriented* signal net S which has an identified source. Optimal-delay wiring geometries can differ substantially from those of optimal-area (Steiner minimal tree) solutions, particularly as technology moves into submicron regimes and layout dimensions continue to increase. Our discussion reflects the history of our recent research, which has addressed four major issues.

First, there is the issue of *technology-dependent* methodologies versus *technology-independent* methodologies. Analysis of the Elmore delay formula for distributed RC trees motivates a *cost-radius tradeoff* that is clearly dependent on technology, as has been discussed in [13, 16, 17, 62, 63, 156]. Thus, routing tree constructions that are based on aspects of technology, net criticality, or other factors can potentially improve over static, "oblivious" methods.

Second, there is the issue of *"actual delay"* versus *geometric* objectives. Many early works used geometric objectives, e.g., tree cost or tree radius, essentially for algorithmic convenience and tractability of analysis. By contrast, the class of objectives proposed by Boese et al. [32, 34] leads to improved performance by optimizing Elmore delay directly. A review of the various delay estimates, along with data establishing their respective *fidelities* to SPICE-computed delay, is given in the Appendix.

Third, there is the issue of minimizing *net-dependent* delays versus *sink-dependent* delays. Because timing-driven placement and routing will usually be iterated with static timing estimation, critical-path information is often available during the routing tree construction. Thus, a formulation which optimizes delay to a set of critical sinks, as in the work of Boese, Kahng and Robins [34], is of interest.

Finally, the fourth issue involves *quantifying* the the near-optimality of minimum-delay routing heuristics. Just as empirical studies showed that I1S averages within a fraction of one percent from optimal for the rectilinear SMT problem, the "SERT-C" heuristic proposed in [34] is actually very close to optimal in terms of weighted critical sink delays. Boese and coauthors [32] established a basis for this assessment by showing how to construct Steiner trees with *optimal* Elmore delay. Their proof of correctness uses (i) a generalization of Hanan's theorem to Elmore delay-optimal Steiner trees, and (ii) a "peeling" decomposition for optimal Steiner trees.

In addition to these issues and their solutions, we will describe a number of confirming experimental results. The chapter concludes with a survey of other recent advances in performance-driven interconnect design, notably the application of non-tree topologies and wiresizing techniques to improve circuit performance.

3.1 PRELIMINARIES

With the scaling of device technology and die size, interconnection delay now contributes up to 70% of the clock cycle in dense, high performance circuits [18, 77, 234]. As a result, performance-driven layout design has been studied actively since the late 1980's. Because module placement has a significant effect on the space of achievable signal delays, initial research centered on timing-driven placement, in which the objective is for adjacent modules on critical paths to be placed close together. Examples of timing-driven placement algorithms include a "zero-slack" algorithm proposed by Hauge et al. [119]; the fictitious-facilities and floating-anchors methods of Marek-Sadowska and Lin [178]; and a linear programming approach by Jackson et al. [140]. Several other timing-driven placement approaches, including methods based on simulated annealing, have been proposed in [77, 173, 234]. Since in general no global routing solution is available at the placement step, each of these methods uses a simple estimate of interconnection delay, such as those discussed below.

Given a timing-driven module placement, the corresponding timing-driven routing algorithm minimizes average or maximum signal delay from the source terminal to the sink terminals of a signal net. An example method is that Dunlop et al. [79], which determines net priorities based on static timing analysis; nets with high priorities are processed earlier using fewer feedthroughs. Jackson et al. [142] outlined a hierarchical approach, and Prasitjutrakul and Kubitz [192] proposed a router for building-block design based on the A* search algorithm. These results have had great influence on succeeding works, but fall short of providing general, well-motivated solutions to the problem of optimal-delay routing. In what follows, all of our methods will be motivated by a simple, recurring question: what is the *proper objective* for optimal-delay routing tree construction?

3.1.1 Definitions

We define a *signal net* $S = \{s_0, s_1, \ldots, s_n\}$ to be a set of $n+1$ terminals, with s_0 the *source* and the remaining terminals *sinks*. Performance-driven interconnection problems have two basic flavors: geometric instances arising in cell-based design, and weighted graph instances arising in building-block design. In cell-based design, routing cost is closely approximated by Manhattan distance, while in building-block design, routing cost is typically given by the cost of a shortest path in the channel intersection graph of the layout (see Section 1.2). Thus, the signal net S is more generally viewed as being embedded in an underlying routing graph $G = (V, E)$ with $S \subseteq V$. The graph G is connected and has variable edge weights (costs): each edge $e_{ij} \in E$ has a cost $d(v_i, v_j)$ equal to the routing cost between v_i and v_j. We seek a routing tree T in G that spans S. The cost of T is defined to be $cost(T) = \sum_{e_{ij} \in T} d(v_i, v_j)$. When $V = S$, the spanning tree with minimum cost is the MST or T_M.

The cost of a path in G is defined as the sum of its edge costs. The minimum-cost path in G between two vertices v_i and v_j is denoted $minpath_G(v_i, v_j)$, and we use $dist_G(v_i, v_j)$ to denote its cost. In a routing tree, $minpath_T(v_i, v_j)$ is simply the unique v_i-v_j path. The distance in a tree from the source to a given sink s_i is specially denoted as $l_i = dist_T(s_0, s_i)$.

In much of the following discussion, the *radius* of either a signal net or a routing tree will hold special interest. The radius of a routing tree T is $r(T) = \max_{1 \le i \le n} l_i$. Given signal net S and an underlying routing graph G, we use R_i to denote the cost of the shortest s_0-s_i path in G, i.e., $R_i = dist_G(s_0, s_i)$. A shortest paths tree, denoted as an SPT or T_S, has $l_i = R_i$ for all sinks s_i. At times,

we use R to denote the maximum R_i value over all sinks s_i, and we say that R is the *radius* of the signal net. Much of our discussion will concern the case of S being embedded in geometry, so that G is a complete graph with each e_{ij} having cost equal to the Manhattan distance, d_{ij}, between v_i and v_j. In this case, $R_i = d_{0i}$ for all sinks s_i. Finally, a vertex v that is embedded in the plane has x- and y-coordinates v_x and v_y, respectively.

3.1.2 The Linear and Elmore Delay Approximations

The proper objective to use in *efficiently* constructing a "high-performance routing tree" over a given signal net is not yet established. Many works rely on the linear delay model, where the signal delay from s_i to s_j is proportional to the length of the s_i-s_j path in the routing tree. The linear model can be used to motivate essentially geometric routing constructions (e.g., a shortest paths tree has optimum delay at every sink according to the linear model). However, valuable insights are also obtained by considering the Elmore delay model, i.e., the first moment of the impulse response for a distributed RC representation of the routing tree [87].

Elmore delay in an RC tree is defined as follows [205, 240]. Given routing tree T rooted at the source s_0, let e_v denote the edge from node v to its parent in T. The resistance and capacitance of edge e_v are denoted by r_{e_v} and c_{e_v}, respectively. Let T_v denote the subtree of T rooted at v, and let c_v denote the sink capacitance of v (we assume that $c_v = 0$ if v is a Steiner node). We use C_v to denote the *tree capacitance* of T_v, defined to be the sum of sink and edge capacitances in T_v (note that when T_v is a single (leaf) node, C_v is equal to the corresponding sink capacitance c_v). Using this notation, the Elmore delay along edge e_v equals $r_{e_v}(\frac{c_{e_v}}{2} + C_v)$. Let r_d denote the on-resistance of the output driver at the source. Then the Elmore delay $t_{ED}(s_i)$ at sink s_i is

$$t_{ED}(s_i) = r_d C_{s_0} \quad + \sum_{e_v \in path(s_0, s_i)} r_{e_v}(\frac{c_{e_v}}{2} + C_v) \tag{3.1}$$

A fundamental property of this expression, which has been noted by Lin and Mead [176], Rubinstein et al. [205], Tsay [240] and others, is that $t_{ED}(s_i)$ can be evaluated for all $i = 1, \ldots, n$ in $O(n)$ time. Two depth-first traversals of the tree are sufficient: the first traversal calculates all C_v values and the delays on each edge, while the second adds up the delays on each source-sink path. This

calculation is enabling to the efficient methodologies described in Section 3.3 below.

In the Appendix, we review the underlying theory behind several efficient delay approximations, including Elmore's approximation and the class of two-pole (moment-matching) methods. The Appendix also reviews recent work of Boese et al. [30, 31, 32, 33, 34] which shows that Elmore delay has high *fidelity* with respect to SPICE-computed delay over a wide range of technologies. It turns out that although Elmore's formula can yield inaccurate delay estimates, the ranking of alternate routing tree solutions by Elmore delay closely reflects the ranking obtained using SPICE3e2. Similar results have been obtained by Kim et al. [157], who simulated critical-path delays over a suite of 209 ripple-carry adder implementations and found near-perfect correlation between SPICE-computed and Elmore delays. A theoretical motivation for this correspondence, based on group delay, was given in [245]. These results form the basis of our focus on Elmore delay at the end of this chapter.

If r_{e_v} and c_{e_v} are proportional to the length of e_v (with unit resistance and unit capacitance given by $\bar{r}$ and $\bar{c}$), then the $r_d \cdot C_{s_0}$ term in Equation 3.1 has linear dependence on $cost(T)$, while the summation term has quadratic dependence on l_i. As a result, we can distill an essential "cost-radius conflict" inherent in routing tree design: (i) when r_d is relatively large, the $r_d \cdot C_{s_0}$ term dominates the summation and suggests a minimum-cost routing solution, but (ii) when r_d is relatively small, the quadratic dependence on source-sink pathlength dominates, and suggests a "star-like", shortest paths tree topology.

An essentially similar insight was derived in [28, 65, 257] from the simple upper bound on Elmore delay due to Rubinstein et al. [205]. The Elmore delay upper bound for a tree T is simply the summation, over all nodes in T, of the RC product arising when each node capacitance sees all the resistance between the node and the source.[1] Note that this upper bound applies generically to delay at every sink, unlike the sink-specific expression of Equation 3.1. The upper bound can be re-expressed as the sum of four terms: one term is minimized when T has minimum cost; a second term is minimized when T is a shortest paths tree; a third term is minimized when T is what the authors call a "quadratic minimum

[1] Let s'_j be one of a finite number of points used to represent the tree, and let $\bar{c'}_j$ denote the total capacitance at s'_j (when the tree is modeled as composed of a finite number of segments, $\bar{c'}_j$ indicates the sum of the internal capacitance (e.g., if s'_j is a sink) and the wire capacitance between s'_j and the nearest point on the s'_j-s_0 path). If $\bar{R}_j$ denotes the total resistance on the s'_j-s_0 path, then the upper bound on any sink delay in T is $t_{ED} \leq \sum_j \bar{R}_j \bar{c'}_j$, where the summation is taken over all points in the tree, not just the sinks s_i.

Steiner tree"; and the fourth term is a constant. As in the analysis of Equation 3.1, it is the relative size of r_d which indicates the dominant term in the delay expression. The size of r_d relative to the unit resistance $\bar{r}$ is a "resistance ratio" [28, 65, 256] that captures the technology vis-a-vis routing tree design. Values of $\frac{r_d}{\bar{r}}$ have typically decreased in current submicron CMOS and MCM substrate technologies (see Table 3.1), suggesting that the traditional minimum-cost objective is becoming less germane to performance-driven routing.

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

Table 3.1 Interconnect parameters for three CMOS IC technologies and an MCM technology. Parasitics for the IC1 and IC2 technologies are provided by the MOSIS project at the USC Information Sciences Institute; IC3 parasitics are courtesy of the Microelectronics Center of North Carolina; MCM interconnect parasitics are courtesy of Professor Wayne W.-M. Dai of UC Santa Cruz and correspond to data provided by AT&T Microelectronics Division. Unit inductance for the MCM interconnect is $380 fH/\mu m$, and is assumed negligible for IC interconnect. The r_d values are scaled driver resistances. Sink loading capacitances (c_i) are derived for minimum-size transistors.

3.2 GEOMETRIC APPROACHES TO DELAY MINIMIZATION

The above analysis of Elmore delay provides a retrospective validation of several minimum-delay routing tree heuristics which trade off between tree cost and tree radius. In this section, we first survey two early works that adopt such *geometric* "cost-radius" intuitions, and then present three effective classes of heuristics that are also based on purely geometric objectives.

3.2.1 Early Cost-Radius Tradeoffs

An early work of Cohoon and Randall [57] is notable for its prescient insights. For any given signal net, [57] proposed the construction of a "maximum performance tree" corresponding to "a shortest path tree ... with minimum total length", and noted that such a tree seems difficult to construct. While the minimum-cost shortest paths (spanning) tree is easily computed[2], the Steiner version of Cohoon and Randall's question is precisely the rectilinear Steiner arborescence problem discussed in Section 3.2.4 below. A heuristic was given which determines a central trunk for the Steiner topology, then "attempt[s] to combine the best features of an RMST and an RSPT", i.e., a rectilinear minimum spanning tree and a rectilinear shortest paths tree. This idea, too, is interesting in light of the various cost-radius tradeoffs that are discussed below. The heuristic in [57] connects the most distant sink directly to the source with a wire of length R, then proceeds with an MST-like construction; if a terminal s_i is about to be added into the tree with $l_i > R$, then the method reverts back to an SPT-like construction. In this way, all source-sink paths are guaranteed to be of length $\leq R$. A final phase of the heuristic performs edge-overlapping to further reduce the Steiner tree cost.

The work of Cong et al. [61], which was contemporaneous with [57], also observed the existence of conflicting min-cost and min-radius objectives in performance-driven routing. While the shortest paths tree (SPT, or T_S) has the smallest possible radius of any routing tree, its cost might be $\Omega(|S|)$ times greater than the cost of the minimum spanning tree (MST, or T_M); see Figure 3.1. On the other hand, the radius $r(T_M)$ can be much larger than $r(T_S)$.

To address both tree radius and tree cost in the routing construction, [61] proposed the following:

The Bounded-Radius Minimum Routing Tree (BRMRT) Problem: Given a parameter $\epsilon \geq 0$ and a signal net with radius R, find a minimum-cost routing tree T with radius $r(T) \leq (1 + \epsilon) \cdot R$.

The parameter ϵ specifies a tradeoff between the minimum-radius and minimum-cost objectives. When $\epsilon = 0$, a minimum-radius spanning tree is obtained, and as ϵ increases, the weaker radius restriction allows further reduction of tree cost. When $\epsilon = \infty$, a minimum-cost spanning tree is obtained. Figure 3.2 gives

[2]We call such a tree a minimum-cost *spanning arborescence*. It may be computed by executing Dijkstra's single-source shortest paths algorithm, and breaking ties in each pass so that the closest possible node is chosen among all possible parents of the new permanent node.

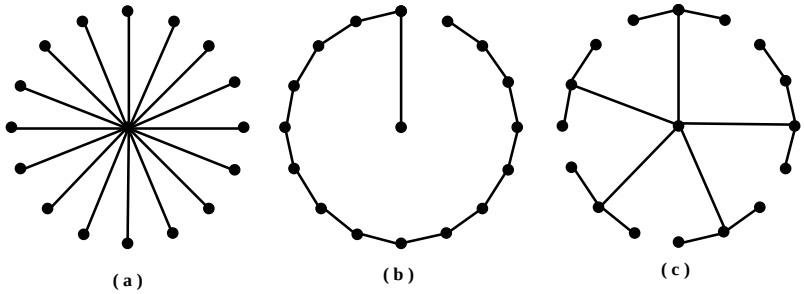
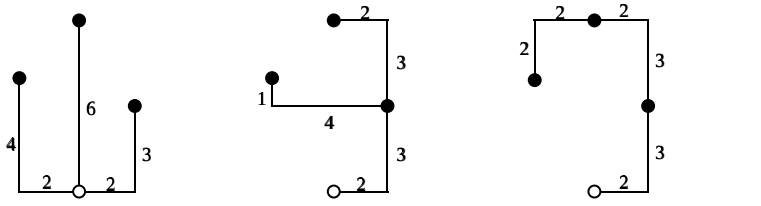


Figure 3.1 Three interconnection trees for the same signal net with s_0 at the center: (a) the shortest paths tree T_S; (b) the minimum spanning tree T_M; and (c) a "tradeoff" between the two constructions.

an example with three distinct spanning trees obtained using different values of ϵ: Figure 3.2(a) shows a minimum-radius spanning tree corresponding to the case $\epsilon = 0$, with $r(T) = 6$; Figure 3.2(b) shows a solution with $\epsilon = 1$ and $r(T) = 10$; and Figure 3.2(c) shows the minimum spanning tree corresponding to the case $\epsilon = \infty$, with $r(T) = 14$. The complexity of the BRMRT formulation for spanning trees is still open; when Steiner points are allowed in the routing tree, choosing $\epsilon = \infty$ yields the Steiner minimal tree formulation.



(a) ε = 0, cost(**T**) = 17, r(T)=6 (b) ε = 1, cost(**T**) = 15, r(T)=10 (c) ε =∞, cost(**T**) = 14, r(T)=14

Figure 3.2 Increasing ϵ may result in decreased tree cost, but increased tree radius.

The Bounded-Prim (BPRIM) Algorithm

Recall that in cell-based design methodologies, routing costs are closely approximated by geometric distance, and the underlying routing graph is essentially the complete graph $G = (V, E)$ with $V = S$. In this regime, spanning tree solutions will be of interest, even for performance-driven routing formulations: (i) spanning trees are often easier to compute than Steiner trees, and (ii) a spanning solution can be easily converted into a corresponding Steiner solution by edge-overlapping, while retaining essentially identical radius parameters. For the BRMRT variant which seeks a bounded-radius *spanning* tree, an effective heuristic follows the general scheme of Prim's minimum spanning tree construction [196]. This "Bounded-Prim" (BPRIM) algorithm (Figure 3.3) grows a tree $T = (V', E')$ which initially contains only the source s_0. At each step, terminals $s_i \in V'$ and $s_j \in S - V'$ are determined such that $d(s_i, s_j)$ is minimum. If adding the edge (s_i, s_j) to T does not violate the radius constraint, i.e., $l_i + d(s_i, s_j) \leq (1 + \epsilon) \cdot R$, the edge (s_i, s_j) is added to T. Otherwise, the algorithm "backtraces" along the path from s_i to s_0 in T, and finds the first terminal $s_{i'}$ such that the edge $(s_{i'}, s_j)$ is *appropriate*, i.e., $l_{i'} + d(s_{i'}, s_j) \leq R$. The edge $(s_{i'}, s_j)$ is then added to the tree. In the worst case, the backtracing will terminate with $s_{i'} = s_0$, since edge (s_0, s_j) is certain to be appropriate.

Note that the backtracing chooses $s_{i'}$ so that $l_{i'} + d(s_{i'}, s_j) \leq R$, instead of the more obvious condition $\leq (1 + \epsilon) \cdot R$. This introduces some "slack" at s_j, so that terminals added later within an $\epsilon \cdot R$ neighborhood of s_j will not cause additional backtracing. Limiting the amount of backtracing in this way keeps the cost of the resulting tree closer to that of the minimum spanning tree, while still guaranteeing that backtracing is always possible. By contrast, the method of Cohoon and Randall always enforces $\epsilon = 0$ in its construction. The most direct implementation of BPRIM requires $\Theta(n^2)$ time since each new terminal can force examination of most of the previously added terminals.

It is easy to see that $r(T_{BPRIM})$ is never greater than $r(T_M)$ if the MST T_M is unique.

Lemma 3.2.1 *If the MST T_M is unique, then $r(T_{BPRIM}) \leq r(T_M)$.*

Proof: If $r(T_M) \leq (1 + \epsilon) \cdot R$, then $r(T_{BPRIM}) = r(T_M)$ since T_{BPRIM} and T_M will each be uniquely constructed, and will be identical to each other. Otherwise, $r(T_{BPRIM}) \leq (1 + \epsilon) \cdot R < r(T_M)$ by construction. □

Algorithm BPRIM: Computing a bounded-radius spanning tree
Input: Net S with radius R, source s_0; parameter $\epsilon \geq 0$
Output: Spanning tree T_{BPRIM} with $r(T_{BPRIM}) \leq (1+\epsilon) \cdot R$
$T = (V', E') = (\{s_0\}, \emptyset)$
While $\|V'\| < \|S\|$
Select $s_i \in V'$ and $s_j \in S - V'$ minimizing $dist(s_i, s_j)$
If $l_i + dist(s_i, s_j) \leq (1+\epsilon) \cdot R$ **Then** $s_{i'} = s_i$
Else find the first terminal $s_{i'}$ along the path in T from s_i to s_0 such that $l_{i'} + dist(s_{i'}, s_j) \leq R$
$V' = V' \cup \{s_j\}$
$E' = E' \cup \{(s_{i'}, s_j)\}$
Output $T_{BPRIM} = T$

Figure 3.3 Algorithm BPRIM: computing a bounded-radius spanning tree T_{BPRIM} for a given signal net S, with source $s_0 \in S$ and radius R, using parameter $\epsilon \geq 0$.

When T_M is not unique, the radii of different minimum spanning trees can vary by an unbounded amount, and $r(T_{BPRIM})$ may be greater than $r(T_M)$. Thus, Lemma 3.2.1 will not hold for all choices of T_M. In the example of Figure 3.4, a Prim-like minimum spanning tree algorithm may choose a connection to y_1 instead of x_1, or y_2 instead of x_2, etc., such that $r(T_{BPRIM}) \gg r(T_M)$ even though the two trees have identical cost. Of course, $r(T_{BPRIM})$ cannot exceed the maximum possible $r(T_M)$. Choosing a minimum-radius T_M when the MST is not unique has unknown complexity.

In general, the worst-case cost performance ratio between $cost(T_{BPRIM})$ and the cost of the optimal bounded-radius minimum spanning tree will depend on the ϵ and $|S|$. Experimental results [63] show that $\frac{cost(T_{BPRIM})}{cost(T_M)}$, which is clearly an upper bound on the cost performance ratio, is in practice bounded by a small constant even when $|S|$ is large. However, the cost performance ratio is not bounded by a constant for any value of ϵ.

Theorem 3.2.2 *For any value of ϵ, the ratio of $cost(T_{BPRIM})$ to the cost of the optimal bounded-radius minimum spanning tree can be arbitrarily large.*

Proof: The construction of Figure 3.5 shows that BPRIM will have unbounded cost performance ratio. The optimal solution is shown on the left with all

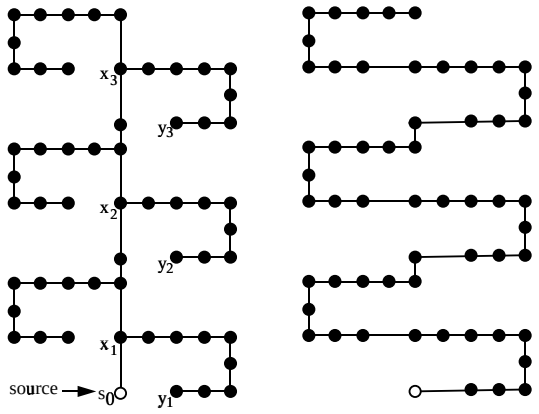


Figure 3.4 A construction for which the radius of an MST (right) is arbitrarily larger than that of a minimum-radius MST (left).

source-leaf pathlengths equal to R. Terminal y is situated so that the pathlength from the source to any leaf via y is slightly greater than $(1+\epsilon)\cdot R$. This will cause the BPRIM construction to backtrace all the way back to the source from every leaf, yielding an unbounded performance ratio. For any value of ϵ, y can be replaced by many closely spaced terminals so that BPRIM creates an appropriately long path between s_0 and x. □

Extensions of BPRIM

The bounded-radius construction can also be applied to minimum spanning tree methods other than Prim's algorithm. A more general algorithm template is given in Figure 3.6.

Many distinct variants are possible, depending on how the pair of terminals s_i and s_j are selected inside the inner loop. The following variants H1, H2 and H3 have improved performance over the original BPRIM algorithm [61, 63]. These three variants afford progressively more freedom in the choice of s_j and its point of connection to the existing tree. Whereas BPRIM connects s_j using the first appropriate edge to $s_{i'}$ along the s_i-s_0 path, H1 picks the minimum-length appropriate edge to any $s_{i'}$ on the path; H2 finds the minimum-length appropriate edge to any $s_{i'} \in V'$; and H3 finds the minimum-length appropriate edge between any $s_{i'} \in V'$ and any $s_j \in S - V'$.

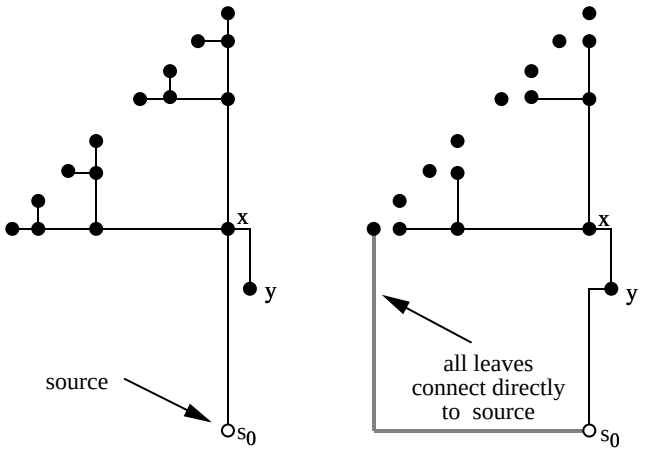


Figure 3.5 The value of $cost(T_{BPRIM})$ is not bounded by any constant factor from optimal for any value of ϵ. The optimal solution is shown on the left, and T_{BPRIM} is shown on the right.

Algorithm Extended-BPRIM: Computing a bounded-radius spanning tree
Input: Net S with radius R, source s_0; parameter $\epsilon \geq 0$ **Output:** Spanning tree T with $r(T) \leq (1+\epsilon) \cdot R$

$T = (V', E') = (\{s_0\}, \emptyset)$
While $\|V'\| < \|S\|$
  Select two terminals $s_i \in V'$ and $s_j \in S - V'$
  with $l_i + d(s_i, s_j) \leq (1+\epsilon) \cdot R$
  $V' = V' \cup \{s_j\}$
  $E' = E' \cup \{(s_i, s_j)\}$
Output T

Figure 3.6 A more general BPRIM template.

- **H1** – Find s_i and s_j as in BPRIM, and select the terminal $s_{i'}$ along the path in T from s_i to s_0 which yields an appropriate edge $(s_{i'}, s_j)$ of minimum length.
- **H2** – Find s_i and s_j as in BPRIM, and select the terminal $s_{i'} \in V'$ which yields a minimum-length appropriate edge $(s_{i'}, s_j)$.

- **H3** – Find a pair of terminals $s_i \in V'$ and $s_j \in S - V'$ that yield a minimum-length appropriate edge (s_i, s_j).

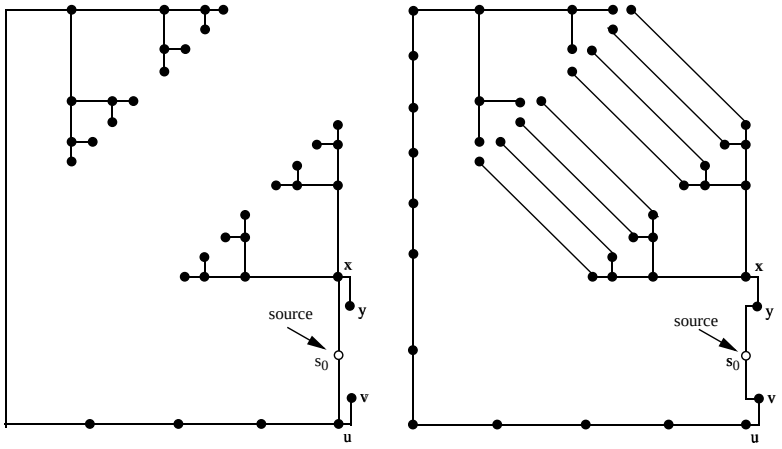


Figure 3.7 Construction showing that the cost performance ratio of both H2 and H3 is not bounded by a constant for any ϵ. The optimal solution is shown on the left; both T_{H2} and T_{H3} will be identical to the tree shown on the right. As with Figure 3.5, the construction can be changed to fit any given value of ϵ by introducing paths of closely spaced points between s_0 and x.

The time complexity of variants H1 and H2 is $O(|S|^2)$, while variant H3 can be implemented to run in time $O(|S|^3)$. Lemma 3.2.1 holds for each of H1, H2, and H3. However, Figure 3.5 shows that variant H1 will also have unbounded cost performance ratio, and the example of Figure 3.7 establishes unbounded performance ratio for variants H2 and H3. Notice that while H1, H2 and H3 appear ordered by increasing power and flexibility, Figure 3.8 shows that BPRIM can outperform these more complicated variants.

3.2.2 Shallow-Light Constructions

In order to bound both the worst-case radius performance and the worst-case cost performance of the routing tree, "shallow-light" tree constructions have

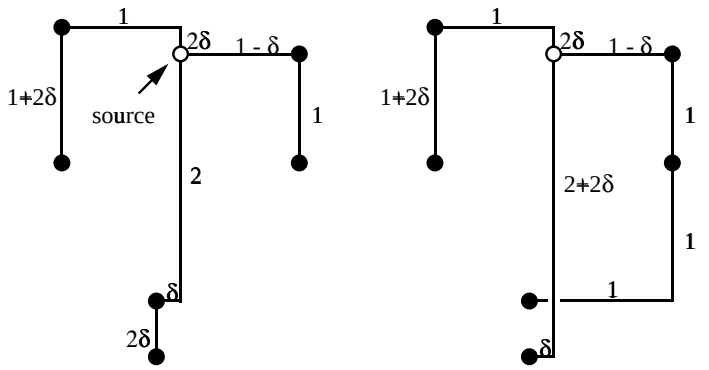


Figure 3.8 Example for which BPRIM (left) outperforms variants H2 and H3 (right); δ is a very small real number and $\epsilon = (2 - 3\delta)/(2 + 3\delta)$.

been proposed which capture properties of both T_M and T_S *simultaneously* to within constant factors of optimal.[3]

Definition: Given a signal net S and parameter $\alpha \geq 1$, a *shallow-light* tree $T = (S, E')$ is a spanning tree over S that satisfies: (i) $l_i \leq \alpha \cdot R_i$, $1 \leq i \leq n$, and (ii) $cost(T) \leq \beta \cdot cost(T_M)$ with the constant β depending only on α. We call such a tree an (α, β)-tree.

Works by three separate groups provide shallow-light constructions [17, 63, 156]. All three groups use the following general technique, pioneered by Awerbuch et al. in [16]:[4]

1. construct T_M;
2. visit the terminals of S in the order of a depth-first traversal of T_M;

[3]The term "shallow-light" seems to have originated in the work of Awerbuch et al. [16], and indicates a tree with bounded radius (i.e., "shallow") and bounded cost or weight (i.e., "light").

[4]This basic technique of Awerbuch et al. [16] can be traced further back to literature in the sparse graph spanner area of computational geometry, e.g., see the work of Levcopoulos and Lingas [170]. Generally speaking, techniques used for sparse graph spanners have strong resonances with VLSI routing objectives (e.g., see [42, 191]). However, a graph spanner has bounded pathlengths between *all pairs* of nodes in a given graph, which is too strong a constraint for our (single-source) routing application.

3. whenever violations of the prescribed radius bound are observed, insert or delete edges as necessary; and

4. return the shortest paths tree (with respect to the single source s_0) over the resulting graph.

Cong et al. [63] proposed the "Bounded Radius, Bounded Cost" (BRBC) algorithm for performance-driven global routing; this algorithm is the focus of the present subsection. The unpublished manuscript of Awerbuch et al. [17] describes an algorithm that is identical to BRBC, and shows that it yields a shallow-light, $(1 + 2\epsilon, 1 + \frac{2}{\epsilon})$-tree for parameter $\epsilon > 0$. Finally, the method of Khuller et al. [156] obtains a $(1 + \epsilon, 1 + \frac{2}{\epsilon})$ shallow-light construction by "relaxing" edges, in contrast to the earlier works of [17, 63] which add complete source-sink shortest paths when violations of the radius bound occur.

In surveying these results, which have occurred in rapid succession over the past several years, several aspects of their precise history should be noted. The seminal work of Awerbuch et al. [16] gave a "diameter shallow-light" tree construction, with simultaneous low *diameter* and low cost, to enable efficient message passing and global function computation over a communication network.[5] The authors of [16] achieved tree diameter within a factor $1 + 2\epsilon$ of optimal, and tree cost within a factor $2 + \frac{2}{\epsilon}$ of optimal, for parameter $\epsilon > 0$. The BRBC method may be viewed as a straightforward "radius shallow-light" extension of Awerbuch et al.'s method in [16]. However, the motivating BRMRT problem formulation is actually quite distinct from the notion of "shallow-light": the definition of "shallow-light" implies a *sink-dependent* radius bound, but the results originally proved for BRBC establish a *net-dependent* radius bound. Specifically, [63] showed that BRBC achieves $l_i \leq (1 + \epsilon) \cdot R$, $1 \leq i \leq n$, while maintaining tree cost within $1 + \frac{2}{\epsilon}$ of optimal. The stronger result, that BRBC is also shallow-light, was obtained in [17].

The following discussion assumes a routing graph $G = (V, E)$ with $V = S$. For ease of notation, we sometimes refer to sinks without subscripts, e.g., v, x, y, etc.

[5] Ho et al. [121, 123] have also proposed heuristics for a minimum-cost bounded-diameter spanning tree formulation.

The BRBC Algorithm

The basic idea of the "shallow-light" recipe is to construct a subgraph Q of G, such that Q spans S and has both small cost and small radius. The shortest paths tree of Q will also have small cost and radius since it is a subgraph of Q, and will therefore serve as a good routing solution. The BRBC algorithm is outlined as follows (Figure 3.10 gives a more formal description):

- Compute a shortest paths tree T_S of G, and compute a minimum spanning tree T_M of G. Also, initialize the graph Q to be equal to T_M.
- Let L be the sequence of vertices corresponding to any depth-first tour of T_M; the tour will traverse each edge of T_M exactly twice (see Figure 3.9), and hence the cost of this tour is $2 \cdot cost(T_M)$.
- Traverse L while keeping a running total, Sum, of traversed edge costs. As the traversal visits each vertex L_i, check whether $Sum > \epsilon \cdot dist_G(s_0, L_i)$. If so, reset Sum to 0 and merge the edges of $minpath_G(s_0, L_i)$ into Q. Continue traversing L while repeating this process.
- Output T_{BRBC} = a shortest paths tree over Q.

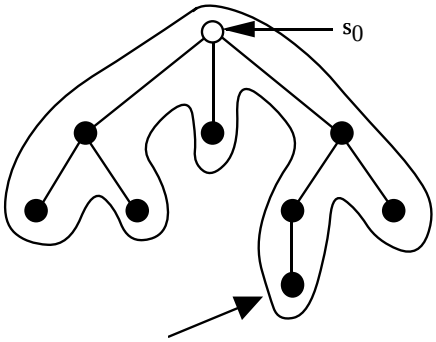


Figure 3.9 A spanning tree and its depth-first tour.

Algorithm BRBC: Computing a bounded-radius, bounded-cost spanning tree
Input: Graph $G = (V, E)$ (with radius R, source $s_0 \in V$), $\epsilon \geq 0$ **Output:** Spanning tree T_{BRBC} with $r(T_{BRBC}) \leq (1+\epsilon) \cdot R$ and $cost(T_{BRBC}) \leq (1 + \frac{2}{\epsilon}) \cdot cost(T_M)$

$Q = T_M$
$L =$ depth-first tour of T_M
$Sum = 0$
For $i = 1$ to $\|L\| - 1$
  $Sum = Sum + dist(L_i, L_{i+1})$
  If $Sum \geq \epsilon \cdot dist_G(s_0, L_{i+1})$ **Then**
  $Q = Q \cup \{$ edges in $minpath_G(s_0, L_{i+1})\}$
  $Sum = 0$
Output T_{BRBC} = shortest paths tree of Q

Figure 3.10 The BRBC algorithm. T_{BRBC} will have radius at most $(1+\epsilon) \cdot R$, and cost at most $(1 + \frac{2}{\epsilon}) \cdot cost(T_M)$.

Theorem 3.2.3 *For any weighted graph G and $\epsilon \geq 0$, $r(T_{BRBC}) \leq (1+\epsilon) \cdot R$.*

Proof: For any $v \in V$, let v_{i-1} be the last node before v on the MST traversal L for which BRBC added $minpath_G(s_0, v_{i-1})$ to Q (see Figure 3.11). By construction, $dist_L(v_{i-1}, v) \leq \epsilon \cdot R$. We then have

$$\begin{aligned} dist_{T_{BRBC}}(s_0, v) &\leq dist_{T_{BRBC}}(s_0, v_{i-1}) + dist_L(v_{i-1}, v) \\ &\leq dist_G(s_0, v_{i-1}) + \epsilon \cdot R \\ &\leq R + \epsilon \cdot R \\ &= (1+\epsilon) \cdot R \end{aligned}$$

□

Theorem 3.2.4 *For any weighted graph G and parameter $\epsilon \geq 0$, $cost(T_{BRBC}) \leq (1 + \frac{2}{\epsilon}) \cdot cost(T_M)$.*

Proof: Let $v_1, v_2, \ldots, v_m$ be the set of nodes to which BRBC added shortest paths $minpath_G(s_0, v_i)$ from the source node, and let $v_0 = s_0$. We have

$$cost(T_{BRBC}) \leq cost(T_M) + \sum_{i=1}^{m} dist_G(s_0, v_i)$$

since T_{BRBC} is a subtree of the union of T_M with all of the edges in the added shortest paths. By construction, $dist_L(v_{i-1}, v_i) \geq \epsilon \cdot dist_G(s, v_i)$ for all $i = 1, \ldots, m$, implying

$$\begin{aligned} cost(T_{BRBC}) &\leq cost(T_M) + \sum_{i=1}^{m} \frac{1}{\epsilon} \cdot dist_L(v_{i-1}, v_i) \\ &\leq cost(T_M) + \frac{1}{\epsilon} \cdot cost(L) \\ &= cost(T_M) + \frac{2}{\epsilon} \cdot cost(T_M) \\ &= (1 + \frac{2}{\epsilon}) \cdot cost(T_M) \end{aligned}$$

□

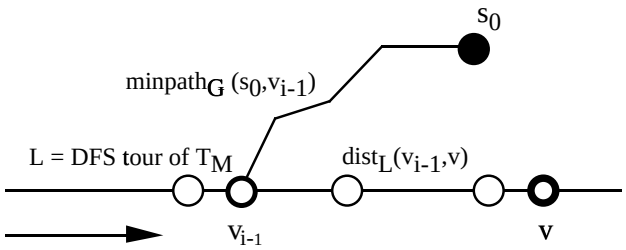


Figure 3.11 The BRBC construction.

Theorem 3.2.4 suggests that for $\epsilon = 0$, the ratio $\frac{cost(T_{BRBC})}{cost(T_M)}$ is not bounded by any constant; this is illustrated by the example of Figure 3.1, for which $\frac{cost(T_{BRBC})}{cost(T_M)}$ is $\Omega(|S|)$.

Bounded-Radius Steiner Trees

BRBC generalizes to the case where we seek to connect a subset of the vertices in the routing graph, and can use the remaining vertices as Steiner points.[6] The BRMRT problem then becomes the "Bounded-Radius Optimal Steiner Tree"

[6]This is the case for building-block VLSI design, where the underlying routing graph is the channel intersection graph as defined by Preas [193], Dai, Asano and Kuh [70] and Kimura [158]. Other very similar routing graphs have been proposed in the context of escape lines by Hightower [120] and in the context of line intersection routing by Cohoon and Richards [58].

(BROST) problem, which simplifies to the NP-complete Steiner problem in graphs when the radius bound is set to $+\infty$.

Observe that in the BROST problem, constructing a "minimum spanning tree" for S in G is itself an instance of the graph Steiner problem. A BRBC analog for the Steiner case must therefore first approximate the minimum-cost Steiner tree that connects S within G.[7] Given an approximate minimum-cost Steiner tree $\hat{T}$, the same shallow-light construction will immediately yield a routing tree with radius bounded by $(1+\epsilon) \cdot r(\hat{T})$, and cost bounded by $(1+\frac{2}{\epsilon}) \cdot cost(\hat{T})$.

The heuristic of Kou, Markowsky and Berman (KMB) [159, 249] can be used to build a Steiner tree $\hat{T} = T_{KMB}$ in the underlying routing graph, such that $cost(T_{KMB})$ will be at most twice the cost of an optimal Steiner tree T_{opt}.[8] We may traverse a depth-first tour L of T_{KMB}, adding into T_{KMB} the edges in selected shortest paths from the source to vertices of L, just as in the original BRBC method. We then compute the shortest paths tree in the resulting graph and output the union of all shortest paths from the source to terminals in S (this will include intermediate non-terminals on the shortest paths as Steiner points). We call the resulting method the **BRBC_S** algorithm.

Theorem 3.2.5 *For any weighted routing graph $G = (V, E)$, set of signal net terminals $S \subseteq V$, and parameter ϵ, $r(T_{BRBC\_S}) \leq (1+\epsilon) \cdot R$ and $cost(T_{BRBC\_S}) \leq 2 \cdot (1+\frac{2}{\epsilon}) \cdot cost(T_{opt})$.*

Proof: By the previous arguments, $r(T_{BRBC\_S}) \leq (1+\epsilon) \cdot R$. In addition, $cost(T_{BRBC\_S}) \leq (1+\frac{2}{\epsilon}) \cdot cost(T_{KMB})$. Since $cost(T_{KMB}) \leq 2 \cdot cost(T_{opt})$, we have $cost(T_{BRBC\_S}) \leq 2 \cdot (1+\frac{2}{\epsilon}) \cdot cost(T_{opt})$, thus yielding the cost bound.[9] □

[7] Strictly speaking, this analogy is not a requirement. While we have used L = a depth-first tour of a spanning tree, any tour of the vertices – hopefully with reasonably small cost – will suffice (e.g., a traveling salesman tour). The only requirement for the tour is that it visit every node in S.

[8] Recall from Section 2.7 that the KMB algorithm works as follows. Given a graph $G = (V, E)$ and a signal net $S \subseteq V$, construct the complete graph G' over the vertices in S, with each edge weight equal to the cost of the corresponding shortest path in G. Then, compute $MST_{G'}$, the minimum spanning tree of G', and expand each edge of $MST_{G'}$ into the corresponding shortest path; this yields a subgraph G'' of G that spans S. Finally, compute $MST_{G''}$ and delete pendant edges from $MST_{G''}$ until all leaves are vertices in S. Output the resulting tree as T_{KMB}.

[9] Using the graph Steiner heuristic of Zelikovsky [254], this cost bound may be further reduced to $\frac{11}{6} \cdot (1+\frac{2}{\epsilon})$ times optimal, and other bounds may similarly be reduced by the factor 1/12. However, we state all of our analyses in terms of the KMB bound since the fractions are simpler, and KMB is more widespread in the current literature (cf. works of

Improvements in Geometry

If the routing is in the geometric plane, so that we can introduce Steiner points at arbitrary locations, the basic algorithm of Figure 3.10 can be modified to introduce Steiner points on the tour L whenever $Sum = 2\epsilon \cdot R$. For each of these Steiner points, we construct a shortest path to the source and add it to Q as in the original $BRBC$ algorithm. Each node in the tour L will be within distance $\epsilon \cdot R$ of a Steiner point, i.e., within $(1+\epsilon) \cdot R$ of the source. In some sense, each shortest path to the source "services" points on L within distance $\epsilon \cdot R$ on either side of the Steiner point. Because this variant relies on an underlying geometry, we call it the **BRBC_G** algorithm. The following radius and cost bounds hold, with the proofs of these bounds following along the same lines as the proofs of Theorems 3.2.3 and 3.2.4.

Theorem 3.2.6 *In the geometric plane,* $r(T_{BRBC\_G}) \leq (1+\epsilon) \cdot R$ *and* $cost(T_{BRBC\_G}) \leq 2 \cdot (1 + \frac{1}{\epsilon}) \cdot cost(T_{opt})$. □

Well-known results which bound the worst-case ratio between the optimum Steiner tree cost and the optimum spanning tree cost in various geometries can yield even better bounds for the above scheme. Two examples are as follows.

Corollary 3.2.7 *In the Manhattan plane,* $r(T_{BRBC\_G}) \leq (1+\epsilon) \cdot R$ *and* $cost(T_{BRBC\_G}) \leq \frac{3}{2} \cdot (1 + \frac{1}{\epsilon}) \cdot T_{opt}$.

Proof: By the result of Hwang [135], the rectilinear minimum spanning tree gives a $\frac{3}{2}$ approximation to the optimal rectilinear Steiner tree.[10] We then apply arguments similar to those used for Theorems 3.2.3 and 3.2.4. □

Corollary 3.2.8 *In the Euclidean plane,* $r(T_{BRBC\_G}) \leq (1+\epsilon) \cdot R$, *and* $cost(T_{BRBC\_G}) \leq \frac{2}{\sqrt{3}} \cdot (1 + \frac{1}{\epsilon}) \cdot T_{opt}$.

Proof: By the result of Du and Hwang [78], the Euclidean minimum spanning tree gives a $\frac{2}{\sqrt{3}}$ approximation to the optimal Euclidean Steiner tree. We again apply the arguments of Theorems 3.2.3 and 3.2.4. □

Cohoon and Ganley [104] and Chiang et al. [53] which use techniques similar to KMB for global routing).

[10] Recall that the result of Berman and Ramaiyer [25] and Zelikovsky [253] imply that this constant may be further reduced to $\frac{11}{8}$, or even less [24].

This result improves with increased flexibility in the wiring geometry, e.g., if octolinear or 30-60-90 degree wiring is allowed instead of rectilinear wiring. By applying the result of [210] for λ-*geometries* (allowing angles $\frac{i\pi}{\lambda}$), a cost bound of $\frac{2}{\sqrt{3}} \cos \frac{\pi}{\lambda} \cdot (1 + \frac{1}{\epsilon})$ may be established. When λ approaches ∞, this bound approaches the bound of Corollary 3.2.8 above.

We now close the discussion of the BRBC algorithm by showing how the BRMST and BROST formulations diverge from the original shallow-light criterion above.

Sink-Dependent Bounds and the Shallow-Light Result

For certain applications, one may wish to impose different wirelength constraints on different source-sink paths within a given signal net, since the circuit timing is path-dependent rather than net-dependent. Any timing-critical path between a primary input and a primary output has two components: (i) internal module delays, and (ii) one or more source-sink connections, each of which is part of a signal net that connects an output of one module to an input of another module. Intuitively, any source-sink connection on a timing-critical path will require a small value of ϵ, whereas a source-sink connection that is not on any critical path might allow a larger value of ϵ in order to reduce tree cost. (This issue will become the focus of Section 3.3.2 below.) With this in mind, [63] addressed the following variant formulation:

The Non-Uniform Bounded-Radius Minimum Routing Tree Problem: Given a signal net with source s_0 and radius R, and given values $\epsilon_i \geq 0$ associated with the sinks s_i, find a minimum-cost routing tree T with $l_i \leq (1 + \epsilon_i) \cdot R$ for each s_i.

The BRBC method is easily modified to handle this variant, by changing the condition inside the Figure 3.10 loop from "$Sum \geq \epsilon \cdot dist_G(s, L_{i+1})$" to "$Sum \geq \epsilon_{i+1} \cdot dist_G(s, L_{i+1})$". We call this variant the $\mathbf{BRBC}_{\vec{\epsilon}_i}$ algorithm. Extensions to (geometric) Steiner routing are also straightforward. The following source-sink pathlength bound is obtained analogously to the result of Theorem 3.2.3:

Lemma 3.2.9 *For any weighted routing graph G with source s_0, radius R, and parameters $\epsilon_1, \epsilon_2, \ldots, \epsilon_n$, $dist_{T_{BRBC_{\vec{\epsilon}_i}}}(s_0, s_i) \leq (1 + \epsilon_i) \cdot R$ for each sink s_i.* □

Application of earlier arguments yields the cost bound

$$cost(T_{BRBC\_\epsilon_i}) \leq (1 + \frac{2}{min(\epsilon_1, \epsilon_2, \ldots, \epsilon_n)}) \cdot cost(T_M),$$

and the analysis in [63] establishes a somewhat better bound:

Lemma 3.2.10 *For any weighted routing graph G with source s_0 and parameters $\epsilon_1 \leq \epsilon_2 \leq \ldots \leq \epsilon_n$, $cost(T_{BRBC\_\epsilon_i}) \leq (1 + \frac{k}{k-1} \cdot \frac{2}{HM(\epsilon_1, \epsilon_2, \ldots, \epsilon_n)}) \cdot cost(T_M)$, where HM denotes harmonic mean and $k = \lceil \frac{2 \cdot cost(T_M)}{(1+\epsilon) \cdot R} \rceil$.* □

All of these bounds for the BRBC algorithm are in terms of the "net-dependent" radius objective that is inherent in the BRMST formulation, i.e., all l_i are bounded by multiples of R. Because R can be much greater than a given sink radius R_i, a bound of $l_i \leq (1 + \epsilon_i) \cdot R$ may not be meaningful in practice. Thus, a stronger and more compelling result is that of Awerbuch et al. [17], who showed that the BRBC algorithm is actually shallow-light. Recall that the proof of Theorem 3.2.3 showed

$$\begin{aligned} dist_{T_{BRBC}}(s_0, v) &\leq dist_{T_{BRBC}}(s_0, v_{i-1}) + dist_L(v_{i-1}, v) \\ &\leq dist_G(s_0, v_{i-1}) + dist_L(v_{i-1}, v). \end{aligned}$$

Awerbuch et al. (see Lemma 2.2 of [17]) use the "other triangle inequality" in observing that

$$dist_G(s_0, v_{i-1}) \leq dist_G(s_0, v) + dist_L(v_{i-1}, v).$$

This can be combined with the above relation to yield

$$\begin{aligned} dist_{T_{BRBC}}(s_0, v) &\leq dist_G(s_0, v) + 2 \cdot dist_L(v_{i-1}, v) \\ &\leq (1 + 2\epsilon) \cdot R_v \end{aligned}$$

where $R_v = dist_G(s_0, v)$.

Theorem 3.2.11 *BRBC constructs a shallow-light, $(1 + 2\epsilon, 1 + \frac{2}{\epsilon})$-tree for parameter $\epsilon \geq 0$.* □

The KRY Algorithm

The algorithm of Khuller, Raghavachari and Young (KRY) [156] provides what is essentially a best-possible shallow-light tree construction. The KRY method also follows the basic template of Awerbuch et al. in performing a DFS traversal of T_M. However, when an analog of the Sum variable exceeds the prescribed radius bound, KRY adds only a piece of the shortest path back to the source, i.e., it adds edges from the shortest path one at a time until the distance to the source is sufficiently reduced. By not adding complete shortest paths as in the BRBC approach, the cost of the construction is kept low.

For each sink $v \in S$, $v \neq s_0$, KRY maintains both a source-sink pathlength upper bound $UB[v]$ and a *parent pointer* $p[v]$. The value $UB[v]$ is an upper bound on the cost of traveling from v to s_0 in the current graph via parent pointers. All pathlength upper bounds are initially set to $UB[v] = +\infty$, and all parent pointers initially point to $p[v] = s_0$. The key operation is a "*Relax*" step which resembles a typical shortest-paths recurrence. $Relax(u, v)$ checks whether there is a "shorter" path to v through u, vis-a-vis the pathlength upper bound. In other words, if $UB[v] > UB[u] + d(u, v)$, then the algorithm sets $UB[v] \leftarrow UB[u] + d(u, v)$ and $p[v] \leftarrow u$. By calling $Relax(u, v)$ with u being the parent of v in T_S, the *Relax* operation can be used to add an edge of the v-s_0 shortest path into the solution. Figure 3.12 gives a high-level description of KRY, following the presentation in [156].

Because each edge is relaxed exactly twice during the depth-first traversal, and because at most a linear number of relaxations can result from calls in the subroutine Add-Path, KRY is a linear-time algorithm. However, it requires precomputation of both T_M and T_S, which cannot be achieved in less than $\Theta(n \log n)$ time in the geometric plane. The following results of Khuller et al. establish the shallow-light and "unimprovable" qualities of the KRY construction.

Theorem 3.2.12 *KRY constructs a shallow-light,* $(1 + \epsilon, 1 + \frac{2}{\epsilon})$*-tree for parameter* $\epsilon \geq 0$. □

Theorem 3.2.13 *For any* $\epsilon > 0$ *and any* β *with* $1 \leq \beta < 1 + \frac{2}{\epsilon}$*, there exist graphs for which no spanning* $(1 + \epsilon, \beta)$*-tree exists.* □

Khuller et al. further show that for such values of ϵ and β, it is NP-complete to even determine whether a given $G = (V, E)$ with source $s_0 \in V$ contains a

Algorithm KRY: Computing a $(1+\epsilon, 1+\frac{2}{\epsilon})$-tree

Input: Vertex set S with source s_0; T_M; T_S; $\epsilon \geq 0$
Output: Spanning tree T_{KRY} with $l_i \leq (1+\epsilon) \cdot R_i \ \forall s_i \in S$
and $cost(T_{KRY}) \leq (1+\frac{2}{\epsilon}) \cdot cost(T_M)$

Initialize $UB[v] = \infty$, $p[v] = s_0$ for all $v \in S - \{s_0\}$
Call DFS(s_0)
Return tree $T_{KRY} = \{(v, p[v] \mid v \in S - \{s_0\}\}$

Subroutine DFS(u) : Traverse subtree of T_M rooted at u,
relaxing edges to add partial paths from T_S

If $UB[u] > (1+\epsilon) \cdot R_u$ **Then**
    Add-Path(u)
For each child v of u in T_M **Do**
    $Relax(u, v)$
    DFS(v)
    $Relax(v, u)$

Subroutine Add-Path(v) : Relax along the v-s_0 shortest path

If $UB[v] > R_v$ **Then**
    u = parent of v in T_S
    Add-Path(u)
    $Relax(u, v)$

Figure 3.12 The KRY algorithm. T_{KRY} will have radius at most $(1+\epsilon) \cdot R$, and cost at most $(1+\frac{2}{\epsilon}) \cdot cost(T_M)$.

spanning $(1+\epsilon, \beta)$-tree. Improvements for the Steiner and geometric cases are straightforward, and can employ approximations of T_M and T_S as described earlier for the BRBC method. It is interesting to note that analogous shallow-light properties hold for KRY even when the signal net contains *multiple* sources [156]. This can be particularly relevant for routing of large critical nets on-chip, where a balanced tree of buffers is used to drive the many fan-ins and reduce rise-time delays. Essentially, the leaves of the buffer tree will correspond to multiple sources in the net routing problem.[11]

[11] Other applications of multiple-source routing arise in clock distribution, e.g., with a very large monolithic buffer that must be treated as multiple sources [76, 175], with a hierarchical buffering scheme, or with two-level clock routing in MCM packaging [260]. Clock routing generally also demands skew control, so there is only a partial connection to the present discussion.

3.2.3 The Prim-Dijkstra Tradeoff

It is well-known that the min-cost and min-radius objectives can be *separately* addressed by Prim's MST algorithm [196] and Dijkstra's SPT algorithm [74], respectively. Tarjan [235] discusses the similarity between the Prim and Dijkstra algorithms: each is a variant of the "labeling method" that builds a spanning tree from a fixed source by adding the edge that minimizes an algorithm-specific *key*. In the following, the min-cost and min-radius objectives are addressed *simultaneously* via direct combinations of the Prim and Dijkstra constructions. The combination of competing objectives, via a tradeoff of algorithms that are respectively optimal for each objective, is somewhat unusual. While the two Prim-Dijkstra tradeoff constructions that we discuss are not shallow-light, they can be implemented to run in $O(n^2)$ time and in practice yield lower signal delays than the shallow-light constructions. Our discussion is cast in geometry, but the methods involved are applicable to general weighted graphs.

The PD1 Tradeoff

Prim's algorithm begins with the tree consisting only of s_0. The algorithm then iteratively adds edge e_{ij} and sink s_i to T, where s_i and s_j are chosen to minimize

$$d_{ij} \quad s.t. \quad s_j \in T,\ s_i \in S - T \tag{3.2}$$

Dijkstra's algorithm also begins with the tree consisting only of s_0. The algorithm then iteratively adds edge e_{ij} and sink s_i to T, where s_i and s_j are chosen to minimize

$$l_j + d_{ij} \quad s.t. \quad s_j \in T,\ s_i \in S - T \tag{3.3}$$

The similarity between (3.2) and (3.3) is the basis for the PD1 tradeoff, which iteratively adds edge e_{ij} and sink s_i to T, where s_i and s_j are chosen to minimize

$$(c \cdot l_j) + d_{ij} \quad s.t. \quad s_j \in T,\ s_i \in S - T \tag{3.4}$$

for some choice of $0 \leq c \leq 1$. When $c = 0$, PD1 is identical to Prim's algorithm and constructs trees with minimum cost. As c increases, PD1 constructs a tree

with higher cost but lower radius, and is identical to Dijkstra's algorithm when $c = 1$. Sample executions of PD1 for $c = \frac{1}{3}$ and $c = \frac{2}{3}$ are shown in Figure 3.13(a)-(b). The following properties of the PD1 tradeoff were shown by Alpert et al. in [13, 14].

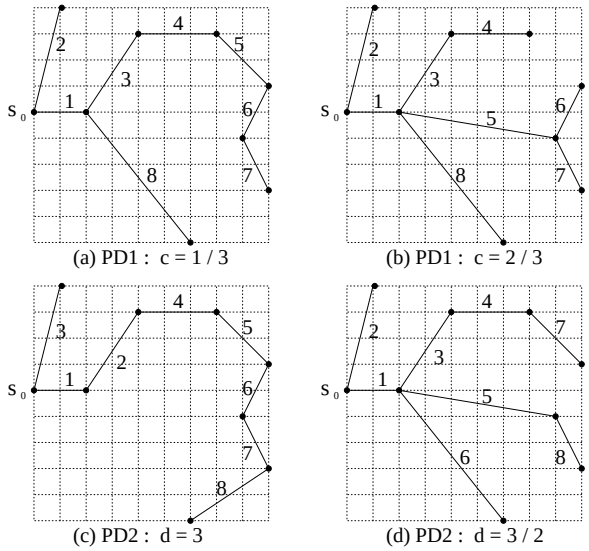


Figure 3.13 Execution of PD1 and PD2 on an 8-sink instance in the Euclidean plane. Edge labels indicate the order in which the algorithms add edges into the tree. PD1 is illustrated in (a) with $c = \frac{1}{3}$ (radius = 15.91, cost = 26.43), and in (b) with $c = \frac{2}{3}$ (radius = 10.32, cost = 29.69). PD2 is illustrated with "corresponding" parameterizations (see Table 3.2 below) in (c) with $p = 3$ (radius = 17.00, cost = 23.63), and in (d) with $p = \frac{3}{2}$ (radius = 10.00, cost = 30.28).

Lemma 3.2.14 *PD1 constructs a tree T_{PD1} with $c \cdot l_i \leq R_i$ for all sinks s_i.* □

Lemma 3.2.15 *For any fixed values of c and B with $0 < c < 1$ and $B > 0$, there exists an edge-weighted graph instance $G = (S, E)$ for which PD1 will yield a tree with $cost(T_{PD1}) > B \cdot cost(T_M)$.* □

Subsequently, Lenhof, Salowe and Wrege [169] showed a bound for the PD1 tree cost in geometry:

Lemma 3.2.16 *For instances embedded in Euclidean space of any dimension d, PD1 constructs a tree with cost within an $O(\log n)$ factor of cost(T_M).*

This result provides some encouragement regarding the still-open conjecture in [12, 13] that PD1 is actually shallow-light for geometric instances.

The PD2 Tradeoff

The notation L_p, as defined in Chapter 2, usually denotes a vector norm for $p > 0$. Here, we say that the L_p sum of quantities $x_1, x_2, \ldots, x_n$ has value $({x_1}^p + {x_2}^p + \ldots + {x_n}^p)^{1/p}$, and we write this as $||x_1, x_2, \ldots, x_n||_p$. Now, observe that within the framework developed by [235], Dijkstra's algorithm can be viewed as using a key that is the L_1 sum of edge costs in the source-sink path. This observation suggests a second Prim-Dijkstra tradeoff, which we call PD2: iteratively add edge e_{ij} and sink s_i to T, where s_i and s_j are chosen to minimize

$$||{l_j}^p \ , \ d_{ij}||_p \quad s.t. \quad s_j \in T, \ s_i \in S - T \tag{3.5}$$

for some choice of $1 \leq p < \infty$, where ${l_j}^p$ denotes the L_p sum of edge costs in the s_0-s_j path in T. Sample executions of PD2 for $p = 3$ and $p = \frac{3}{2}$ are shown in Figure 3.13(c)-(d).

Lemma 3.2.17 *When $p = \infty$, PD2 is identical to Prim's algorithm.* □

Lemma 3.2.18 *When $p = 1$, PD2 yields a shortest path tree.* □

When $p = \infty$, the PD2 objective reduces to $\max\{|l_j|, d_{ij}\}$, which corresponds to the bottleneck shortest paths formulation. In other words, if the cost of a path is the cost of the longest edge in that path, then PD2 constructs a shortest-path tree in this sense when $p = \infty$. Notice that once a bottleneck edge with large cost is present in some source-sink shortest path, a bottleneck shortest-path tree is maintained as long as we append *any* edge that has less cost than

the bottleneck edge. Thus, the optimal bottleneck tree is not unique. So that PD2 with $p = \infty$ will capture the limiting behavior from large finite values of p, and furthermore return a minimum spanning tree, Alpert et al. specify that ties in Equation (3.5) should be broken by choosing the s_i which also minimizes d_{ij}. This tie-breaking rule allows (3.5) to capture a Prim-Dijkstra tradeoff. In other words, PD2 returns T_S for $p = 1$, and returns T_M for $p = \infty$ when the tie-breaking rule is applied.

The PD2 tradeoff was discovered independently by Alpert et al. [14] and by Salowe, Richards and Wrege [207], with the work of both groups prompted by the original PD1 tradeoff [12, 13]. Salowe et al. discovered PD2 by applying Tarjan's general single-source shortest path labeling method [235] to the "bottleneck shortest paths" problem,[12] i.e., they use the label $\max\{|l_j|, d_{ij}\}$, where $|l_i|$ denotes the largest edge cost in the s_0-s_i path, thus generalizing the objective of Equation (3.5).

Salowe et al. [207] have shown that PD2 constructs a tree T with $l_i \leq R_i \cdot n^{1-1/p}$ for all sinks s_i, and that this bound is tight. For any finite value of p, PD2 may yield a tree with cost an unbounded factor greater than the MST cost, even in geometry.

3.2.4 Rectilinear Steiner Arborescences

So far, this section has developed essentially geometric methods that are tunable to given technology parameters via the cost-radius (or T_M-T_S) tradeoff that follows from analysis of Elmore delay. One non-tunable method has been motivated by the Elmore delay *upper bound* of Rubinstein, Penfield and Horowitz [205]. Recall that this Elmore delay upper bound is obtained by summing the product of a node's capacitance and its "upstream" resistance (i.e., between the node and s_0), over all node locations in the routing tree. Minimizing the value of this upper bound is a net-dependent objective, in that the same upper bound applies to every sink in the tree. Cong, Leung and Zhou [65] show that a routing tree which minimizes this objective will combine elements of the minimum spanning tree, the shortest paths tree, and the "quadratic minimum Steiner tree" (a tree that minimizes the summation of source-node pathlengths, taken over all possible node locations). Therefore, a minimum-cost *rectilinear Steiner arborescence* is of interest since it heuristically addresses all of these terms in the decomposed upper bound at once.

[12] Cf. the "min-max" routing trees in weighted layout regions of Chiang et al. [51].

Definition: Given a signal net S in the Manhattan plane with source s_0, a *rectilinear Steiner arborescence* (RSA) is a Steiner tree T that spans S, with $l_i = R_i$ for all sinks s_i.

The **RSA Problem** seeks a minimum-cost RSA, i.e., a "minimum-cost shortest-path Steiner tree" (recall the "maximum-performance tree" sought by Cohoon and Randall [57]) as shown in Figure 3.14. The RSA problem has been reviewed at length by Rao et al. [201], who ascribe it to the 1979 Ph.D. work of Ladeira de Matos [71]. The problem's complexity is still open; cf. the works of Trubin [239] and Rao et al. [201]. Ho et al. [122] note that an RSA may be viewed as a "rectilinear multicast" in contexts outside of VLSI routing. Previous analysis has often dealt with the case where all sinks of S lie in the first quadrant, with s_0 at the origin (e.g., [201]). If this case can be solved in polynomial time, there is an easy polynomial-time solution for the general case of sinks located in all four quadrants (e.g., [65, 122]).

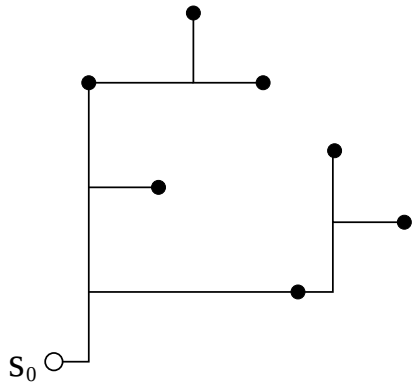


Figure 3.14 A minimum-cost rectilinear Steiner arborescence.

Given a signal net located in the first quadrant of the Manhattan plane, the heuristic of [201] maintains a set called *ROOT* consisting of roots of subtrees which will eventually merge to form the heuristic RSA solution. Initially, *ROOT* contains the roots of n trivial trees (the sinks in S, located in the first quadrant with respect to the source). The method then iteratively replaces a pair of roots by a single "merged" root that is as far as possible from the source, and terminates when $|ROOT| = 1$. More formally:

1. Given signal net S with s_0 at the origin and $s_1, \ldots, s_n$ in the first quadrant, place the n sinks of S in $ROOT$. Initialize the output tree T to be empty.
2. LOOP:
3. Find $p, q \in ROOT$ which maximize the sum $min(p_x, q_x) + min(p_y, q_y)$, i.e., the sum of the minimum x- and y-coordinates of p and q.
4. Update $ROOT$ by replacing p and q by a new root with coordinates $(min(p_x, q_x), min(p_y, q_y))$.
5. Update T by adding edges to the new root from p and from q.
6. UNTIL $|ROOT| = 1$.

Figure 3.15 shows how the construction maintains the feasibility of achieving shortest-possible paths from the source to all sinks. The tree is constructed bottom-up ("outside-in"), always choosing a new root that is dominated by two existing roots and that allows the greatest flexibility for later roots. Note that it is possible for the new root to be either p or q, e.g., if p dominates q, then the new root will be q itself.

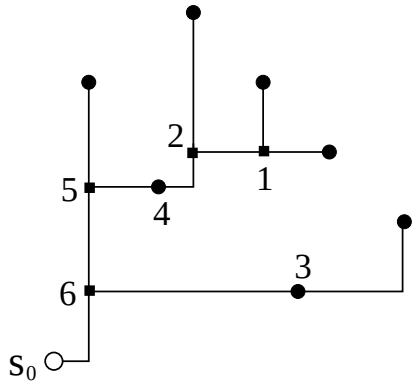


Figure 3.15 Illustration of the RSA heuristic of Rao et al.

Under certain conditions, Rao's heuristic "does the right thing", i.e., two roots are merged or a single root is partially extended without worsening any previous suboptimality in the construction. For example, consider two roots $p, q \in ROOT$ with q being the root with minimum $d(p, q)$ that is that is dominated by p (if there are ties, let q be the rightmost such root). Let *closest_x* denote

the root which is to the upper left of p and has maximum x-coordinate, if such a root exists. Similarly, let $closest\_y$ be the root which is to the lower right of p and has maximum y-coordinate, if such a root exists. If $p_x - closest\_x_x \geq d(p,q)$ and $p_y - closest\_y_y \geq d(p,q)$, then it is clearly optimal for p to connect directly to q. Cong et al. [65] call such a connection a *safe move* because it cannot worsen the suboptimality of an existing set of roots; see Figure 3.16(a) for an illustration.

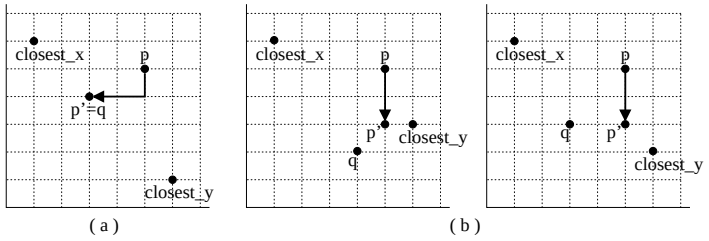


Figure 3.16 Illustration of safe moves in the heuristic RSA construction.

A second example involves root p with $p_x - closest\_x_x \geq d(p,q)$ and $p_y - closest\_y_y < d(p,q)$. In this case, it is safe to introduce a tree edge from p downward to p', where $p'_x = p_x$ and $p'_y = max(closest\_y_y, q_y)$; Figure 3.16(b) shows the two possible results. The symmetrical configuration has $p_x - closest\_x_x < d(p,q)$ and $p_y - closest\_y_y \geq d(p,q)$, and introduces a safe connection from p leftward to $p' = (max(closest\_x_x, q_x), p_y)$. In this symmetrical case, q must be the leftmost root with minimum $d(p,q)$ that is dominated by p.

When no safe move is possible, [65] applies the method of Rao et al., and terms the resulting connection(s) to the new root $(min(p_x, q_x), min(p_y, q_y))$ a "heuristic move". The resulting strategy is called the **A-tree** algorithm. A construction that uses only safe moves will be optimal, and techniques to bound the suboptimality associated with heuristic moves give rise to an optimal algorithm with exponential worst-case runtime. [65] reports that for 4-, 8- and 16-sink nets, the A-tree method produces trees with cost within 4% of the optimal RSA cost. However, despite using only "heuristic" moves, the algorithm of Rao et al. has essentially the same empirical performance as A-tree. (An interesting question is whether there are examples "with no ties" for which A-tree is better than the method of Rao et al.)

With respect to performance bounds, Rao et al. prove that their heuristic has performance ratio ≤ 2 when instances lie in the first quadrant. Since the A-tree algorithm uses either safe moves or the same moves as Rao et al., it trivially also has performance ratio ≤ 2. The example of Figure 3.17 shows that these bounds are tight, i.e., both algorithms can be forced to return tree cost that approaches twice optimal. The construction in the figure consists of an array of sinks with vertical distance = 2 and horizontal distance = 1 between adjacent sinks in the array. The column of sinks at the right is offset downward by distance $1 - \epsilon$ so that there are no ties and no safe moves.

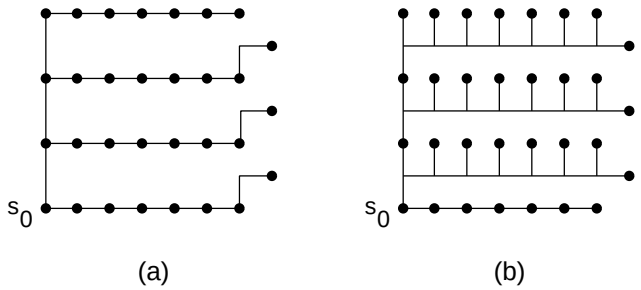


Figure 3.17 A pathological instance for both A-tree and the method of Rao et al. The solution (a) is optimal, while the solution (b) will be returned by either heuristic.

Interestingly, the I1S approach of Chapter 2 can be modified to yield an effective "Iterated 1-Arborescence" (I1Arb) methodology. As one would expect, I1Arb iteratively selects single Steiner points to minimize the cost of the *spanning* arborescence over the sinks and Steiner points selected thus far (recall that optimal spanning arborescences are efficiently computable). Alexander et al. [4] report that this approach is attractive for a variety of reasons, including its more natural ability to address the general case where sinks occur in all four quadrants. On uniformly random inputs, I1Arb has essentially the same performance as A-tree and Rao et al. for small examples, and slightly outperforms these previous heuristics for $n > 20$. I1Arb also escapes such pathological instances as the one illustrated in Figure 3.17: we do not yet know of examples for which the I1Arb tree cost is greater than $\frac{7}{6}$ times optimal in one quadrant, or greater than $\frac{5}{4}$ times optimal in all four quadrants.

3.2.5 Experimental Results and Discussion

We now discuss experimental results for the cost-radius tradeoffs achieved by the geometric approaches in this section, as well as the implications for minimum-delay signal routing.

Comparison of Cost-Radius Tradeoffs

All of the algorithms discussed in this section were implemented using ANSI C in the Sun environment. Tests were made using signal nets having up to 50 sinks, with sink locations randomly generated from a uniform distribution in the 1000 x 1000 grid. Figures 3.18(a)-(b) show that BPRIM produces a smooth tradeoff between tree cost and tree radius,[13] and a similar tradeoff is seen for BRBC in Figures 3.18(c)-(d). As ϵ decreases, each of the cost and radius curves shifts monotonically from that of the minimum spanning tree to that of the shortest paths tree.

For any given value of ϵ, the BPRIM approach is greedier than the BRBC approach, and tends to yield a routing solution with small cost but radius approaching $(1 + \epsilon) \cdot R$. The BRBC approach is more conservative, and tends to yield a routing solution with slightly larger tree cost but radius noticeably smaller than $(1 + \epsilon) \cdot R$. Thus, the cost-radius curve for BRBC is shifted slightly from that of BPRIM. In practice, efficiency and provability would suggest choosing BRBC or another shallow-light method over BPRIM.

More detailed experiments in [14] have compared the Prim-Dijkstra tradeoffs PD1 PD2, the KRY construction, the BRBC construction, and the standard MST construction. Notice that for a given signal net instance, each cost-radius tradeoff generates a *family* of spanning trees corresponding to the range of parameter values. The study of such families of output trees is useful for determining the parameter values best suited to particular technology or area-performance requirements.

For each signal net, we generated a "family" of 51 output trees for PD1 with values of the parameter c ranging from 0 to 1 at intervals of 0.02. To generate corresponding families of trees for PD2, KRY and BRBC, input parameters

[13] Cong et al. [63] report that on average, variant H1 dominates BPRIM, H2 dominates H1, and H3 dominates H2; here, "dominates" implies a smaller average cost for any given radius bound. The qualitative nature of the tradeoff remains for all variants, with the cost curves simply being shifted downward for the more sophisticated variants. It should be noted that while $\epsilon = 2.00$ does not guarantee that $T_{BPRIM} = T_M$, this generally holds in practice.

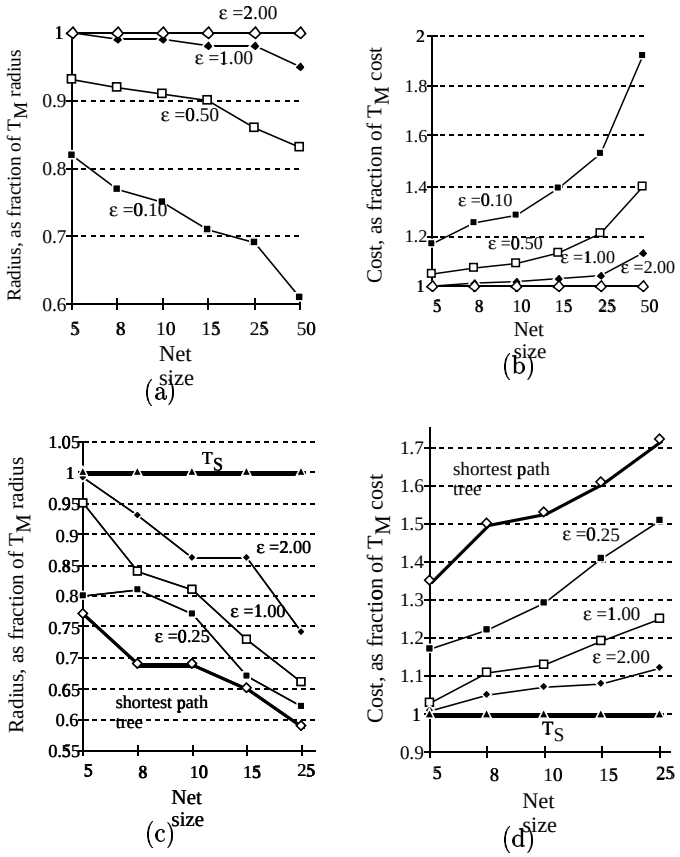


Figure 3.18 Tradeoff between tree cost and tree radius produced by the BPRIM (a-b) and BRBC (c-d) algorithms. As the parameter ϵ increases, the tree cost approaches $cost(T_M)$.

were matched with the PD1 parameter values according to relationships inferred from the algorithms' limiting behaviors (see Table 3.2). Use of these relationships generally leads to a good sampling of the families of trees generated by PD2 and KRY. However, since BRBC tends to generate trees virtually

identical to T_M for $\epsilon \geq 1.5$, we study the family of 51 trees generated by BRBC with ϵ ranging from 0 to 1.5 at intervals of 0.03.

	PD1	PD2	BRBC	KRY
User parameter	c	p	ϵ	α
Yields T_M when	$c = 0$	$p = \infty$	$\epsilon = \infty$	$\alpha = \infty$
Yields T_S when	$c = 1$	$p = 1$	$\epsilon = 0$	$\alpha = 1$
Relation to c	c	$p = \frac{1}{c}$	$\epsilon = \frac{1-c}{c}$	$\alpha = \frac{1}{c}$

Table 3.2 Equivalences of algorithm parameters.

Each algorithm was executed over its family of parameter values, for signal nets of 16 sinks chosen randomly from a uniform distribution in a 1cm by 1cm Manhattan square. The results are shown in the graphs of Figure 3.19; each point in the graphs represents an average over 250 such instances. All four algorithms "smoothly" trade off between cost and radius, with PD1 being clearly superior, i.e., for any desired cost-radius tradeoff, PD1 performs uniformly better than the other algorithms. The utility of PD1 is especially clear for the tradeoff region that is of likely practical interest, when we wish to reduce tree radius without sacrificing more than 10% or 20% extra tree cost. While PD2 does not do as well as PD1, it nevertheless seems to provide superior cost-radius tradeoffs when compared against KRY or BRBC.

Comparison of Signal Delays

In [14], other experiments compared the signal delays of the various tree constructions, using uniformly random signal nets of 4, 8 and 16 sinks (PD1 dominates the performance of PD2, and so we discuss results only for the former). For each of the four interconnect technologies in Table 3.1, delays at all sink nodes were computed using the Two-Pole simulator code developed by Zhou et al. [256] and corrected by S. Muddu [187]. The Two-Pole simulator uses moment-matching techniques to model the response of distributed RLC interconnects, and has been reported to produce very accurate results when tested against SPICE3e [256]; cf. the discussion of accuracy and fidelity in the Appendix. Delay was measured as the rise time to a stable value of 0.9 times the reference voltage for a step input. For each instance, each algorithm was executed with each of the 51 user parameters described above, and the lowest delay value of any tree in the family was recorded.

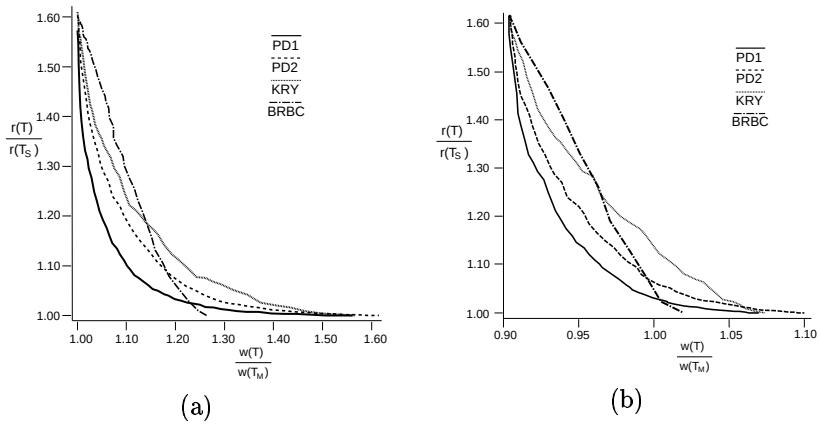


Figure 3.19 Graph of radius ratio $\frac{r(T)}{r(T_S)}$ versus cost ratio $\frac{cost(T)}{cost(T_M)}$ for PD1, PD2, KRY and BRBC for uniformly random instances of 16 sinks in the square Manhattan grid. Each point indicates the algorithm performance for a specific parameter value, averaged over 250 instances. The graph (a) shows results for spanning tree topologies, and (b) shows the same experiments with tree edges overlapped to induce a Steiner topology.

Table 3.3 shows the resulting maximum sink delays, normalized to the corresponding value for the MST routing, and averaged over 250 instances. Analogous results for average sink delay are qualitatively similar [14]. The table also shows the average value of each algorithm's best parameterization, which indicates how the ideal cost-radius tradeoff parameter is correlated with technology and net size (for example, the best PD1 c parameter for 16 sinks is 0.23 for IC1 and 0.73 for MCM). If only one spanning tree construction is allowed, it seems that the "best" parameter will generally yield a tree with low delay. The delay reductions achieved by the Prim-Dijkstra tradeoff reinforce the intuition that minimum-cost routing trees are less useful for newer interconnect technologies.

The KRY delays are surprisingly good in view of the algorithm's inferior cost-radius tradeoff. While PD1 seems to yield a more "natural" tree (e.g., the KRY tree is often self-intersecting – see Figure 3.20), KRY does benefit from its tendency to branch early from the source s_0, which results in relatively

Spanning Trees		Avg sink delay vs. MST (best parameter)			
#sinks	Method	IC1	IC2	IC3	MCM1
4	PD1	0.911 (0.10)	0.866 (0.32)	0.854 (0.34)	0.712 (0.55)
	KRY	0.912 (19.56)	0.866 (16.22)	0.854 (15.83)	0.712 (8.10)
	BRBC	0.928 (0.09)	0.891 (0.10)	0.880 (0.10)	0.768 (0.11)
8	PD1	0.808 (0.15)	0.778 (0.47)	0.759 (0.49)	0.540 (0.75)
	KRY	0.850 (9.42)	0.781 (4.83)	0.760 (3.96)	0.540 (1.79)
	BRBC	0.899 (0.08)	0.848 (0.06)	0.834 (0.05)	0.678 (0.04)
16	PD1	0.800 (0.20)	0.720 (0.48)	0.697 (0.50)	0.429 (0.82)
	KRY	0.808 (3.57)	0.723 (1.99)	0.696 (1.87)	0.424 (1.19)
	BRBC	0.893 (0.12)	0.839 (0.13)	0.824 (0.13)	0.648 (0.12)

Table 3.3 Average source-sink delay in the best tree for each algorithm. Values are given as a ratio to corresponding MST delay values, averaged over 250 random instances. Numbers in parentheses give the average best parameter value for each algorithm.

little off-path tree capacitance for any given source-sink path. While the Prim-Dijkstra methods offer advantages over previous performance-driven routing constructions, the success of KRY underscores the continuing need for better routing tree analysis and design techniques.

Steiner Routing

Recall that many global routing approaches require rectilinear Steiner tree constructions. As we have discussed, a popular approach (cf. the MST-Overlap discussion in Chapter 2) converts a spanning tree to a Steiner tree by overlapping the embeddings of tree edges within the union of their bounding boxes. This has the advantage of preserving the spanning tree radius within the eventual Steiner tree output. While Ho et al. [124] provided the optimal edge-overlapping construction, it cannot always be applied to the present spanning constructions. This is because high-degree nodes may occur, so that the spanning tree fails to be separable. Thus, in the following we discuss simulation results for Steiner trees that are obtained by applying a greedy edge-overlapping algorithm to the spanning constructions. Our greedy method considers each pair of adjacent edges in the tree, and calculates the cost reduction achievable by optimally overlapping these two edges (i.e., inducing a Steiner point).

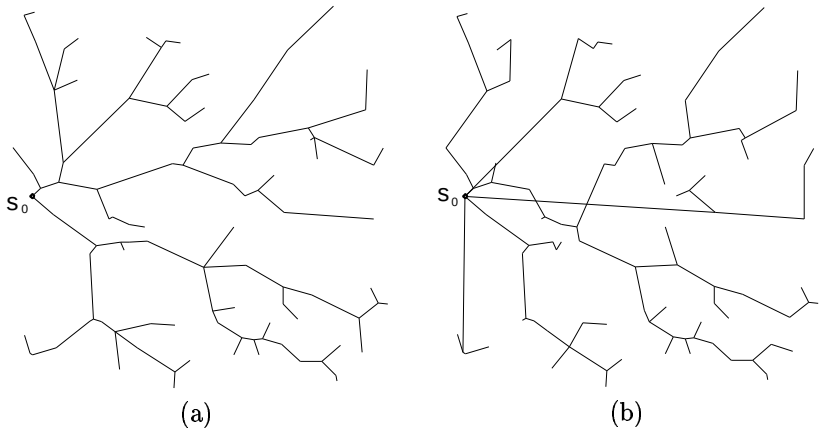


Figure 3.20 Execution of PD1 with $c = 0.5$ (a) and KRY [156] with $\alpha = 1.5$ (b), on a 100-sink example using Euclidean distance.

The candidate Steiner point (i.e., the overlapping of two edges) which yields maximum cost savings is iteratively added until no further cost reduction is possible.[14]

Figure 3.19(b) and Table 3.4 show that the performance-driven spanning tree constructions with lowest delay still have lowest delay when Steiner points are incorporated. Interestingly, the average best values of the input parameters shift to more star-like spanning topologies when the Steiner conversion is employed. This is because edge-overlapping decreases cost without affecting radius, so that it is advantageous to use spanning trees with higher cost and lower radius. Some anomalies may result since the overlapping process

[14]The output of this heuristic is nearly identical to that of the optimal edge-overlapping algorithm of Ho et al. (called S-RST in [124]). For random 10-node instances, greedy edge-overlapping averages 8.8% cost reduction from an input minimum spanning tree, while S-RST is reported to average 9.0% reduction. For random 25-node instances, the greedy heuristic averages 9.3% percent cost reduction over the minimum spanning tree, while S-RST is reported to average 9.5% reduction.

diminishes the star-like nature of the tree topology, i.e., a Steiner tree can have greater sink delay than its spanning tree precursor.[15]

Steiner Trees		Avg sink delay vs. MST (best parameter)			
#sinks	Method	IC1	IC2	IC3	MCM1
4	PD1	0.807 (0.25)	0.775 (0.27)	0.763 (0.28)	0.694 (0.32)
	KRY	0.807 (26.12)	0.776 (24.37)	0.763 (24.56)	0.694 (21.85)
	BRBC	0.817 (0.06)	0.787 (0.05)	0.775 (0.06)	0.716 (0.06)
8	PD1	0.749 (0.50)	0.696 (0.55)	0.680 (0.56)	0.551 (0.64)
	KRY	0.751 (9.86)	0.698 (6.15)	0.682 (5.55)	0.550 (3.38)
	BRBC	0.777 (0.18)	0.735 (0.15)	0.720 (0.17)	0.625 (0.13)
16	PD1	0.711 (0.52)	0.644 (0.60)	0.624 (0.62)	0.443 (0.75)
	KRY	0.715 (3.99)	0.647 (2.48)	0.628 (2.28)	0.445 (1.29)
	BRBC	0.765 (0.24)	0.719 (0.25)	0.702 (0.25)	0.579 (0.29)

Table 3.4 Average source-sink delay in the best tree for each algorithm, after edge-overlapping has been used to induce a Steiner routing. Values are given as a ratio to corresponding MST delay values, averaged over 250 random instances. Numbers in parentheses give average best parameter value for each algorithm.

Finally, we observe that the *tunable* cost-radius tradeoffs are more useful than leading "fixed" methods, which cannot be parameterized to change with the interconnect technology. In particular, the improvements afforded by PD1 over the fixed constructions discussed above (A-trees or MST-Overlap heuristic SMTs) can be substantial. Alpert et al. [14] have described a simple comparison versus the A-tree results reported in [65], based on normalizing to the performance of BRBC with $\epsilon = 1.0$ (this is possible since identical MCM interconnect parameters and Two-Pole simulation methodology were used in [14] and [65]). Even with a fixed "best" parameter value for the technology, PD1 has over 25% expected delay reduction versus A-tree. We have found that delay reductions of similar magnitude are expected over the traditional MST-Overlap construction for minimum-cost routing trees.

[15] Highly star-like topologies can possibly introduce other difficulties such as crossing wires, nodes with degree > 4, and coupling effects. These effects are ignored in this chapter, since they are generally transparent to the SPICE and Two-Pole simulation methodologies.

3.3 MINIMIZATION OF ACTUAL DELAY

The geometric minimum tree cost, bounded tree radius, and cost-radius tradeoff objectives of Section 3.2 were motivated by analysis of the Elmore delay approximation (see also the detailed discussion in the Appendix). However, such objectives are *abstractions*; they do not directly optimize Elmore delay. Indeed, the relevance of a given geometric objective often depends on the prevailing technology, on the particular distribution of sink locations for a given signal net, and on the user's ability to find the parameter value (ϵ in BRBC, or c in PD1) which yields a good solution for a particular instance. Thus, we now discuss methods which remove any abstraction of delay in the routing objective, in that a high-quality delay estimate is optimized *directly*.

3.3.1 Greedy Optimization of Elmore Delay

Much of this section will focus on variants of a greedy, yet demonstrably near-optimal, approach to minimum-delay routing that was proposed by Boese, Kahng and Robins in [34]. This method optimizes Elmore delay directly as the routing tree is constructed. The simplest embodiment of this Elmore-based approach is the *Elmore routing tree* (ERT)spanning tree construction (Figure 3.21). The ERT algorithm is analogous to Prim's MST construction [196]. It starts with a trivial tree $T = (V, E)$ containing only the source s_0, and iteratively finds the terminal $s_i \in V$ and the sink $s_j \in S - V$ such that adding edge (s_i, s_j) to T results in a tree with minimum Elmore delay. In other words, each added sink minimizes $max_{s_k \in V} t_{ED}(s_k)$, the maximum Elmore delay at any sink in the growing tree.[16] This approach recalls the method of Prasitjutrakul and Kubitz [192], which uses A* search and the Elmore delay formula in a performance-driven routing tree construction. The method of [192] also grows a routing tree over S starting from the source s_0; A* search in a routing graph (e.g., the channel intersection graph) is used to find the Elmore delay-optimal Steiner connection from a new sink to the existing tree. The key distinction from ERT is that Prasitjutrakul and Kubitz do not allow any choice in picking this new sink: their algorithm always adds the sink that is closest to the existing tree, and thus ignores the underlying Elmore delay criterion. This difference in the order of adding sinks can be seen in Figure 3.22, which depicts the progress of the ERT variant which returns a Steiner, rather than spanning, topology.[17]

[16] Recall that $t_{ED}(s_i, s_j)$ is the Elmore delay between sinks s_i and s_j, and that $t_{ED}(s_i)$ is the shorthand notation for $t_{ED}(s_0, s_i)$. When no particular delay model is assumed, $t(s_i, s_j)$ or $t(s_i)$ will be used to denote the analogous quantities.

[17] In the example shown, Prasitjutrakul and Kubitz would add sink 3 before sink 2, and sink 5 before sink 4. An instance with a much greater difference between ERT and the

The greedy approach implicit in the ERT algorithm can be generalized to any delay model by applying the appropriate estimator in Line 3 of Figure 3.21. For example, Zhou et al. [258] propose the use of calls to their Two-Pole simulator within a similar greedy construction; Sriram and Kang [228] also suggest this strategy for MCM routing using a second-order delay model.

ERT Algorithm	
Input: signal net S with source $s_0 \in S$	
Output: routing tree T over S	
1.	$T = (V, E) = (\{s_0\}, \emptyset)$
2.	**While** $\|V\| < \|S\|$ **do**
3.	Find $s_i \in V$ and $s_j \in S - V$ that minimize the maximum Elmore delay from s_0 to any sink in the tree $(V \cup \{s_j\}, E \cup \{(s_i, s_j)\})$
4.	$V = V \cup \{s_j\}$
5.	$E = E \cup \{(s_i, s_j)\}$
6.	**Output** resulting spanning tree $T = (V, E)$

Figure 3.21 The ERT Algorithm: direct incorporation of the Elmore delay formula into a greedy routing tree construction.

Fact 3.3.1 *The ERT algorithm can be implemented to run in $O(n^3)$ time, assuming that unit wire resistance, unit wire capacitance, and sink loading capacitance are all fixed constants.*

Proof: If a new tree edge incident to sink $s_i \in V$ (Line 3 of Figure 3.21) minimizes the maximum Elmore delay $max_{s_k \in V} t_{ED}(s_k)$, it must connect s_i to the sink $s_j \notin V$ that is closest to s_i. Thus, at each pass through the **while** loop, we simply compute the shortest "outside connection" for each possible $s_i \in V$ in $O(n^2)$ time, and then add each of the $O(n)$ shortest outside connections to T in turn. Evaluating the Elmore delays at all sinks in each of the resulting trees requires $O(n)$ time per tree. Hence, each pass through the **while** loop requires $O(n^2)$ time, implying $O(n^3)$ total time complexity. □

The same ERT approach can yield a Steiner routing when the new sink s_j is allowed to connect to an edge of the existing tree, possibly inducing a Steiner

algorithm of [192] would consist of many sinks closely spaced along a long path. The method of [192] forces the sinks to be added into the tree according to the path order, which can be suboptimal. Another difference from the method of [192] is that ERT does not consider obstacles, i.e., it still maintains a largely geometric perspective.

node on this edge at its point closest to s_j. (The embedding of each L-shaped edge remains undetermined until a Steiner node is placed on it.) Because star-like topologies can be optimal, we also allow a connection directly to the source. Thus, the number of ways in which $s_j \notin V$ can be added is at most the number of nodes in the current tree. The resulting *Steiner Elmore routing tree* (SERT) algorithm modifies Line 3 in Figure 3.21 to find $s_j \notin V$ and $(v, v') \in E$, such that connecting s_j to the closest point x on edge (v, v') minimizes the maximum source-sink Elmore delay in the resulting tree. Assuming that x is distinct from v and v', Line 4 is then modified so that $V = V \cup \{s_j, x\}$ and Line 5 is modified so that $E = E \cup \{(v, x), (v', x), (x, s_j)\} - \{(v, v')\}$. Figure 3.22 shows the SERT construction for an 8-sink signal net using the IC2 technology parameters from Table 3.1 above.

No SERT implementation is known that is faster than $O(n^4)$. Obstacles to a faster implementation seem to be: (i) in the modified Line 3 of the algorithm template, $\Theta(n^2)$ Steiner connections must always be considered, and (ii) it is possible that the connection which minimizes $max_k t_{ED}(s_k)$ does not minimize the delay at any individual sink in T.

3.3.2 The Critical-Sink Routing Tree Problem

Within our taxonomy of minimum-delay routing heuristics, ERT and SERT are "generic", *net-dependent* approaches. This terminology becomes clear when we consider the role of signal net routing in the overall layout process. In broad terms, performance-driven layout of cell-based designs entails determination of timing-critical paths by static timing analysis, after which cells in these paths are placed close together (see, e.g., [77, 119, 140, 173, 178, 234]). The static timing analysis thus *iteratively* drives changes within both the cells placement and the global routing phases.

Existing performance-driven placement algorithms may be classified as either *net-dependent* or *path-dependent*. *Net-dependent* placement typically uses centroid-connected star cost [227], probabilistic estimates of Steiner tree cost [141], minimum spanning tree cost [77] or bounding box semiperimeter [178] to estimate wire capacitance and signal delay for a multi-terminal net. From this information, critical timing paths between primary inputs and primary outputs are computed, and module placements are then updated to reduce these "net-dependent" objectives for signal nets along the critical paths. *Path-dependent* placement considers the delay between the source and a particular *critical sink* of a multi-terminal net. The critical sink is typically determined via timing

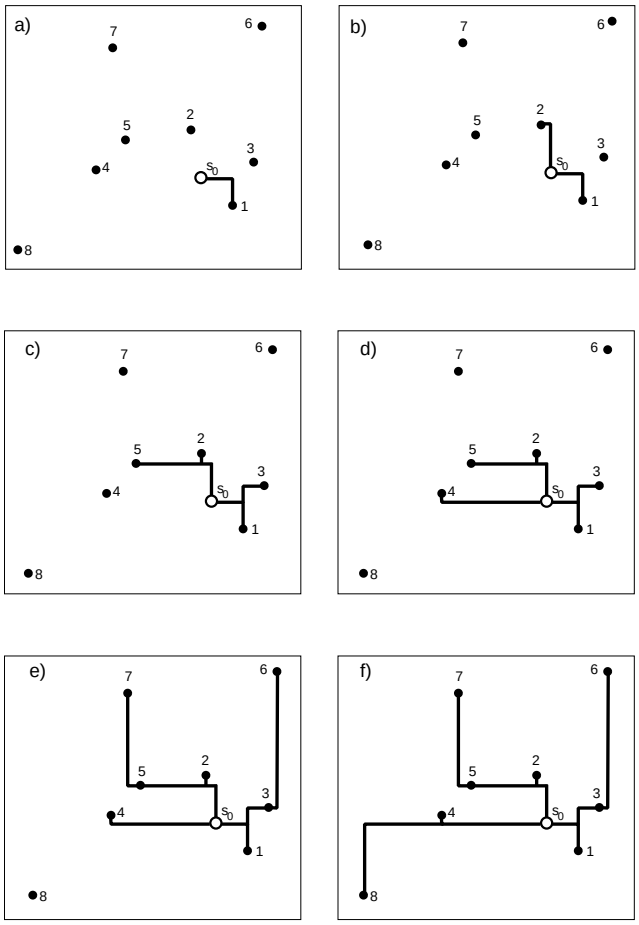


Figure 3.22 The SERT Steiner tree construction for an 8-sink signal net, using IC2 parameters. The source terminal is labeled 1, and sinks are numbered in order of distance from the source.

analysis using known module delays and estimated path delays. For example, Lin and Du [173] use the linear delay approximation so that their method updates the module placement to reduce the rectilinear distance between sources and critical sinks. Other path-dependent methodologies include those of Hauge et al. [119] and Teig et al. [236].

If a timing-critical path passes through a given net, the path-dependent approach can provide an explicit bound on the allowable delay at that net's critical sink. The net-dependent approach arguably provides only implicit routing constraints, but identification of critical sinks is possible after timing analysis, or *a priori* by finding paths that contain more module delays. By contrast, observe that the routing constructions discussed so far generally address only *net-specific* objectives: minimum cost (T_M or T_{I1S}), minimum radius (T_S), cost-radius tradeoff (T_{BRBC}, T_{KRY}, T_{PD1}, etc.), or maximum Elmore delay (T_{ERT} or T_{SERT}). Boese, Kahng and Robins [34] noted the resulting "placement-routing mismatch": generic, net-dependent methods fail to exploit the critical-path information that is available during iterative performance-driven layout. As a result, designers cannot realize the full potential of high-quality timing-driven module placements. The following *critical-sink routing problem* formulation is therefore of interest.

Critical-Sink Routing Tree (CSRT) Problem: Given a signal net $S = \{s_0, s_1, ..., s_n\} \subset \Re^2$ with source s_0 and sink criticalities $\alpha_i \geq 0$, $i = 1, \ldots, n$, construct a routing tree $T(S)$ such that $\sum_{i=1}^{n} \alpha_i \cdot t(s_i)$ is minimized.

This CSRT formulation seeks a routing tree $T(S)$ that minimizes a weighted sum of delays at critical sinks. Implicitly, we will evaluate sink delays according to Elmore delay, $t_{ED}(s_i)$. Also, our discussion will not consider any of the buffering or wiresizing optimizations that can also be used to reduce sink delays.

The CSRT formulation can be coerced into capturing traditional routing objectives (e.g., *average delay* to all sinks is minimized by using all $\alpha_i = c$ for some positive constant c. However, the case most relevant to current design practice identifies exactly one critical sink, denoted s_c. (Our discussion centers on this case, but the methodologies that we develop can be generalized to the case where a small number of critical sinks are specified.) Figure 3.23 shows how the presence of a critical sink can affect the optimal routing solution. The figure shows a signal net with critical sink s_c, along with three routing trees: (a) the optimal (minimum-cost) Steiner tree, (b) the optimal RSA (A-tree), and (c) the optimal-delay CSRT with respect to critical sink s_c. Part (d) of Figure 3.23(d) shows two distinct, optimal RSA's (A-trees) for a three-sink net. Some reflection on these examples and Equation 3.1 lead to the following observations:

- The minimum-cost Steiner tree solution (a) has large delay to the critical sink s_c due to its long source-sink path.

- In an optimal SPT or RSA (b), the requirement of a monotone path to every sink can result in large tree capacitance which again leads to large delay at s_c.
- The optimal-delay CSRT construction (c) reflects both the minimum-cost and the SPT solutions, and illustrates the dependence of the optimal routing topology on the choice of critical sink.
- Finally, Equation 3.1 implies that the number of Steiner points in the s_0-s_c path should be minimized, and the Steiner points "shifted" toward s_0 (i.e., branches off of the s_0-s_c path should occur as close to the source as possible). Figure 3.23(d) shows two trees which are both shortest-path trees and Steiner minimal trees, yet the tree at right has less signal delay at s_c.

That CSRT is intractable is not surprising, since choosing r_d large enough will make the C_0 term dominate Equation 3.1 and yield an SMT formulation. Boese et al. [33] have given a succinct proof (Figure 3.24) that for any choice of technology parameters $\bar{r}$, $\bar{c}$, r_d, and c_i, the SMT problem can be reduced to CSRT with a single critical sink, proving that CSRT is NP-hard. The generic version of CSRT (i.e., with the maximum sink delay criterion) is also NP-hard; this is proved by modifying the example of Figure 3.24 so that s_c is located far enough from s_0 that it has largest sink Elmore delay.

Geometric CSRT Heuristics

The examples of Figure 3.23 suggest that the optimal CSRT solutions tend to minimize total tree cost, subject to the path from s_0 to s_c being monotone (i.e., of minimum possible length). This is basically a simultaneous consideration of (geometric) radius and cost parameters, akin to the methods of the previous section but with modifications to account for the critical sink. Based on this intuition, Boese, Kahng and Robins [34] suggested the *CS-Steiner* approach (Figure 3.25).

CS-Steiner first constructs a heuristic minimum-cost Steiner tree over all terminals of S except the critical sink, then adds s_c into the tree so that $t_{ED}(s_c)$ is small. Boese et al. studied several variants which use I1S to construct the initial tree T_0 in Line 1. Then, Line 2 is accomplished as follows:

- **H0–** Introduce a single wire from s_c to s_0.

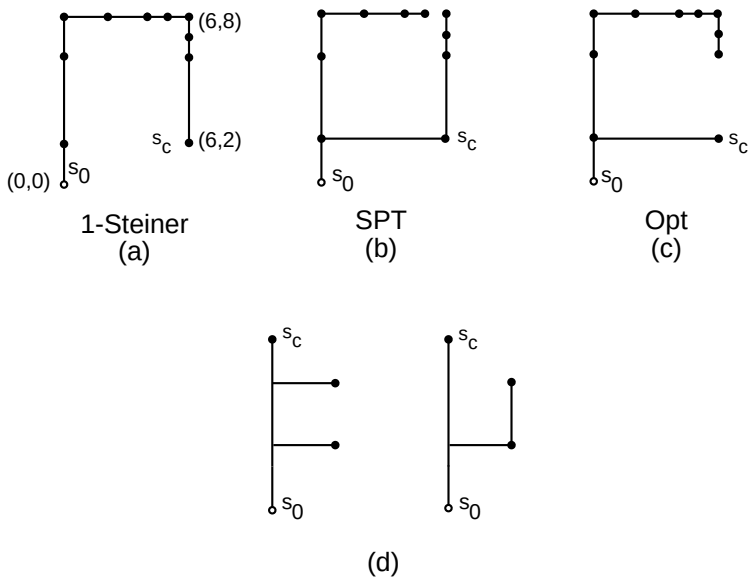


Figure 3.23 Parts (a)-(c): Steiner minimal tree (cost 2.0 cm, $t(s_c) = 3.34$ ns); optimal SPT or RSA (cost 2.5 cm, $t(s_c) = 2.26$ ns); and optimal-delay critical-sink routing tree (cost 2.2 cm, $t(s_c) = 1.67$ ns) for the same signal net. Coordinates shown are in mm, and the 1.2μ IC2 technology parameters in Table 3.1 were used with the Two-Pole simulator of Zhou et al. [256] and a 90% rise time delay criterion. Part (d): two distinct optimal SPT or RSA solutions for a signal net with three sinks.

- **H1–** Introduce the shortest possible wire that can join s_c to T_0, subject to the s_0-s_c path being monotone.
- **HBest–** Try all shortest connections from s_c to edges in T_0, as well as from s_c to s_0; perform timing analysis on each of these routing trees and return the tree with lowest delay at s_c.

The time complexities of these variants are dominated by the construction of the heuristic SMT T_0 (Line 1), or possibly by the timing analysis in the case of the HBest variant.

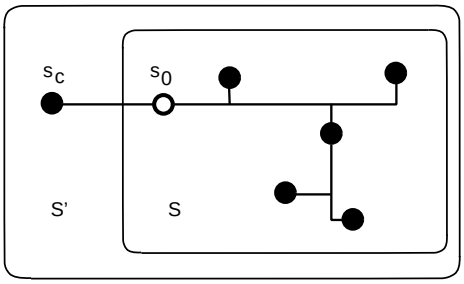


Figure 3.24 The CSRT problem with a single critical sink is NP-hard for any choice of technology parameters. A rectilinear SMT instance S is transformed into a CSRT instance $S' = S \cup \{s_c\}$ which has critical sink s_c directly left of $s_0 \in S$ which has smallest x-coordinate. The optimal CSRT solution consists of a rectilinear SMT over S, plus the edge (s_0, s_c).

Algorithm CS-Steiner
Input: signal net S; source $s_0 \in S$; identified critical sink $s_c \in S$ **Output:** heuristic CSRT solution T
1. Construct heuristic minimum-cost tree T_0 over $S - \{s_c\}$. 2. Form T by adding a *direct connection* from s_c to T_0, i.e., such that the s_0-s_c path in T is monotone.

Figure 3.25 The CS-Steiner heuristic.

An interesting complement to the CS-Steiner construction is *Global Slack Removal* (GSR), a postprocessing algorithm due to [34] which shifts edges to remove "U" and "V" configurations from the routing tree. Intuitively, such configurations correspond to the "slack" in non-monotone source-sink paths. The GSR algorithm has similarities to the method developed independently by Chen and Sarrafzadeh for wirelength minimization in single-layer routing [49].[18] However, the method of Chen and Sarrafzadeh is aimed at reducing tree cost, which is unnecessary here since CS-Steiner begins with a (near minimum-cost) I1S construction. The salient property of GSR is that it maximizes the mono-

[18]Similar "clean-up" techniques have been proposed in a number of contexts over the years. For example, the 1981 survey of Soukup [225] attributes such a method to Akers in a switchbox routing application.

tonicity of all source-sink paths and reduces Elmore delay to all sinks. This is accomplished without increasing tree cost.

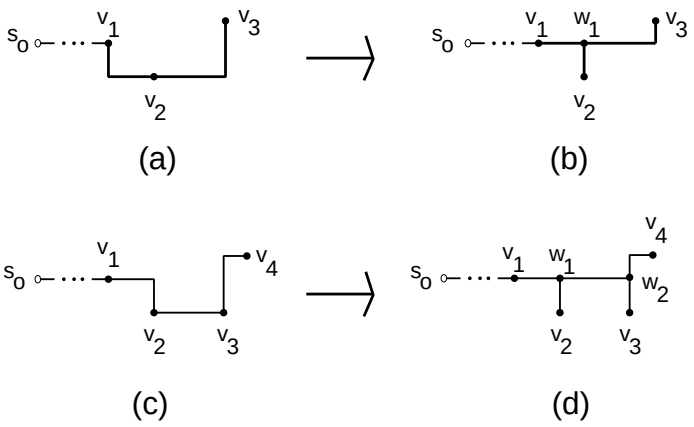


Figure 3.26 Removal of a V (top) or a U (bottom) by the GSR algorithm.

A "V" is defined to be a subpath of three consecutive nodes on a source-sink path in the routing tree, such that the cost of the subpath is greater than the distance between its endpoints (see subpath v_1-v_3 in Figure 3.26(a)). Similarly, a U is a subpath of four consecutive nodes on a source-sink path such that the cost of the subpath is greater than the distance between its endpoints (see subpath v_1-v_4 in Figure 3.26(c)). The nodes of a V or a U can be either Steiner nodes or terminals in S. A V is removed by introducing a Steiner node to eliminate the overlap between the two edges in the V, as in Figure 3.26(b). If a U (say, $v_1v_2v_3v_4$) does not contain any V's, then its middle edge (v_2, v_3) must be either completely horizontal or vertical; the U-removal corresponds to shifting this middle edge and adding up to two new Steiner nodes as shown in Figure 3.26(d).

Figure 3.27 more precisely describes the GSR algorithm. In the figure, a U (V) is said to be *located at* v when v is the node of the U (V) topologically furthest from s_0. The term $children(v)$ denotes the set of nodes that are children of v when the tree is rooted at s_0; $parent(v)$ denotes the parent of v in the rooted tree. The variable Q indicates a queue which ensures that each node in the tree is processed before its children (e.g., depth-first preorder). We make three observations:

```
Algorithm Global Slack Removal (GSR)
Input: Steiner tree T with source s_0
Output: Steiner tree T with all U's removed
1.    Remove all Steiner nodes of degree ≤ 2 from T;
2.    Q ← {s_0};
3.    While Q ≠ ∅
4.        v ← Dequeue(Q);
5.        For each node v' ∈ children(v) do
6.            Q ← Enqueue(v');
7.            If there is a V located at v'
8.                Remove_V(v')
9.            If there is a U located at v'
10.               Remove_U(v')
11.               Clean_Up(v')
12.   Remove all Steiner nodes of degree ≤ 2 from T;

Subroutine Clean_Up(node: v')
C1.   If there is a V located at parent(v')
C2.       Call Remove_V(parent(v'))
C3.   If there is a U located at v'
C4.       Call Remove_U(v')
C5.       Call Clean_Up(v')
C6.   Else
C7.       If there is a U located at parent(v')
C8.           Call Remove_U(parent(v'))
C9.           Call Clean_Up(parent(v'))
```

Figure 3.27 Pseudo-code for the Global Slack Removal (GSR) algorithm. Local variables include a queue Q and nodes v and v'. Remove_V(v) and Remove_U(v) are as illustrated in Figure 3.26.

- The "top-down" order enforced by Q is necessary because processing nodes after their children can introduce new U's that remain in the output tree. While distinct top-down orderings can produce distinct output trees, any output tree will satisfy the properties listed in Theorems 3.3.2 and 3.3.3 below.

- Removing a V or U at a node can possibly create new V or U configurations that must be removed along the path back to s_0 by the Clean_Up subroutine. Accounting for the possibility that Clean_Up processes any source-sink path up to a linear number of times, an $O(n^2)$ runtime upper

bound follows from the algorithm description. While examples exist for which I1S creates an input tree that forces $\Omega(n^2)$ runtime [34], in practice multiple calls to the subroutine Clean_Up occur for very few nodes, and GSR exhibits close to linear time complexity.

- Finally, examples are easily constructed which show that Steiner nodes of degree ≤ 2 can be introduced which inhibit removal of some U configurations. Since such low-degree Steiner nodes are superfluous, they are removed after every iteration.

Boese et al. [34] showed that the tree returned by GSR has no V's or U's, and that it dominates the input tree in terms of total tree cost, path length from the source to each sink, and Elmore delay at each sink.

Theorem 3.3.2 *Given any tree T as input, GSR returns a tree T' containing no V's and no U's.* □

Theorem 3.3.3 *Given any tree T as input, GSR will return a tree T' such that (i) $cost(T') \leq cost(T)$; (ii) for each $i > 0$, $dist_{T'}(s_0, s_i) \leq dist_T(s_0, s_i)$; and (iii) the Elmore delay $t_{ED}(s_i)$ at each s_i in T' is less than or equal to the Elmore delay $t_{ED}(s_i)$ in T.* □

Another geometric approach that addresses the CSRT problem (with a single critical sink) was proposed by Hong et al. [126]. This "Constructive Force-Directed " (CFD) algorithm is essentially another cost-radius tradeoff, but in a novel form: wires are grown from each sink to follow a weighted combination of the direction to s_0 and the direction to the closest growing ends of other wires. Intuitively, these directions correspond to attractive forces which direct the routing construction. The CFD approach seems reasonable (e.g., simple weighting schemes will yield a plausible RSA heuristic). [126] discusses specific weighting schemes and algorithmic issues (e.g., how a new growing end is determined once two growing ends meet), and describes the associated performance of the method. Since the CFD solution can have many jogs and detours, a clean-up phase similar to GSR may be useful.

CSRT Heuristics That Optimize Elmore Delay Directly

To address the CSRT formulation, Boese et al. modify their SERT method to yield the *"Steiner Elmore Routing Tree with identified Critical sink"*, or SERT-C, algorithm. SERT-C begins with a tree containing the single edge (s_0, s_c)

and then continues as in the SERT algorithm, minimizing $t_{ED}(s_c)$ rather than $max_{s_k \in V} t_{ED}(s_k)$. The algorithm is formally described in Figure 3.28.

SERT-C Algorithm
Input: signal net S with source $s_0 \in S$, critical sink s_c **Output:** critical-sink routing tree T over S
1. $T = (V, E) = (\{s_0, s_c\}, \{(s_0, s_c)\})$ 2. **While** $\|V\| < \|S\|$ **do** 3. Find $s_j \in S - V$ and $(v, v') \in E$ such that connecting s_j to the closest point x on (v, v') minimizes $t_{ED}(s_c)$ in the tree $(V \cup \{s_j, x\}, E \cup \{(v, x), (v', x), (x, s_j)\} - \{(v, v')\})$ 4. $V = V \cup \{s_j, x\}$ 5. $E = E \cup \{(v, x), (v', x), (x, s_j)\} - \{(v, v')\}$ 6. **Output** resulting Steiner tree $T = (V, E)$

Figure 3.28 The SERT-C Algorithm: direct incorporation of the Elmore delay formula into a greedy critical-sink routing tree construction. Note that it is possible for $x = v$ or $x = v'$ in Line 3.

In some sense, SERT-C takes the complete opposite approach from CS-Steiner. CS-Steiner begins with a heuristic minimum-cost Steiner tree over $S - \{s_c\}$, then perturbs it to include s_c with minimum delay $t(s_c)$. By contrast, SERT-C starts with the required s_0-s_c connection, then grows the routing tree around it while keeping $t_{ED}(s_c)$ as small as possible. As with the ERT and SERT algorithms, SERT-C's direct optimization of Elmore delay within the construction allows flexibility with respect to parameters of the technology and the input instance. Interestingly, the critical-sink problem formulation allows the path-dependent SERT-C algorithm to have nearly quadratic speedup over the generic net-dependent SERT algorithm.

Fact 3.3.4 *The SERT-C algorithm can be implemented to run in* $O(n^2 \log n)$ *time.*

Proof: The effect on $t_{ED}(s_c)$ of inserting a new edge (v, s_j) into T arises only in the C_k terms of Equation (3.1), and is an *additive constant* no matter when (v, s_j) is added into the tree. Initially, compute the best connection from each non-critical sink to the tree that contains only the edge (s_0, s_c). For each new sink added, at most three new edges will be inserted into the tree. It requires

constant time to calculate the effects of connections from a given sink outside T to these three new edges; all previously computed effects remain unchanged. Using a priority queue, for each $s_j \notin V$ the delay effects of connecting to these new edges can be recorded in $O(\log n)$ time, and the current minimum-cost connection for s_j can be retrieved in $O(\log n)$ time. Thus, each pass through the **while** loop of Figure 3.28 can be accomplished in $O(n \log n)$ time, giving an overall time complexity of $O(n^2 \log n)$. □

Figure 3.29 shows SERT-C constructions for various choices of critical sink, again using the IC2 technology parameters and the same 8-sink signal net in Figure 3.22. The tree constructed when s_c is node 2 or node 6 is also the I1S solution, and the tree constructed when s_c is node 7 is also the generic SERT result.

3.3.3 Experimental Results

CS-Steiner Trees

Table 3.5 compares the outputs of I1S and the CS-Steiner variants H0, H1 and HBest, with GSR post-processing applied, for random signal nets of 4 and 8 sinks, and technology parameters corresponding to those in Table 3.1. Results are given for tree cost (WL) and 50% rise time computed using the Two-Pole simulator of [256]; the HBest variant also uses calls to the Two-Pole simulator in its delay analysis of candidate connections.[19]

I1S, like BRBC, KRY, PD1, A-tree, etc., is net-dependent and returns the same tree for a given net no matter which sink happens to be critical. Thus, the I1S sink delays $t(s_i)$ are "generic". On the other hand, the CS-Steiner variants can return a different tree for each choice of critical sink. Thus, for each CS-Steiner variant we record the delay $t(s_i)$ in the *specific* tree that results from identification of s_i as the critical sink. Each entry in Table 3.5 represents an average taken over every sink node (i.e., all possible choices of critical sink) in 100 random signal nets. The results show that the simple strategy of connecting the critical sink directly to the source (i.e., H0) is quite successful. Variants

[19] That H0+GSR outperforms HBest+GSR is due to an inconsistency in the use of the Two-Pole simulator by HBest. To speed the evaluation of all candidate connections, HBest models each edge of a candidate tree with a single *RLC* segment during its calls to the Two-Pole simulator. Unfortunately, this does not capture the moments of the interconnect accurately (see the discussion in the Appendix), and HBest can choose a suboptimal connection. On the other hand, when evaluating the sink delays of the final output tree, each tree edge is modeled using a large number of *RLC* segments, resulting in greater simulation accuracy.

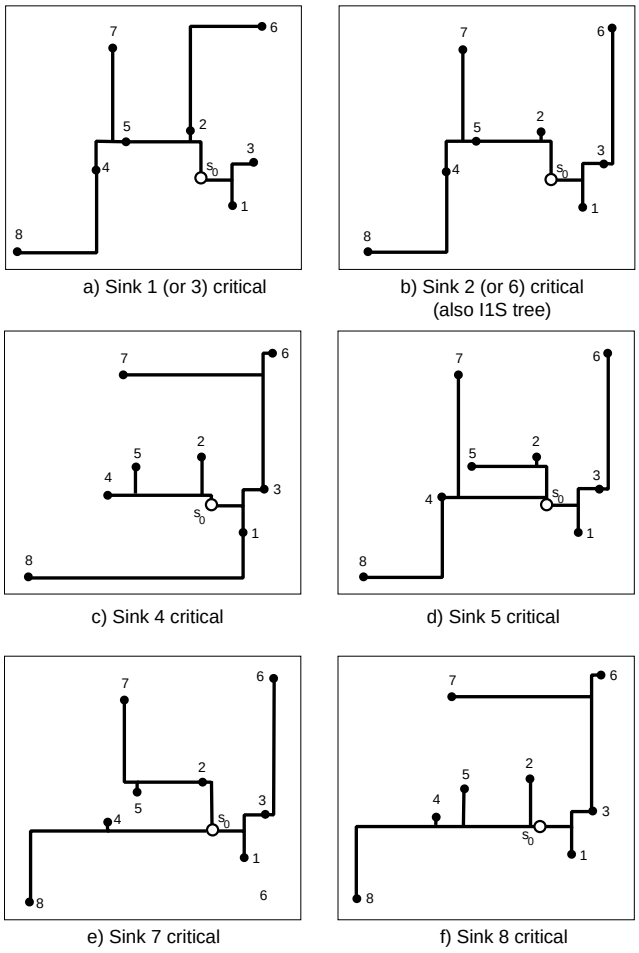


Figure 3.29 SERT-C tree constructions for an 8-sink net, showing variation of solution with choice of critical sink s_c.

H0 and HBest significantly reduce delay to the critical sink, particularly when MCM interconnect parameters are used.

$\|S\| = 5$		IC1	IC2	IC3	MCM
Critical Sink Delay	I1S	0.549 ns	0.331 ns	0.218 ns	2.31 ns
	I1S+GSR	.978	.970	.968	.952
	H0+GSR	.980	.876	.849	.550
	H1+GSR	.960	.934	.922	.857
	HBest+GSR	.929	.867	.844	.593
Ave WL	I1S	1.48 cm	1.48 cm	1.48 cm	14.8 cm
	H0+GSR	1.29	1.29	1.29	1.29
	H1+GSR	1.04	1.04	1.04	1.04
	HBest+GSR	1.07	1.10	1.11	1.22

$\|S\| = 9$		IC1	IC2	IC3	MCM
Critical Sink Delay	I1S	0.848 ns	0.520 ns	0.342 ns	4.09 ns
	I1S+GSR	.964	.954	.950	.927
	H0+GSR	.824	.700	.664	.333
	H1+GSR	.883	.827	.810	.665
	HBest+GSR	.817	.721	.693	.340
Ave WL	I1S	2.18 cm	2.18 cm	2.18 cm	21.8 cm
	H0+GSR	1.22	1.22	1.22	1.22
	H1+GSR	1.06	1.06	1.06	1.06
	HBest+GSR	1.11	1.12	1.12	1.21

Table 3.5 Comparison of CS-Steiner variants against I1S. Each critical sink delay value corresponds to an average over each possible critical sink in 100 random signal nets. I1S results are reported in the physical units (nanoseconds or centimeters) while other results are reported relative to I1S. I1S+GSR produced essentially the same average WL values as I1S.

Elmore Routing Trees

Tables 3.6 and 3.7 compare outputs of the ERT,SERT and SERT-C heuristics against those of the MST, PD1 and I1S constructions for 5- and 9-terminal nets. The algorithms were executed on the same sets of random inputs used in the CS-Steiner experiments, and the same delay simulation methodology was used. In general, these results show the Elmore-based "Elmore routing tree" approach of Boese, Kahng and Robins to be quite effective.

$\|S\| = 5$		IC1	IC2	IC3	MCM
Critical Sink Delay	MST	0.645 ns	0.395 ns	0.262 ns	2.82 ns
	PD1	.904	.863	.885	.777
	ERT	.879	.804	.782	.472
	I1S	0.549 ns	0.331 ns	0.218 ns	2.31 ns
	SERT	.967	.921	.908	.584
	SERT-C	.947	.870	.839	.567
Maximum Sink Delay	MST	0.758 ns	0.485 ns	0.326 ns	3.86 ns
	PD1	.876	.835	.822	.759
	ERT	.855	.786	.764	.544
	I1S	0.627 ns	0.393 ns	0.262 ns	3.06 ns
	SERT	.955	.919	.908	.699
	SERT-C	.970	.962	.954	.859
Average WL	MST	1.64 cm	1.64 cm	1.64 cm	16.4 cm
	PD1	1.16	1.16	1.16	1.04
	ERT	1.10	1.18	1.19	1.61
	I1S	1.48 cm	1.48 cm	1.48 cm	14.8 cm
	SERT	1.06	1.11	1.13	1.66
	SERT-C	1.06	1.15	1.16	1.28

Table 3.6 Comparison of ERT, SERT and SERT-C variants against the MST, PD1 and I1S constructions for 5-terminal nets. Each critical-sink entry corresponds to an average over delay computations for all possible choices of critical sink in each of 100 random signal nets. Spanning ERT constructions are compared with MST and PD1; Steiner SERT and SERT-C constructions are compared with I1S. The MST and I1S results are reported in the physical units (nanoseconds or centimeters); other results are reported relative to these values.

$\|S\| = 9$		IC1	IC2	IC3	MCM
Critical Sink Delay	MST	0.984 ns	0.609 ns	0.403 ns	4.80 ns
	PD1	.837	.770	.749	.608
	ERT	.837	.741	.702	.329
	I1S	0.848 ns	0.520 ns	0.342 ns	4.09 ns
	SERT	.884	.806	.781	.384
	SERT-C	.847	.735	.693	.340
Maximum Sink Delay	MST	1.213 ns	0.792 ns	0.533 ns	7.05 ns
	PD1	.805	.747	.730	.632
	ERT	.790	.699	.668	.399
	I1S	1.028 ns	0.664 ns	0.444 ns	5.92 ns
	SERT	.853	.780	.759	.481
	SERT-C	.914	.892	.892	.846
Average WL	MST	2.43 cm	2.43 cm	24.3 cm	24.3 cm
	PD1	1.09	1.09	1.09	1.07
	ERT	1.15	1.25	1.27	2.15
	I1S	2.18 cm	2.18 cm	2.18 cm	21.8 cm
	SERT	1.09	1.18	1.22	2.27
	SERT-C	1.06	1.11	1.14	1.22

Table 3.7 Comparison of ERT, SERT and SERT-C variants against the MST, PD1 and I1S constructions for 9-terminal nets.

If no critical sink is specified during the routing construction, a net-dependent spanning implementation of ERT will still reduce the delay to whatever sink eventually turns out to be critical. This is verified by measuring delays at randomly chosen critical sinks: for 8-sink nets, this delay reduction is 16%, 26%, and 30% in the three IC technologies and is 67% in the MCM technology. ERT also improves upon PD1 by 0% (IC1), 4% (IC2), 6% (IC3), and 46% (MCM), even though PD1 is allowed to output the best tree over all parameter values c for each instance. For Steiner routing, SERT is also a good "generic" construction: with 8-sink nets, improvements in delay to a random critical sink are, e.g., 19% for IC2 and 62% for MCM, when compared to I1S. In terms of the net-dependent maximum sink delay criterion, which is the natural measure for the ERT and SERT constructions, improvements over MST and I1S are similar (the percentages are somewhat greater for IC technologies, and somewhat smaller for MCM).

Due to limitations of the present modeling and simulation methodology, delay reductions may not attain these magnitudes in practice. However, the conclusions as to relative delays of the various constructions are almost certainly valid (cf. the discussion of accuracy and fidelity of delay estimators in the Appendix). It should also be noted that the ERT and SERT constructions can be somewhat star-like, especially for MCM parameters, due to the maximum sink delay criterion. Since the resulting tree costs will be significantly higher than those of I1S, practitioners may smoothly recapture wirelength (e.g., when the net is not on any critical path) by simulating a larger r_d value in the construction.

When a critical sink s_c is known, further reductions in delay can be achieved. For example, the SERT-C algorithm improves over SERT by additional reductions in critical-sink delay of 5%, 7% and 6% for the three IC technologies, and 8% for MCM. More significant advantages from knowing s_c are gained in terms of tree cost. Particularly for MCM parameters, SERT-C trees have much less cost than SERT trees, even while improving the critical-sink delay. An interesting side note is that the SERT-C maximum sink delay also decreases relative to I1S. It is thus likely that the overall delay skew in the routing tree will be reduced even when the user addresses the path-dependent critical-sink criterion, as opposed to the net-dependent maximum delay criterion. Finally, SERT-C produces very similar delays and costs when compared to the CS-Steiner variants HBest and H0. However, SERT-C is more practical since it runs in $O(n^2 \log n)$ time and does not require any simulator calls as does HBest (the underlying I1S call in CS-Steiner itself requires $O(n^3)$ time).

3.3.4 Optimal-Delay Routing Trees

Given the success of the geometric performance-driven routing heuristics, and the further success of the Elmore-based methods, it is natural to ask whether substantial further advances are possible. In other words, we seek to define the *achievable envelope* of routing tree designs with respect to performance. The study of *delay-optimal trees* can provide a bound on possible gains from future work in performance-driven routing. This, in turn, can provide impetus toward research in other performance-driven layout techniques, such as driver- and wire-sizing. The characterization of optimal-delay trees also yields improved layout and performance estimators for placement, floorplanning and high-level synthesis.

Spanning Trees and BBORT

For the maximum sink delay objective, Boese et al. [30, 31] have used branch-and-bound to find a spanning *optimal routing tree* (ORT) solution according to Elmore delay. Starting with a trivial tree containing only s_0, the algorithm incrementally adds one edge at a time to the growing tree while updating the the maximum sink delay. If this delay value exceeds the maximum sink delay in any *complete* candidate tree seen so far, the search backtracks to use a different edge at the previous step. A recursive implementation of this *Branch-and-Bound Optimal Routing Tree* (BBORT) search is shown in Figure 3.30. BBORT adds sinks in a breadth-first manner, with the children of any parent added in increasing order of their indices. It can be seen that there is a unique sink ordering corresponding to each tree topology, and that each topology will be visited at most once. In Figure 3.30, Lines 7-9 embody the branch-and-bound search. If the current tree T' has delay greater than or equal to t_{min} (the current best-known delay for a complete tree), then procedure Add_Edges terminates and the algorithm backtracks. Otherwise, if T' is a complete spanning tree, then t_{min} is set to the delay of T', or if T' is a partial tree, then Add_Edges recursively adds more edges to T'.

Algorithm BBORT

Input: signal net S with source $s_0 \in S$
Output: optimal-delay tree T_{opt} over S

1. $T = (V, E) = (\{s_0\}, \emptyset)$
2. $t_{\min} = \infty$
3. Call Add_Edges(T)
4. Output T_{opt}

Procedure Add_Edges(Tree: $T = (V, E)$)
5. **While** there exist $s_i \in V$ and $s_j \notin V$ such that
   $T' = (V \cup \{s_i\}, E \cup \{(s_i, s_j)\})$ is a new tree topology **Do**
6. Compute maximum sink delay $t(T')$
7. **If** $t(T') < t_{\min}$ **Then**
8. **If** $|T'| = |S|$ **Then** $T_{opt} = T'$; $t_{\min} = t(T')$
9. **Else** Call Add_Edges(T')

Figure 3.30 The Branch-and-Bound Optimal Routing Tree (BBORT) algorithm (recursive implementation).

Table 3.8 compares the maximum sink Elmore delays and the tree costs of the ORT (i.e., found by BBORT), ERT, SPT and MST solutions. The SPT is

a rectilinear spanning arborescence, i.e., the minimum-cost spanning tree in which all source-sink paths are monotone. Each entry in the table represents an average over 200 randomly generated signal nets, with the same technology parameters and delay simulation methodology as above. Delay for each tree is normalized to the ORT delay of the same net, and tree cost is normalized to the MST cost of the net.

$\|S\| = 5$	IC1		IC2		IC3		MCM	
	delay	cost	delay	cost	delay	cost	delay	cost
ORT	1.0	1.103	1.0	1.140	1.0	1.146	1.0	1.432
ERT	1.007	1.104	1.010	1.159	1.011	1.172	1.009	1.585
SPT	1.085	1.290	1.058	1.290	1.054	1.290	1.089	1.290
MST	1.169	1.0	1.272	1.0	1.311	1.0	1.894	1.0

$\|S\| = 7$	IC1		IC2		IC3		MCM	
	delay	cost	delay	cost	delay	cost	delay	cost
ORT	1.0	1.133	1.0	1.175	1.0	1.190	1.0	1.547
ERT	1.017	1.142	1.022	1.215	1.027	1.252	1.024	1.892
SPT	1.130	1.395	1.096	1.395	1.091	1.395	1.161	1.395
MST	1.282	1.0	1.451	1.0	1.499	1.0	2.457	1.0

Table 3.8 Near-optimality of Elmore delays and tree costs of various constructions, using IC1, IC2, IC3 and MCM parameters. Tree cost is normalized to MST cost, and delay is normalized to ORT delay.

For 7-terminal nets in the IC1 technology, the ERT has maximum sink Elmore delay averaging only 1.7% greater than optimal; by contrast, the MST has maximum sink Elmore delay averaging 28.2% greater than optimal. For 5-terminal nets, ERT delays average 0.7% above optimal, while MST delays average 16.9% above optimal. The confidence in these estimates of ERT suboptimality is very high, e.g., the 1.7% suboptimality obtained for 7-terminal nets and the IC1 technology has 95% confidence interval (i.e., within twice the standard error of the average) between 1.3% and 2.1%. Even with the worst results in the table, for IC3 parameters and 7-terminal nets, ERT remains within 2.7% of optimal Elmore delay, with 95% confidence interval for this estimate between 2.2% and 3.2%.

Toward Elmore Delay-Optimal Steiner Trees

To completely delimit the "performance envelope" for routing trees and to assess the near-optimality of the SERT and SERT-C constructions, it is necessary to compute Elmore delay-optimal Steiner routing trees. Both the Steiner optimal routing tree (SORT), which minimizes $max_{s_i} t_{ED}(s_i)$, and the Steiner optimal routing tree with identified critical sink (SORT-C), which minimizes $t_{ED}(s_c)$, are of interest. At first glance, computing either of these trees seems difficult: since there are potentially an infinite number of candidate Steiner node locations, even branch-and-bound may be infeasible.

Two theoretical results obtained by Boese et al. [32, 33] enable exact determination of the performance envelope for routing trees. The first result limits the Steiner nodes of Elmore-optimal trees to the same "Hanan grid" that contains the Steiner candidate set of an SMT instance. This implies that a finite algorithm exists which determines optimal CSRT solutions for any positive combination of Elmore delays to critical sinks. The second result gives a "peeling decomposition" of any Elmore-optimal Steiner tree into a sequence of subtrees, each of which adds a sink by a *closest connection* to some edge of the previous subtree. Together, these results afford a branch-and-bound search method that extends BBORT to optimal critical-sink Steiner topologies. A brief review of the results in [32, 33] is as follows (again, all delays $t(s_i)$ are assumed to be Elmore delay).

Let T^* be a CSRT solution that minimizes $f = \sum_{i=1}^{n} \alpha_i \cdot t(s_i)$. Without loss of generality, assume that all $\alpha_i > 0$ and that T^* contains no degree-2 Steiner nodes. A tree will be considered to be a collection of nodes (possibly terminals of S) and edges, so that $v \in T$ for node v and $e \in T$ for edge e are both meaningful. A *straight edge* is an edge that is completely vertical or horizontal; other edges are *L-shaped.*

The *closest connection* between three nodes is the location of the single Steiner node in a minimum-cost Steiner tree over the three nodes. This unique location has coordinates given by the medians of the x- and the y- coordinates of the three nodes. The *closest connection* between a node v and an edge e is the closest connection between v and the two endpoints of e. If a Steiner tree T over S is rooted at s_0, define $T \backslash v$ to be the tree induced by removing node v and all its descendants from T, and then removing all remaining degree-2 Steiner nodes. Node $v \in T$ is *connected to* an edge $e \in T \backslash v$ if its parent node in T is located on edge e. If $parent(v)$ is located at the closest connection between

v and an edge $e \in T \backslash v$ to which v is connected, then v is said to make a *closest connection* to e in T.

An enabling observation is that in the optimal tree T^*, Elmore delay at every sink is a concave function of the distance x that separates the closest connection between v and e from their actual connection point. Any linear combination of concave functions is itself concave; thus, any positive combination of sink Elmore delays is also concave in x. Using the fact that a concave function defined over a convex domain takes on its minimum value at an extreme point of the domain, Boese et al. showed that T^* is composed of closest connections:[20]

Lemma 3.3.5 *Suppose node $v \in T^*$, $v \neq s_0$, is connected to edge $e \in T^* \backslash v$. Then either $parent(v) = s_0$ or v makes a closest connection to e in T^*.* □

For any routing tree T rooted at s_0 and for any $v \in T$, let T_v to be the subtree of T rooted at v. A *segment* is a contiguous set of straight edges in T which are either all horizontal or all vertical; a *maximal segment* (MS) is a segment that is not properly contained in any other segment. Let M be an MS in T. The node in M closest to s_0 on a source-sink path containing M is called the *entry point* of M. A segment containing all points in M to one side of M's entry point is called a *branch* (sometimes a branch will include the entire MS). A branch b is a *branch off of* MS M' if M' and b are incident at a single node which is not the entry point to M'. L-shaped edges are also defined as branches.

An MS M divides the plane into two half-planes: the half-plane containing the edge between M's entry point and its parent is the *near side* of M, and the other half-plane is the *far side* of M. Branches off of M that are located on its near (far) side are called *near (far) branches*. In addition, a *sink* located on M is defined to be a *far branch* off of M if it is not the entry point to a larger far branch. For any segment S, $Near(S)$ (resp. $Far(S)$) denotes the set of near (resp. far) branches off of the maximal segment containing S. Figure 3.31 gives an example of an MS M with endpoints p_1 and p_2, entry point p_0, and four branches, including near branch b_1, far branches b_2 and b_4, and another far branch consisting only of sink s_3.

Two more lemmas from [32] respectively follow from (i) basic properties of the Elmore delay formula, and an edge-shifting argument applied to the balance

[20] The technique of exploiting concavity is much more powerful than the "segment-shifting" that Hanan used to prove his original result for minimum-cost trees. Indeed, the edge shifts used by Hanan can be suboptimal in terms of Elmore delay. Applications of this technique to more sophisticated delay estimates may be promising.

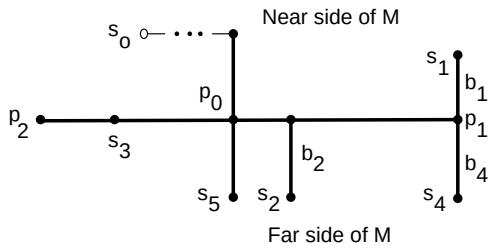


Figure 3.31 Example of a maximal segment M with entry point p_0, one near branch b_1, and three far branches, including b_2. Note that by definition, s_3 forms a far branch with no edges. Edge (p_0, s_6) is *not* a far branch off of M because p_0 is not an entry point to the MS containing (p_0, s_6).

between $|Far(M)|$ and $|Near(M)|$, and (ii) Elmore delay at all sinks being a concave function of the position of any maximal segment.

Lemma 3.3.6 *Let M be a maximal segment in T^* that does not contain s_0. Then $|Far(M)| > |Near(M)|$.* □

Lemma 3.3.7 *Any maximal segment in T^* must contain either a sink or the source.* □

Corollary: Any Steiner node in T^* is located on the Hanan grid. □

Hanan's original theorem may be viewed as being equivalent to a special case of this Corollary with $r_d \to \infty$. (Hanan proved that all edges of some rectilinear SMT lie on the Hanan grid; this Corollary is stated with respect to node locations.)

Finally, Boese et al. show that T^* can be constructed by starting with the trivial tree $T_0 = (\{s_0\}, \emptyset)$ and successively adding sinks according to some ordering $s_1, s_2, \ldots, s_n$ to create trees $T_1, T_2, \ldots, T_n = T^*$, with each s_i making a closest connection to some edge in T_{i-1}. This result follows from the existence of a reverse, "peeling" decomposition of T^*.

Theorem 3.3.8 *There is a sequence of trees* $T_0 = (\{s_0\}, \emptyset), T_1, T_2, \ldots, T_n = T^*$ *such that for each* i, $1 \leq i \leq n$, *(i) there is a sink* $s_i \in T_i$ *such that* $T_{i-1} = T_i \backslash s_i$, *and (ii) either* s_i *is connected to* s_0, *or* s_i *makes a closest connection in* T_i *to some edge in* T_{i-1}.

Note that this peeling decomposition extends to the classic Steiner minimal tree problem when $r_d \to \infty$. Similar to the first decomposition theorem in the Steiner literature [135], this second decomposition provides both a characterization of, and an effective means of generating, optimal Steiner trees.

Steiner Trees and BB-SORT-C

Based on the the above results, a simple modification to algorithm (BBORT) BBORT can find an optimal Steiner routing tree for any linear combination of Elmore delays at critical sinks. Rather than considering connections from each sink s_j outside the current tree to each sink s_i inside the tree as in BBORT, the *Branch-and-Bound* method for *Steiner Optimal Routing Trees with Critical Sinks* (BB-SORT-C) considers connections from s_j to each edge created when s_i was added to the tree. In other words, each node s_i that is already contained in T is replaced as a possible connection point by each of the edges created when s_i was added to the tree earlier. The branch-and-bound pruning is used to reduce the complexity of the search and avoid redundant topologies. Since BB-SORT-C searches over all possible ways to construct a Steiner tree sequentially with each sink added by a closest connection to an edge in the current tree, the algorithm returns T^*. Interestingly, Boese et al. [33] show that for the maximum sink delay objective (the CSRT formulation with all α_i = a constant and an L_∞ sum), the Hanan grid result does not hold. In other words, there are examples for which no delay-optimal tree lies on the Hanan grid. This suggests that the technique used to generalize Hanan's theorem for a concave delay function was in some sense sharp: while the positive sum of concave functions is always concave, the maximum of concave functions is not necessarily a concave function.

Table 3.9 compares the critical sink Elmore delays of SERT-C and SORT-C (i.e., found by BB-SORT-C) routing trees. Each entry represents an average over 200 random nets, with the same technology parameters and delay simulation methodology as above. Delay for each tree is normalized to the SORT-C delay, and tree cost is normalized to the I1S cost. For 6-sink nets and the IC1 technology, SERT-C achieves Elmore delay that is on average within 11.1% of optimal;

results for IC2, IC3, and MCM parameters are very similar.[21] Although the SERT-C algorithm is not as close to optimal as the ERT algorithm for the types of delay measures studied here, these results provide strong guidance for future efforts in performance-driven routing. Even if future work improves the near-optimality of critical sink routing constructions, Table 3.9 shows that any future improvement in Elmore delay will be at most from 8% to 12% for signal nets with up to 6 sinks.

$\|S\| = 5$	IC1		IC2		IC3		MCM	
	delay	cost	delay	cost	delay	cost	delay	cost
SORT-C	1.0	1.111	1.0	1.161	1.0	1.175	1.0	1.296
SERT-C	1.042	1.046	1.049	1.120	1.046	1.140	1.000	1.296
1-Steiner	1.117	1.0	1.228	1.0	1.275	1.0	1.455	1.0
$\|S\| = 7$	IC1		IC2		IC3		MCM	
	delay	cost	delay	cost	delay	cost	delay	cost
SORT-C	1.0	1.112	1.0	1.158	1.0	1.165	1.0	1.262
SERT-C	1.083	1.047	1.114	1.106	1.112	1.112	1.001	1.256
1-Steiner	1.200	1.0	1.362	1.0	1.429	1.0	1.634	1.0

Table 3.9 Near-optimality of Elmore delays and tree costs of various Steiner tree constructions, using IC1, IC2, IC3 and MCM parameters. Tree cost is normalized to I1S cost, and delay is normalized to BB-SORT-C delay. Standard errors for SERT-C delays are shown in parentheses.

3.3.5 Remarks

This section has introduced both the "direct" optimization of Elmore delay, and the critical-sink formulation, and described several heuristics. The greedy "Elmore routing tree" variants – ERT, SERT and SERT-C – give promising results in terms of both generic (max sink delay) and critical-sink performance-driven routing. Since the constructions are fairly close to optimal, alternate methods of improving delay beyond the routing topology design would seem worth pursuing.[22]

[21] Average running times for nets with $|S| = 5$ (in CPU seconds on a Sun 4) are 0.006 (BB-SORT-C), 0.0004 (SERT-C), and 0.0025 (1-Steiner). Average running times for nets with $|S| = 7$ are 0.46 (BB-SORT-C), 0.0008 (SERT-C), and 0.0074 (1-Steiner).

[22] Combining "fidelity" studies such as those in the Appendix with the above studies of Elmore delay suboptimality, one can obtain upper bound estimates of the ERT, SERT and

Which of these routing heuristics is most useful will depend on the application. CS-Steiner variants H0 and HBest yield the smallest delay values for a single critical sink, but tend to have high time complexity. The SERT-C heuristic has $\Theta(n^2 \log n)$ time complexity, and extends to the case of nets with multiple critical sinks (apply SERT with max delay objective to the critical sinks, then apply SERT-C with a weighted average delay objective to connect the remaining sinks). The "generic" ERT and SERT heuristics can also be applied before critical path information becomes available (reduction of the time complexities of ERT and SERT remains an interesting open problem). For nets on non-critical paths, minimizing area will take precedence over minimizing delay, hence traditional minimum-cost Steiner tree heuristics such as I1S, or simulation of a higher r_d value in any ERT variant, will be preferable. Both the CS-Steiner and ERT approaches extend to incorporate wiresizing and address general-cell layout with arbitrary routing region costs. Finally, Vittal and Marek-Sadowska [244] have recently given input instances for which ERT and SERT return highly suboptimal routing trees. The alphabetic tree based approach in [244] uses more "global" criteria than the greedy ERT construction, and can thus escape such pathological instances.

3.4 NEW DIRECTIONS

While existing routing algorithms center largely on topology design, the scaling of VLSI technology has shifted layout optimizations to account for interconnect- and device-level phenomena. We conclude this chapter by sketching two recent directions in the delay-driven design of VLSI interconnects. First, we discuss modification of the *wire geometry*, as opposed to the wire topology, for improved signal propagation. A second idea – the use of *non-tree* routing topologies to reduce signal delay – is also quite foreign vis-a-vis our development so far. Experimental evidence suggests that these approaches can substantially improve signal delay as well as skew, reliability, and other attributes of the interconnect design. In the future, these forms of interconnect optimization will take on greater importance for performance-driven routing applications.

SERT-C delay suboptimality with respect to SPICE-computed delay. For spanning trees over 5-terminal nets, Boese et al. [33] estimate that the optimal tree according to Elmore delay will be between 3% and 10% above SPICE-optimal, depending on the technology. Since the SERT-C heuristic is between 0% and 5% above optimal in terms of Elmore delay for 5-terminal nets, the SPICE delay suboptimality of SERT-C heuristic can be estimated to range from 3% for MCM to at most 12% for 0.5 μm and at most 15% for 1.2μm and 2.0 μm CMOS IC technologies.

3.4.1 Wiresizing

The previous discussion has assumed that VLSI interconnections have uniform width and thickness, in that only the *length* of an interconnection is controlled by the designer. However, optimization of wire geometries to improve signal propagation is an established precept, e.g., in microwave and analog circuit design. A large body of parameter extraction, process simulation, and three-dimensional electromagnetic simulation techniques all address the three-dimensional – as opposed to one-dimensional – nature of integrated circuit wiring. As wire width and spacing continue to decrease, and as device switching speeds continue to increase, previous "second-order" phenomena (line coupling, deposition profiles, inductive effects, etc.) become more significant. The concept of *wiresizing* is motivated by the basic tradeoff between capacitance and resistance in the wire geometry.

For a given interconnect technology, let us change the definition of "unit resistance" so that $\bar{r}$ now denotes the resistance of a unit-length, *unit-width* wire segment; we define unit capacitance $\bar{c}$ similarly. Then, a segment of width w units will have resistance per unit length of $\frac{\bar{r}}{w}$ and capacitance per unit length of $\bar{c} \cdot w$. Given an RC tree with variable wire widths, we can substitute the appropriate lumped values for each segment in the distributed representation of the tree. The evaluation of signal delays, e.g., according to the Elmore formula, remains the same as in our previous discussion.

Fisher and Kung [94], Zhu et al. [261], and Pullela et al. [198] have used wiresizing to optimize the design of clock distribution networks; see Friedman [100] for an overview of related techniques. Dutta and Marek-Sadowska [80] have previously used wiresizing in the design of power and ground networks, where upper bounds on current densities must be satisfied to achieve reliability. For performance-driven routing of arbitrary signal nets, Cong et al. [64, 65] and Sapetnekar [209] have given the main early results, corresponding to the case where the tree layout is prescribed and only the segment widths can be varied.

In [64, 65], it is assumed that only a small number r of widths $W_1, W_2, \ldots, W_r$ are available to implement the wire segments in the interconnect tree. The resulting *discrete wiresizing problem* formulation uses the same weighted sum of critical-sink delays objective [34] that we have discussed above:

The Discrete Wiresizing Problem: Given the set of edges E of a routing tree T, n sink criticalities $\alpha_i > 0$, and a set of r available wire widths $W = \{W_1, W_2, \cdots, W_r\}$, find a wire width assignment $f : E \rightarrow W$ to minimize the weighted sum of critical sink delays $\sum_{i=1} \alpha_i \cdot t(s_i)$.

Following [65], let $w_j = f(e_j)$ denote the width assignment of edge e_j. Also, let f^* indicate the optimal wire width assignment, with w_j^* being $f^*(e_j)$. We abuse notation and allow T to contain more "edges" than sinks (e.g., if T is embedded in the grid graph); this allows a given connection to contain multiple segments (i.e., edges) with possibly different widths. Let $Anc(e_j)$ denote the set of all ancestor edges on the unique path from the source to edge e_j, excluding e_j itself; similarly, let $Des(e_j)$ denote the set of descendant edges, $\{e_k \mid e_j \in Anc(e_k)\}$.

To make the discrete wiresizing problem tractable, it is assumed that sink delays are given either by the Elmore delay upper bound of Rubenstein et al. or by Elmore delay itself. Cong et al. state that the following two properties hold for the Elmore delay upper bound [65] and for Elmore delay [64].

(Monotonicity)[23] For any routing tree T, the optimal wire width assignment f^* satisfies $w_i^* \geq w_j^*$ whenever e_i is an ancestor of e_j.

(Separability) The optimal width w_i^* of edge e_i depends only on the width assignments $\{w_j \mid e_j \in Anc(e_j) \cup Des(e_j)\}$ of e_i's ancestors and descendants.

According to [65], once e_i and the edges in $Ans(e_i)$ have been assigned widths, the optimal width assignments for each "single-stem subtree" – i.e., a maximal subtree within $Des(e_i)$ which has exactly once edge adjacent to e_i – can be independently determined. For example, in Figure 3.32 (reproduced from [209]), once w_1^* has been fixed, w_2 and w_3 can be optimized independently.

Together, these two properties imply that the disjoint maximal subtrees below any given edge can be optimized independently (Separability), and only monotone root-leaf width assignments need be considered (Monotonicity). Thus, Cong et al. propose an $O(|E|^{r-1})$ recursive algorithm to solve the discrete wiresizing problem, essentially by enumerating all monotone wire width assignments on every source-sink path in T.

More recently, Sapetnekar [209] has suggested minimization of the *maximum* sink Elmore delay, rather than the weighted sum of sink Elmore delays. For this

[23] Cf. the discussion of synthesis of clock trees and other interconnects in [18].

objective, the separability property does not hold, and the solution methods of Cong et al. do not apply. In Figure 3.32, $t_{ED}(s_2)$ depends on the widths w_1 and w_2 which define the resistive and capacitive elements of the s_0-s_2 "main path". Separability fails because $t_{ED}(s_2)$ also depends on the off-path capacitive load defined by w_3: minimizing $t_{ED}(s_2)$ is achieved by using minimum width for e_3. The symmetric situation holds for $t_{ED}(s_2)$. Since minimum width of, say, e_2 implies greater resistivity on the s_0-s_2 path, it is not possible to optimize the maximum sink delay by optimizing the two sink delays independently. Sapetnekar also treats a *continuous* version of the wiresizing problem, i.e., the set of available wire widths forms an interval $W = [w_{min}, w_{max}]$, and proposes an efficient heuristic based on sensitivity analysis.

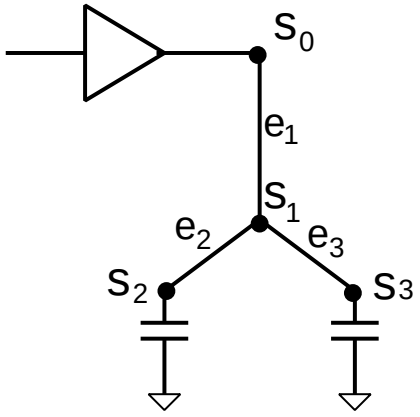


Figure 3.32 Counterexample to the separability property for the maximum sink Elmore delay objective, reproduced from [209].

For the discrete wiresizing problem, [64, 65] proposed the following $O(|E|^2 \cdot r)$ greedy heuristic, which iteratively changes a single wire width to improve the delay objective while keeping all other wire widths fixed. The algorithm, which we call Static Greedy Wiresizing (SGW), terminates when no single wire width change can improve the delay objective. Figure 3.33 gives an equivalent description of this strategy, with an arbitrary delay calculation being allowed in evaluating the current wire width assignment. The template shows the "increasing version" of the algorithm, i.e., we start with minimum wire widths and increase the width of individual segments while always reducing the delay objective. Symmetrically, it is also possible to run a "decreasing version" of

SGW which greedily decreases wire widths starting from an initial maximum-width wiresizing.

Algorithm Static Greedy Wiresizing (SGW) (increasing version):
Input: $T = (V, E)$ rooted at source s_0, delay objective $t(T)$, finite set $W = \{W_1 < W_2 < \ldots < W_r\}$ of discrete edge widths, wire width assignments $f(e_i) \equiv W_1$ for all $e_i \in E$ **Output:** Wire width assignment $f : E \rightarrow W$ for tree T

For each node $s_i \in V$ such that $e = (s_0, s_i) \in E$ **Do**
Call SGW on the subtree rooted at s_i
Repeat
$\text{delay}_{old} = t(T)$
Increase $f(e)$ from W_k to W_{k+1}
Until $\text{delay}_{old} < t(T)$ or $f(e) = W_{r+1}$
Decrease $f(e)$ from W_k to W_{k-1}

Figure 3.33 The Static Greedy Wiresizing (SGW) algorithm (increasing version). For convenience, we assume the existence of a width W_{r+1} that is greater than the maximum allowed width W_r.

Given two wire width assignments f_1 and f_2 for some tree T, we say that f_1 *dominates* f_2 if $f_1(e_i) \geq f_2(e_i)$ for all edges e_i. Cong et al. [64, 65] showed the following:

(Dominance) For any tree T with wire width assignment f_1, let f_2 be a wire width assignment obtained by a sequence of single wire width changes, each of which improves the delay objective while leaving all other wire widths fixed. Then f_2 dominates (is dominated by) f^* if and only if f_1 dominates (is dominated by) f^*.

The dominance property implies that the increasing and decreasing versions of SGW can provide lower and upper bounds on the optimum wire width for each edge in T.

Dynamic Wiresizing

The SGW method is called *static* because the interconnect topology is fixed before wiresizing begins. To take advantage of possible synergy between these two processes, Hodes et al. [125] have proposed a *dynamic wiresizing* heuristic which combines the Steiner Elmore routing tree (SERT) and static greedy

wiresizing (SGW) approaches. Starting with only the source terminal, the construction iteratively adds a new terminal to minimize the Elmore delay objective in the resulting *wiresized* topology. In other words, SGW is called once with candidate edge in the construction, and the edge which yields the lowest-delay wiresized tree is added. It should be noted that the wiresizing serves strictly as a guide, in that edges of a partial topology are restricted to have minimum width during the construction. Only when the topology spans the entire net is SGW invoked a final time and the resulting wiresized Steiner tree returned. The resulting DWSERT algorithm is described in Figure 3.34.

Algorithm Dynamically Wiresized Steiner Elmore Routing Tree (DWSERT)

Input: Signal net S with source $s_0 \in S$
Output: Wiresized low-delay Steiner tree spanning S

$T = (V, E) = (\{s_0\}, \emptyset)$
$M = S - \{s_0\}$
While $M \neq \emptyset$ do
  Find $u \in M$, and point w on some edge of E which minimizes the maximum Elmore delay from s_0 to any leaf in the *wiresized* tree SGW$(V \cup \{u, w\}, E \cup \{(u, w)\})$
  $V = V \cup \{u, w\}$
  $E = E \cup \{(u, w)\}$
  $M = M - \{u\}$
Output SGW$(T = (V, E))$

Figure 3.34 Algorithm DWSERT: constructing a dynamically wiresized low-delay routing tree.

Hodes et al. [125] have compared the performance of DWSERT against that of I1S, SERT, A-tree, and the statically wiresized versions I1S + SGW, SERT + SGW, and A-tree + SGW. Their testbed consists of sets of 50 random nets with terminal locations chosen from a uniform distribution in the $100000\mu \times 100000\mu$ grid, and the source terminal randomly chosen in each net. Interconnect parameters correspond to the MCM technology in Table 3.1, i.e., a regime where A-tree and wiresizing achieve performance gains.

Table 3.10 gives the average percentage reduction in SPICE-computed maximum sink delay, relative to I1S values. Static wiresizing substantially improves sink delay when applied to the I1S or A-tree topologies. However, SERT topologies admit less improvement since they are already highly star-like for the MCM technology (there is no advantage to widening a direct source-sink tree edge). The table also gives the average percentage increase in wiring area, again rela-

tive to I1S values. DWSERT, along with A-tree + SGW, appears superior to the other methods. Finally, Figure 3.35 shows the wiresized I1S, A-Tree, and DWSERT constructions for a small random net.

Max Sink SPICE Delay (↓) / Wire Area (↑)		
Algorithm	$\|S\| = 5$	$\|S\| = 10$
I1S	-0.0 / +0.0	-0.0 / +0.0
I1S + SGW	-28.6 / +38.2	-34.4 / +33.5
A-tree	-7.4 / +0.3	-23.4 / +5.6
A-tree + SGW	-38.2 / +90.0	-53.6 / +89.3
SERT	-27.2 / +66.4	-52.2 / +138.1
SERT + SGW	-31.6 / +111.1	-54.4 / +173.4
DWSERT	-32.2 / +62.1	-56.2 / +99.9

Table 3.10 Performance comparisons for DWSERT and I1S, SERT, and A-tree constructions, as well as their wiresized (+ SGW) versions. We show average percentage reduction in maximum sink delay, and average percentage increase in wiring area; both are with respect to I1S.

3.4.2 Non-Tree Routing

An implicit premise of previous methods is that the interconnection topology must be a tree. In retrospect, this is natural since a tree achieves electrical connectivity with a minimum amount of wire. We conclude this chapter with a brief investigation into "non-tree" routing, i.e., the use of arbitrary *routing graph* topologies. While delay minimization remains our central motivation, non-tree routing can have other advantages, including reduction of reflections, increased reliability, and reduced skew in sink delays. The latter two considerations have led to some previous use of non-tree topologies for VLSI routing: (i) for power/ground distribution, graph topologies are used to enhance reliability by lowering current densities and electromigration [80, 89, 90], and (ii) for clock distribution, graph topologies are used to control skew and minimize the impact of process variation [175].

It is easy to see that adding extra wires to an existing routing tree can improve certain source-sink delays. While additional wire will always increase the total tree capacitance, creating multiple (parallel) paths can substantially lower particular internode resistances, as shown in Figure 3.36. Consequently, with the

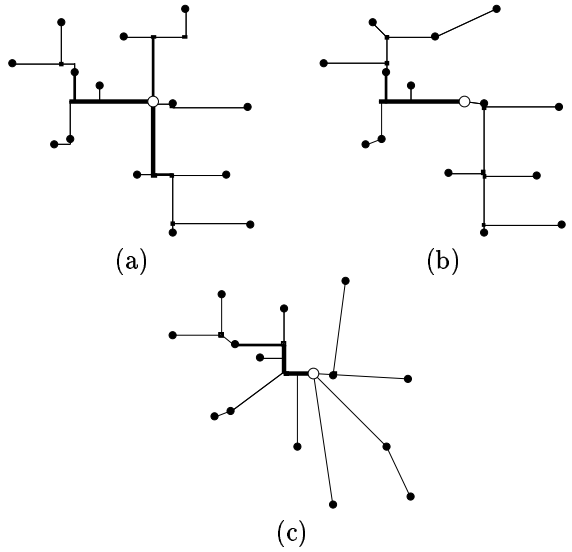


Figure 3.35 Comparison of routing tree constructions for a random 15-terminal net (hollow dot is s_0). (a) A-tree + SGW has maximum sink delay = 3.00ns; (b) I1S + SGW has maximum sink delay = 4.05ns; (c) DWSERT has maximum sink delay = 2.55ns.

trend toward thinner and more resistive VLSI interconnects, the use of non-tree routing seems increasingly attractive.

McCoy and Robins [181] have studied the following *Optimal Routing Graph* (ORG) problem, which is a generalization of the ORT problem discussed above.

The Optimal Routing Graph (ORG) Problem: Given a signal net $S = \{s_0, s_1, ..., s_n\}$ with source s_0, find a set N of Steiner points and routing graph $G = (S \cup N, E)$ such that G spans S and minimizes $t(G) = \max_{i=1}^{n} t(s_i)$.

The ORG problem extends to address a critical-sink formulation by associating a criticality parameter $\alpha_i > 0$ with each sink s_i. We then seek a routing graph

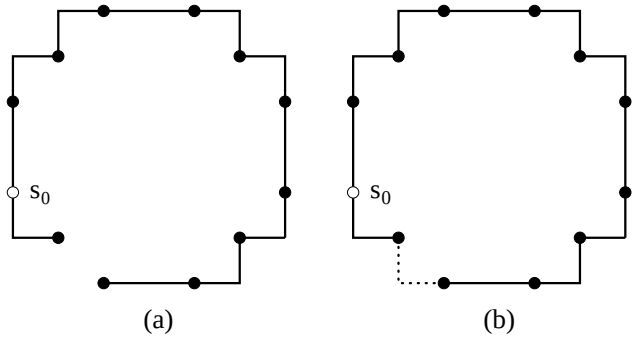


Figure 3.36 Adding an extra edge to the MST reduces maximum source-sink SPICE delay from 1.3ns in (a) to 1.0ns in (b), while incurring a wirelength penalty of 9%. Simulation parameters correspond to a MOSIS 0.8μ CMOS process.

that minimizes $\sum_{i=1}^{n} \alpha_i \cdot t(s_i)$, and for tractability of the delay calculation again use the distributed RC representation and Elmore's delay approximation. Note that the discussion of Section 3.1.2 treats only the computation of Elmore delay in an RC tree. Chan and Karplus have given an efficient computation for Elmore delay in general RC graph topologies [41] (see also Martin and Rumin [179]). The method of Chan and Karplus decomposes the interconnect graph into a spanning tree plus a set of m extra edges; the extra edges are added back into the graph one by one, and the Elmore delay is updated with each edge. The time complexity of this Elmore delay calculation is $O(n \cdot m)$.

A Simple ORG Heuristic

In [181], the ORG problem is addressed as follows. Starting with a "reasonable" initial topology (e.g., a heuristic SMT or an MST), a new edge is found which minimizes the delay objective in the resulting routing graph. This edge is then added to the routing graph, and the process is iterated until no edge will further improve the delay. Steiner points may be introduced via edge-edge and point-edge connections to afford greater flexibility in the delay and wirelength optimization. A formal description of the resulting *Low Delay Routing Graph* algorithm is given in Figure 3.37. Although the LDRG complexity is $\Theta(n^4)$

per iteration, the method will be reasonably efficient for most nets, e.g., those having 10 or fewer terminals.

Algorithm Low Delay Routing Graph (LDRG)
Input: signal net S with source s_0 **Output:** low-delay routing graph $G = (\hat{S}, E)$
Compute a Steiner tree $G = (\hat{S}, E)$ over $\hat{S} = S \cup N$, where N are the possible Steiner points, and $E \subseteq \hat{S} \times \hat{S}$ is the set of Steiner tree edges **While** there is an edge $e_{ij} \in \hat{S} \times \hat{S}$ which minimizes $t((\hat{S}, E \cup \{e_{ij}\})) < t(G)$ **Do** $E = E \cup \{e_{ij}\}$ **Output** G

Figure 3.37 The Low Delay Routing Graph (LDRG) algorithm: greedy construction of a low delay routing graph based on a heuristic Steiner tree. Elmore delay t_{ED} is used to guide the construction.

The LDRG heuristic has been tested on sets of 100 random nets for each of several net sizes, with terminal locations chosen from a uniform distribution in a square layout region. Interconnect technology parameters correspond to IC1, IC2 and MCM in Table 3.1. The particular LDRG implementation that we discuss begins with the I1S heuristic SMT, and uses the code of Chan and Karplus [41] in the Elmore delay computation. For greater accuracy, SPICE3e2 is used to evaluate signal delays in the LDRG output graph.

Figure 3.38(a) shows the average percentage reduction in maximum sink delay, compared with the I1S algorithm. Substantial improvement can be seen for the MCM and IC2 technologies; with the former, average delay improvement is 38% for 5-terminal nets and 44% for 10-terminal nets. Corresponding increases in tree cost, versus I1S, are shown in Figure 3.38(b). It appears that for the LDRG method, the percentage improvement in delay represents a reasonable return on the percentage increase in tree cost. Interestingly, the MCM technology seems to admit a regime where the maximum delay and the tree cost are both decreasing at the same time.

Finally, an added benefit of non-tree routing is a significant reduction in signal skew (i.e., the maximum difference between signal arrival time between any two sinks). Figure 3.38(c) shows the average percentage improvement in signal skew versus the I1S Steiner routing. For 10-terminal nets, LDRG yields 44%

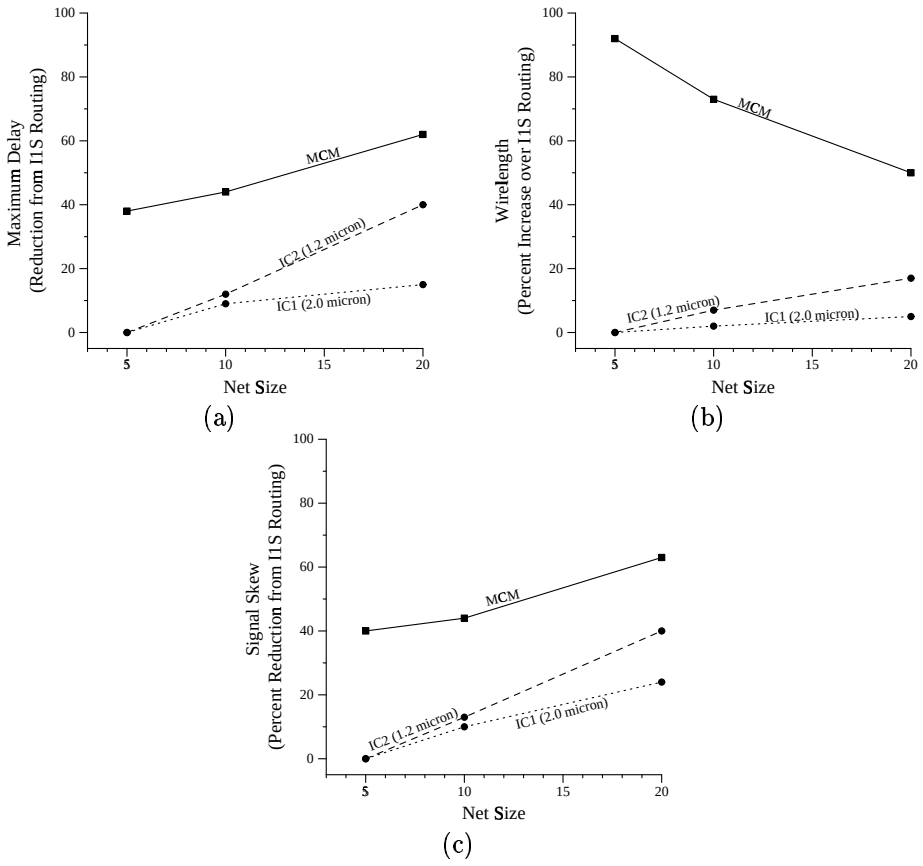


Figure 3.38 (a) Average percentage reduction in maximum sink delay, versus I1S routing. In other words, we plot the quantity $1 - \frac{t_{ED}(LDRG)}{t_{ED}(I1S)}$, expressed as a percentage; (b) Average percentage increase in tree cost, versus I1S routing; (c) Average percentage reduction in skew of signal arrival times, versus I1S routing.

skew reduction for the MCM technology, and 13% and 10% skew reductions for the IC2 and IC1 technologies, respectively.

In conclusion, non-tree routing topologies seem promising for performance optimization, particularly in regimes where long signal routes must be made, or where routing densities are low and an area-speed tradeoff is possible (e.g., MCM substrate routing). Furthermore, non-tree routings possess such advantages as open-fault tolerance, reduced skew, and reduced signal reflection [40]. On the other hand, issues such as signal wavefront interference (due to multiple point-to-point conduction paths), and the resulting possibility of false switching, may need careful investigation. In the future, one can envision extensions of LDRG to encompass critical-sink routing and wiresizing, as well as a better initial Steiner topology in the construction.

4

SKEW

Overview of the Chapter

The heart of a digital system is its *clock*, which is the control signal that synchronizes the flow of data among functional elements. To achieve maximum system performance, it is necessary to limit the *clock skew*, i.e., the maximum difference in arrival times of the clock signal at synchronizing elements (sequential registers, or *clock sinks*) of the design. This has been idealized in the recent literature as the "zero-skew clock routing problem", which seeks a routing tree that delivers the synchronizing clock pulse from its source to all clock sinks simultaneously. At the same time, the cost of the clock routing tree must be minimized in light of system power requirements, signal integrity, and area utilization. This chapter views clock tree construction to minimize skew and tree cost as a combination of two processes – *topology generation* and *geometric embedding* – and presents methods which accomplish each of these processes using either linear delay or Elmore delay to guide the construction. Our focus is on the sequence of recent works by Jackson et al. [143], Kahng et al. [144], Tsay [240], Boese et al. [29] Chao et al. [44, 45], Edahiro [82, 84], Zhu and Dai [259], and Kahng and Tsao [153] which lead to the present state of single-layer, exact zero-skew clock tree constructions.

In the first part of this chapter, the linear delay model is used to motivate a *pathlength-balanced tree* problem formulation. We describe a class of simple methods, based on iterative geometric matching, which perform simultaneous topology generation and geometric embedding of the clock tree. These methods typically yield zero pathlength skew for both cell-based and building-block designs.

The second part of the chapter describes the Deferred-Merge Embedding (DME) algorithm, which in linear time embeds any given connection topology into the Manhattan plane with exact zero skew and minimum tree cost. The DME algorithm consists of two phases: (i) bottom-up identification of loci for "balance points" within a minimum-cost zero-skew tree, followed by (ii) top-down selection of actual locations for these balance points within the zero-skew solution. DME achieves substantial wirelength reductions over previous constructions in the literature, and can be applied with any monotone delay model (i.e., any model according to which sink delays are monotone in the length of any tree edge). Studies of various clock topology generators in conjunction with the DME embedding show the contribution of both the topology generation and the geometric embedding to successful clock tree synthesis. We also show the generality of the DME approach by describing extensions which address prescribed-skew clock routing, min-max delay constraints in general signal net routing, and a bounded-skew clock routing formulation.

Finally, the third part of the chapter reviews the topology generation and geometric embedding of DME's exact zero-skew construction, and unifies these ideas with the objective of a *single-layer*, or "planar-embeddable", clock routing solution. Under the linear delay model, the two phases of the DME algorithm can be replaced by a single top-down pass. Whereas the original DME algorithm required a prescribed topology as input, combining the two DME phases allows the clock tree topology to be determined dynamically and flexibly, at the same time that it is being embedded optimally. This naturally leads to a DME-like construction of a zero-skew, single-layer clock tree.

The chapter concludes by noting several additional issues and problem formulations, including sensitivity to process variation, design of buffering hierarchies for minimum phase delay, and design of two-level clock trees for multi-chip module packaging.

4.1 PRELIMINARIES

In synchronous VLSI designs, circuit speed is limited by two main factors: (i) delay on the longest path through combinational logic, and (ii) clock skew, which is the maximum difference in arrival times of the clocking signal at the synchronizing elements of the design. This is seen from the following inequality, which governs the clock period of a design [18]:

$$\text{clock period} \geq t_d + t_{skew} + t_{su} + t_{ds}$$

where t_d is the maximum delay on any path through combinational logic, t_{skew} is the clock skew, t_{su} is the setup time of the synchronizing elements (i.e., sequential registers, or *clock sinks*), and t_{ds} is the propagation delay within the synchronizing elements. The term t_d can be further decomposed into $t_d = t_{d\_interconnect} + t_{d\_gates}$, where $t_{d\_interconnect}$ is the delay through interconnect, and t_{d_gates} is the delay through logic devices on a given critical path. Scaling of VLSI technology decreases the terms t_{su}, t_{ds}, and t_{d_gates}, so that $t_{d\_interconnect}$ and t_{skew} increasingly dominate circuit performance. As noted by Bakoglu [18], within any given system design it is difficult to accommodate t_{skew} that is greater than 10% of the overall system clock period. As a result, there is a large literature dealing with the problem of clock skew minimization under various assumptions.

It is important to realize that many disparate architecture and circuit-level options exist for system clock distribution, but are not taken into account by our treatment. For example:

- the clocking can be pipelined, which brings into consideration the tradeoff between latency and clock frequency [93, 101, 102];
- a given clock can be single-phase or multi-phase, and can employ retiming or "cycle-stealing" techniques [174, 242];
- either a buffer hierarchy (possibly with parameterized buffer cells to match loading impedances) or a single monolithic buffer [18, 100] can be used to drive the clock routing topology;
- wiresizing, "snaking", or passive delay elements can be used to compensate for variation of interconnect and loading impedances, or to reduce sensitivity to process variation during manufacture [198, 240]; and
- high-level functional partitioning can enable multiple clock signals [99] or modular clocking (e.g., for low-power design).

An excellent review of such system-level design issues is provided in the work of Friedman (e.g., see [100]), and a more low-level discussion (e.g., of optimum cascaded driver design, interconnect scaling effects, and wire width optimization) is provided in [18]. In the following, we will focus on interconnect design,

rather than architecture- or device-level issues. Hence, the literature concerning several of the above considerations is beyond our present scope.

A number of clock tree constructions implicitly require small problem complexity. For hierarchical building-block design, Ramanathan and Shin [200] proposed a clock distribution scheme which enumerates all possible clock routings and clock buffer optimizations. The exhaustive search forces the number of blocks at each level of the hierarchy to be small. Burkis [37] and Boon et al.[35] have also proposed hierarchical approaches to clock tree synthesis involving geometric clustering and buffer optimization at each level, and other methods (e.g., Pullela et al. [197] for wire width optimization) similarly use exhaustive search of a relatively small solution space. A mathematical programming formulation which resynthesizes the clock tree to minimize the clock period was given by Fishburn [93]. Such methods have high algorithmic complexity and rely on a hierarchical clustering or a prespecified topology to yield practical runtimes. By contrast, we are interested in clock tree constructions for "flat" problem instances with many sinks at a single level, as will typically occur in large cell-based or multi-chip module designs. In practice, such clock routing instances arise after the placement phase of physical layout has determined the clock sink positions. Large cell-based designs can have clock nets with thousands of sinks located arbitrarily in the layout region.

Clock trees with many sinks were first designed using H-trees [19, 73, 94, 246]. The H-tree structure successfully controls clock skew, but applies chiefly when sinks have identical loading capacitances and are placed symmetrically, as in systolic array architectures. The first general clock tree construction for cell-based layouts with arbitrary sink locations was proposed by Jackson, Srinivasan and Kuh [143]: their method of means and medians (MMM) simultaneously generates and embeds a topology by recursively partitioning the set of sinks into two equally-sized subsets (according to a *median* x- or y-coordinate), and connecting the centroid (i.e., the *mean*) of the entire set to the centroids of the two subsets. The MMM solution exhibits reasonable skew on average, although it is possible to construct small examples for which source-sink pathlengths in the MMM solution may vary by as much as half of the chip diameter [144].

Definition of the Zero-Skew Clock Routing Problem

Formally, we define a clock routing instance to be a set of n sink locations in the Manhattan plane, $S = \{s_1, s_2, \ldots, s_n\} \subset \Re^2$. The set of sinks S is also called a *clock net*, and we often assume that it is embedded in the $L \times L$ grid.

A *connection topology* is a rooted binary tree, G, which has n leaves corresponding to the sinks in S. A *clock tree* $T(S)$ is an embedding of the connection topology in the Manhattan plane, i.e., a placement in $\Re^2$ that assigns each internal node $v \in G$ to a location $pl(T, v)$. When no confusion is possible, we will denote the placement of v simply as $pl(v)$. The clock routing solution can have internal nodes of degree greater than three if some edges of G have zero length in its embedding $T(S)$.

The clock entry point (CEP) of the clock tree is the *source*, s_0. A *terminal* generically denotes the CEP of any subtree of the routing solution; note that any sink by itself is a degenerate subtree of the clock tree, and is its own CEP. Given that the clock tree topology is rooted at the source, any edge between a parent node v and its child w may be identified with the child node; we denote this edge by e_w. The *cost* of e_w is given by its wirelength, denoted $|e_w|$, which is always at least as large as the Manhattan distance between the endpoints of e_w, i.e., $|e_w| \geq d(pl(v), pl(w))$. The cost of $T(S)$ is the sum of the edge costs in $T(S)$.

For a given clock tree $T(S)$, let $t(s_0, s_i)$ denote the signal propagation time on the unique path from the source s_0 to the sink s_i. The *skew* of $T(S)$ is the maximum value of $|t(s_0, s_i) - t(s_0, s_j)|$ over all sink pairs $s_i, s_j \in S$. If the skew of $T(S)$ is exactly zero then $T(S)$ is called a *zero-skew tree* (ZST).

The Zero-Skew Clock Routing (ZSCR) Problem: Given set S of sink locations, construct a ZST $T(S)$ with minimum cost.

This formulation does not consider the possibility of intermediate buffers between the source and the sinks. Thus, it is most relevant to a monolithic single-buffer (cascaded-driver) design. Bakoglu [18] states that the single-buffer approach is more effective than a buffer hierarchy; the Digital Equipment Corporation *Alpha* microprocessor [76] is a leading example of this philosophy. Of course, it should also be noted that the ZSCR formulation, along with its variants discussed below, fails to consider several practical issues.

- We do not consider buffer insertion, wire-sizing, or design issues pertaining to signal integrity (e.g., overshoot/undershoot and false switching, incorporation of slew rate into the delay model, etc.). Nevertheless, in submicron regimes it is becoming easier and more area-efficient to insert buffers rather than to add wire in achieving zero skew. Buffer insertion is also very useful in maintaining integrity of the clock signal waveform.

- Our geometric perspective assumes that interconnect technology parameters are the same on all metal routing layers, and ignores via resistances: only in this way can RC parameters of interconnect segments be derived from wirelength alone. More realistic formulations would incorporate different electrical parameters on the various metal layers, as well as process variation and the objective of process variation-independent skew management.
- The Elmore delay model, which is the most complicated model we use for optimization of large clock trees, computes source-sink delays using only lumped off-path capacitances. Actual sink delays will depend on the specific topology within subtrees that branch off from a given source-sink "main path" [205].[1]
- Finally, "exact zero-skew" is not always a real design goal: a circuit which triggers all synchronizing registers simultaneously may consume an unacceptable amount of power. Hence, a more realistic objective may be to distribute skew such that the registers trigger at different times, but without incurring timing violations.

Despite these limitations, our approach leads to basic techniques that can be easily extended to address more sophisticated or "realistic" objectives. For example, the discussion below outlines extensions to prescribed-delay or bounded-skew global routing, to hierarchical clock routing, and to single-layer clock routing.

4.2 AN EARLY MATCHING-BASED APPROACH

The linear delay approximation allows intuitive geometric ideas to motivate new algorithmic approaches to clock routing [29, 44, 45, 73, 82, 259]. For zero-skew clock tree routing, linear delay simply compares the lengths l_i of source-sink paths, and disregards any off-path topology. Although the Appendix observes that the linear model has poor accuracy and fidelity with respect to SPICE, note that linear delay is exact for emerging optical and wave interconnect [241]. In addition, linear delay has been successfully used in clock tree synthesis (e.g.,

[1]The DME method described below extends to arbitrary monotone delay models (recall that a delay model is *monotone* when increasing the length of a tree edge cannot decrease any source-sink delay). However, monotone models are limited in their ability to capture voltage response in transmission lines (reflection at discontinuities causes signal delay to be nonmonotone).

[73]). One possible explanation is that ignoring off-path topology entails "uniform error" across all sinks when the topology is balanced and when the distribution of sinks is uniform in the layout region; such is apparently the case with actual clock sink placements and clock tree layouts.

In this section, we assume the linear delay model and consider the resulting *pathlength-balanced tree* formulation [144]. The formulation relaxes the original Zero-Skew Clock Routing problem by allowing non-zero skew. Our focus is on a class of heuristics based on iterated geometric matching. The basic approach starts with a forest of subtrees, each of which contains a single sink of the clock net. At each level of the topology we combine pairs of subtrees into larger subtrees, using a heuristic geometric matching over the CEPs in the current forest. The end result is an embedded binary tree topology whose leaves are the sinks of the clock net and whose root (CEP) is the clock source. The method extends to building-block designs via matching in the channel intersection graph of the layout.

Our matching-based approach can guarantee perfect pathlength-balanced trees only for inputs with four or fewer sinks. Nevertheless, in practice the algorithm will yield essentially zero pathlength skew even for very large instances. The performance of this algorithm is "good" in the sense that the output tree cost is on average within a constant factor of the optimal Steiner tree cost, and the worst-case tree cost is bounded by $O(\sqrt{n})$ for n sinks in the unit square, which is the same bound as for the worst-case optimal Steiner tree cost. Furthermore, the matching-based approach seems to afford a good underlying topology for more sophisticated clock tree optimizations.

4.2.1 Pathlength-Balanced Trees

Under the linear delay model, clock skew is the same as *pathlength skew*, i.e., the maximum difference between any two source-sink pathlengths. A tree is a *perfect pathlength-balanced tree* if its pathlength skew is zero. It is not difficult to construct a perfect pathlength-balanced tree if we can use an unlimited amount of wire. For example, we can naively route separate wires of equal length from the source to each sink as shown in Figure 4.1(right), but the resulting tree cost can be an unbounded factor higher than the SMT cost. On the other hand, Figure 4.1(left) shows that minimizing tree cost can result in very large pathlength skew. We wish to construct a tree with both pathlength skew and tree cost as small as possible.

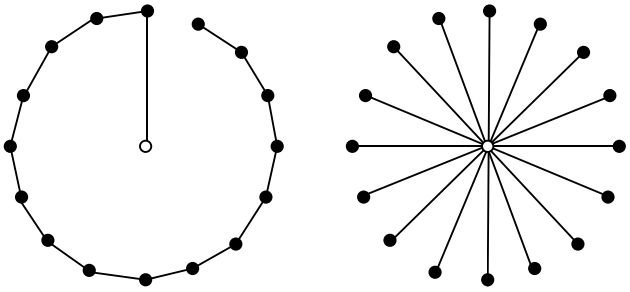


Figure 4.1 Neither the naive pathlength-balanced tree (right) nor the minimum-cost tree (left) is a good clock tree.

The Pathlength-Balanced Tree (PBT) Problem: Given a set of sinks S and a real parameter ψ, find a minimum-cost tree over S having pathlength skew $\leq \psi$.

By setting $\psi = \infty$, the PBT problem simplifies to the NP-complete rectilinear SMT problem. However, the complexity of finding the minimum-cost perfect pathlength-balanced (i.e., zero-skew) tree is still open. In devising a heuristic, our first goal is to achieve an expected routing tree cost (for n random sink locations in the $L \times L$ grid) of $O(L \cdot \sqrt{n})$, since this is also the asymptotic expected cost of the SMT.

4.2.2 The Iterated Matching Approach

Definition: Given a set of k terminals, a *geometric matching* consists of exactly $\lfloor \frac{k}{2} \rfloor$ *edges* between the terminals, with no two edges sharing an endpoint.

The cost of a geometric matching is the sum of the costs of its edges, and the matching is *optimal* if it has minimum cost. Figure 4.2 shows an optimal geometric matching over four terminals.

To construct a tree by iterative matching, we begin with a forest of n terminals corresponding to the sinks of the clock net. Each terminal is the CEP of a degenerate subtree which consists of a single sink, and we will merge these trees in bottom-up fashion until the entire clock tree is obtained. The optimal geometric matching on the n CEPs has $\lfloor \frac{n}{2} \rfloor$ edges, each of which defines a

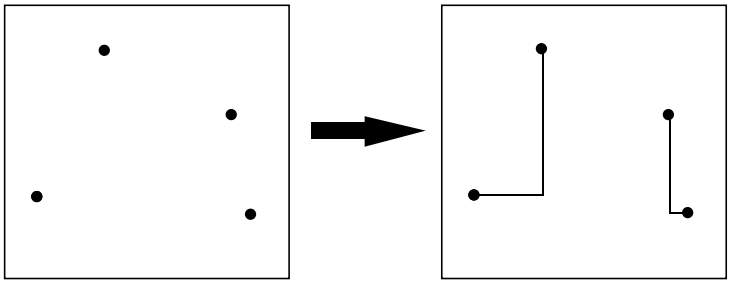


Figure 4.2 An optimal geometric matching over four terminals.

subtree containing two sinks. The optimal (zero skew) CEP for each of these subtrees is the midpoint of the corresponding edge.

In general, each level of the iteration will match terminals corresponding to CEPs (roots) of subtrees in the current forest. However, observe that the matching calculation is oblivious to varying root-leaf pathlengths among the subtrees of the current forest. Thus, when two subtrees are merged into a larger subtree, the optimal new CEP is not necessarily equidistant from the CEPs of the two subtrees. We choose the CEP of each new merged subtree to be the *balance point* p which (i) lies on the "straight" line segment connecting the roots of the two subtrees (i.e., the Euclidean embedding of the matching edge) and (ii) minimizes the maximum pathlength skew from p to the sinks of the merged subtree. Computing the balance point can be done in constant time if we know the minimum and maximum source-sink pathlengths of each subtree; the corresponding values for the new merged subtree can be updated in constant time.

At each level we match only half as many nodes as at the previous level, and the clock tree solution is obtained after $\lceil \log n \rceil$ matching iterations. (If a given level has $2m+1$ CEPs, we find the optimal m-edge matching and match $m+1$ CEPs at the next level.) Figure 4.3 gives a formal description of this algorithm, which we call CLOCK1, and Figure 4.4 illustrates its execution.

Two results establish that $cost(T_{CLOCK1})$ grows at the same asymptotic rate as the worst-case optimal Steiner tree cost, and that $cost(T_{CLOCK1})$ is on average within a constant factor of the optimal Steiner tree cost. (Similar bounds have been established for more recent clock tree constructions, e.g., [84, 259].)

Algorithm CLOCK1: Pathlength-Balanced Tree heuristic for cell-based designs

Input: Clock net S
Output: Pathlength-balanced tree T_{CLOCK1} with root CEP

$T = \emptyset$
$P = S$
While $|P| > 1$
  M = edges of an optimal geometric matching over P
  $P' = \emptyset$
  For $(p_1, p_2) \in M$ **Do**
    T_1 = the subtree of T rooted at p_1
    T_2 = the subtree of T rooted at p_2
    p = a point lying *between* p_1 and p_2 on the line segment from p_1 to p_2, such that p minimizes pathlength skew of the subtree $T_1 \cup T_2 \cup \{(p, p_1), (p, p_2)\}$ rooted at p
    $P' = P' \cup \{p\}$
    $T = T \cup \{(p, p_1), (p, p_2)\}$
  $P = P'$ (plus one unmatched node if $|P|$ was odd)
CEP = root of T = single remaining point in P
Output $T_{CLOCK1} = T$

Figure 4.3 Algorithm CLOCK1: matching-based pathlength-balanced tree heuristic for cell-based designs.

Theorem 4.2.1 *For n arbitrary sink locations in the $L \times L$ grid, $cost(T_{CLOCK1}) = O(L \cdot \sqrt{n})$.*

Proof: For any k terminals in the $L \times L$ grid, the maximum possible cost of an optimal matching is $O(L \cdot \sqrt{k})$ [233]. Since the tree is formed by the edges of a matching on n terminals, plus the edges of a matching on $\frac{n}{2}$ terminals, etc., the tree cost is at most

$$O(L \cdot \sqrt{n}) + O(L \cdot \sqrt{\frac{n}{2}}) + O(L \cdot \sqrt{\frac{n}{4}}) + ... = O(L \cdot \sqrt{n}).$$

□

This is of the same order as the maximum possible cost of an optimal SMT over n terminals in the $L \times L$ grid [229]. A second result addresses the instance-wise relationship between the CLOCK1 tree cost and the optimal Steiner tree cost.

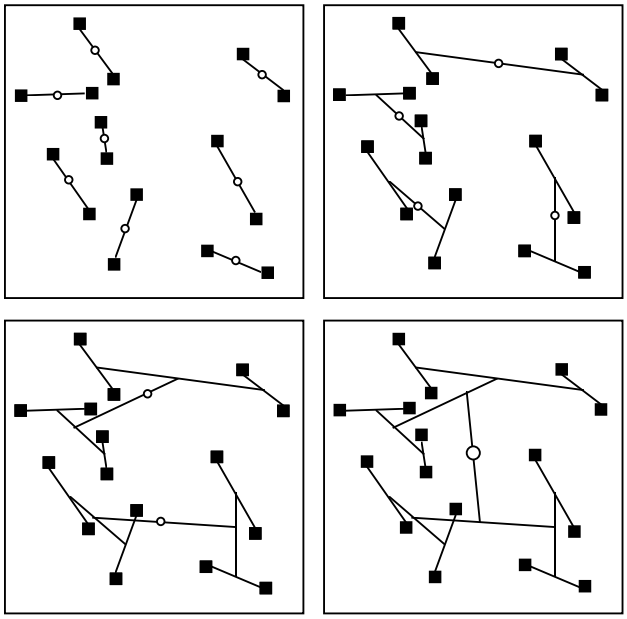


Figure 4.4 Example execution of CLOCK1 on a 16-sink clock net. Solid dots denote terminals, and hollow dots represent the balance points of matching edges. At each level, a geometric matching is computed on the balance points from the previous level. Note that although edges are depicted as straight lines, they are routed rectilinearly.

Theorem 4.2.2 *For nets with sink locations randomly chosen from a uniform distribution in the $L \times L$ grid, $cost(T_{CLOCK1})$ is on average within a constant factor of the optimal Steiner tree cost.*

Proof: The expected minimum Steiner tree cost for n terminals randomly chosen from a uniform distribution in the $L \times L$ Manhattan grid is $\beta \cdot L \cdot \sqrt{n}$, for some constant β [229]. The result follows from the $O(L \cdot \sqrt{n})$ upper bound on the optimal matching cost at any level of the construction. □

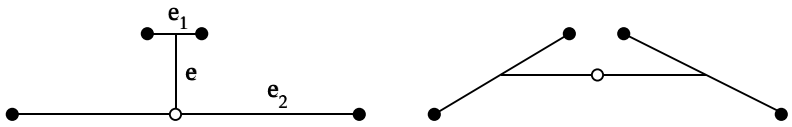


Figure 4.5 H-flipping to reduce pathlength skew: the "H" configuration at left has no zero-skew balance point along the "bar of the H", while the "H" on the right has a zero-skew balance point.

Practical Improvements

Several enhancements are useful in implementing CLOCK1. First, recall that the balance point computation was needed because the matching is oblivious to the difference in source-sink pathlength within the matched subtrees. Computing a balance point intuitively entails "sliding" the CEP along the "bar of the H" (see Figure 4.4), but this is not always sufficient to balance the source-sink pathlengths exactly. Thus, a heuristic optimization called *H-flipping* was proposed in [144]; see Figure 4.5. For each edge e which matches CEPs on edges e_1 and e_2, H-flipping compares the "H" formed by e, e_1, and e_2 with the other "H" over the same four terminals, then selects the alternative with smaller pathlength skew , breaking ties toward smaller tree cost. When there are four sinks in the clock net, H-flipping guarantees zero pathlength skew with at most a factor of three increase in tree cost over the original matching-based construction; this result was shown in [60]. Although this guarantee does not hold for more than four sinks, in practice the H-flipping refinement seems to nearly always yield perfect pathlength-balanced trees, and has negligible effect on tree cost. Since H-flipping requires constant time per terminal, it does not affect the asymptotic time complexity of CLOCK1.

A second implementation issue concerns the complexity of the matching subroutine, which effectively determines the overall CLOCK1 time complexity. Consider the $\lceil \log n \rceil$ matching iterations performed by CLOCK1, and let the underlying matching algorithm require time $S(n) = \Omega(n)$ We may write $S(n) = n \cdot S'(n)$ where $S'(n) = \frac{S(n)}{n}$ is monotonically non-decreasing, and the time complexity of CLOCK1 is:

$$S(n) + S(\frac{n}{2}) + S(\frac{n}{4}) + \ldots \quad = \quad n \cdot S'(n) + \frac{n}{2} \cdot S'(\frac{n}{2}) + \frac{n}{4} \cdot S'(\frac{n}{4}) + \ldots$$

$$\begin{aligned} &\leq n \cdot S'(n) + \frac{n}{2} \cdot S'(n) + \frac{n}{4} \cdot S'(n) + \ldots \\ &= S'(n) \cdot (n + \frac{n}{2} + \frac{n}{4} + \ldots) \\ &\leq 2n \cdot S'(n) = 2S(n) = O(S(n)) \end{aligned}$$

Weighted matching in general graphs can be solved in $O(n^3)$ time [163], and planar geometry allows a speedup to $O(n^{2.5} \log n)$ time [243]. However, such runtimes are still impractical for large instances. Since there is no clear relationship between the optimality of the iterated matching and the pathlength skew of the resulting tree, a practical implementation might employ a more efficient matching heuristic, such as the $O(n \log^2 n)$ greedy approach of Supowit [231].

Finally, a third practical consideration is that a heuristic matching might contain edges that cross each other when embedded in the plane. Seemingly, the output tree can be improved by uncrossing pairs of intersecting edges in the heuristic matching: this will reduce the matching cost in any metric. One can find the k intersections of n line segments in $O(n \log n + k)$ time [47].

4.2.3 Extension to Building-Block Design

Bottom-up iterative matching may also be applied to building-block design, where the layout consists of rectangular blocks with arbitrary size and location in the $L \times L$ grid. Routing is carried out in the regions between blocks (no two blocks abut), and is also possible along the perimeter of the layout. We may represent the layout using a channel intersection graph (CIG), G [58, 70, 158, 193].

Recall that in a graph G with non-negative edge costs, $minpath_G(x, y)$ is a minimum-cost path between nodes x and y, and $dist_G(x, y)$ is the cost of $minpath_G(x, y)$. In the CIG, routing cost between two terminals is no longer approximated by geometric distance, but is instead given by $dist_G(x, y)$. However, we assume that every edge weight in the CIG reflects a geometric distance, namely, the length of the corresponding channel. Thus, the routing graph can be considered to be a subgraph of the $L \times L$ gridgraph. Our objective is still to construct a tree with both cost and pathlength skew as small as possible, subject to tree edges being routed within the routing channels.

Definition: Given a graph $G = (V, E)$ with non-negative edge costs and a set of vertices $S \subseteq V$, a *generalized matching* M over S is a set of shortest paths connecting m disjoint vertex pairs in S, i.e., $M = \{minpath_G(x_1, y_1), minpath_G(x_2, y_2), \ldots, minpath_G(x_m, y_m)\}$, where all $x_i, y_i \in S$ are distinct.

A generalized matching over $S \subseteq V$ is *complete* if $m = \lfloor \frac{|S|}{2} \rfloor$. The *cost* of a generalized matching M is the sum of the costs of the shortest paths in the matching, i.e., $cost(M) = \sum_{i=1}^{m} dist_G(x_i, y_i)$. An *optimal* complete generalized matching on $S \subseteq V$ is one with minimum cost.

Lemma 4.2.3 *Each edge of G belongs to at most one shortest path in an optimal complete generalized matching over $S \subseteq V$.*

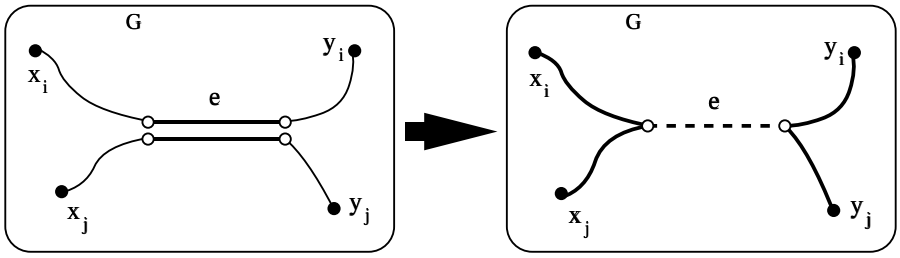


Figure 4.6 An edge can belong to at most one shortest path in an optimal complete generalized matching.

Proof: Let M be an optimal complete generalized matching over S. Suppose edge e appears in distinct shortest paths $minpath_G(x_i, y_i)$ and $minpath_G(x_j, y_j)$ in M as shown in Figure 4.6. We have that

$$dist_G(x_i, x_j) + dist_G(y_i, y_j) \leq dist_G(x_i, y_i) + dist_G(x_j, y_j) - 2 \cdot cost(e),$$

i.e., we can obtain another complete generalized matching over S with smaller cost by replacing $minpath_G(x_i, y_i)$ and $minpath_G(x_j, y_j)$ by $minpath_G(x_i, x_j)$ and $minpath_G(y_i, y_j)$. This contradicts the optimality of M. □

Lemma 4.2.4 *The routing cost between any two terminals of G in the $L \times L$ grid is $\leq 2L$.*

Proof: Given terminals x and y in G, let P_1 be any monotone (staircase) path passing through x and connecting two opposite corners w and w' of the layout grid. Clearly, $cost(P_1) = 2L$. Similarly, let P_2 be a monotone path passing through y and connecting w and w'. Since $cost(P_1) + cost(P_2) = 4L$, either w or w' will be reachable from both x and y with total routing cost at most $2L$, implying $dist_G(x, y) \leq 2L$. □

Using the result of Lemma 4.2.4, an optimal complete generalized matching over n terminals in G has cost at most $2L \cdot \lfloor \frac{n}{2} \rfloor \leq n \cdot L$. Note that this is independent of the number of blocks in the layout.

As before, we may construct a heuristic pathlength-balanced tree via iterated generalized matching over a current set of clock terminals (CEPs). We begin with a forest of n isolated terminals in G corresponding to the sinks of the clock net, and at each level compute an optimal generalized matching over the set of CEPs of subtrees in the current forest. The CEP of each new subtree is the point on the corresponding shortest path in the matching which minimizes pathlength skew among the leaves in the two merged subtrees. Figure 4.7 formally describes the resulting CLOCK2 heuristic, and Figure 4.8 shows an example execution.

Theorem 4.2.5 *For n sinks in the $L \times L$ grid, $cost(T_{CLOCK2}) \leq 2nL$.*

Proof: By Lemma 4.2.4, the cost of a generalized matching on n terminals is bounded by nL. After each iteration, the number of nodes to be matched is reduced by half. Therefore, $cost(T_{CLOCK2}) \leq nL + \frac{nL}{2} + \frac{nL}{4} + \ldots \leq 2nL$. □

To analyze the CLOCK2 time complexity, observe that computing an optimal generalized matching over the set of terminals S requires an edge-weighted complete graph G' over S, with the weight of each (x, y) edge corresponding to $dist_G(x, y)$. Given $G = (V, E)$, the graph G' can be obtained using an $O(|E| \cdot |V| + |V|^2)$ implementation of Floyd's all-pairs shortest paths algorithm [213]. A channel intersection graph induced by b blocks typically (cf. [38]) has $|V| = O(b+n)$, and typically $b = O(n)$. Since G is planar, $|E| = O(|V|)$. Thus, the optimal matching in G' can be obtained in $O(b^2 + n^3)$ time [163].

As with CLOCK1, the time complexity of optimal matching may be impractical, in which case a fast matching heuristic should be used. The heuristic complete generalized matching may be improved by removing any overlapping edges of shortest paths (cf. Lemma 4.2.3), so that no edge appears in more

Algorithm CLOCK2: Pathlength-Balanced Tree heuristic for building-block designs

Input: Clock net S embedded in routing graph G
Output: Pathlength-balanced tree T_{CLOCK2} with root CEP

$T = \emptyset$
$P = S$
While $|P| > 1$
  M = optimal complete generalized matching on P
  $P' = \emptyset$
  For $\{p_1, p_2\} \in M$ **Do**
    T_1 = subtree of T rooted at p_1
    T_2 = subtree of T rooted at p_2
    p = balance point on $minpath_G(p_1, p_2)$ minimizing the skew of the tree $T_1 \cup T_2 \cup minpath_G(p_1, p_2)$
    $P' = P' \cup \{p\}$
    $T = T \cup \{\{p, p_1\}, \{p, p_2\}\}$
  $P = P'$ (plus one unmatched node if $|P|$ was odd)
CEP = Root of T = single remaining point in P
Output $T_{CLOCK2} = T$

Figure 4.7 Algorithm CLOCK2: matching-based pathlength-balanced tree heuristic for building-block designs.

than one shortest path. When we use a fast matching subroutine, such as the greedy heuristic [224], the time complexity of each CLOCK2 iteration is dominated by the $O(b^2)$ all-pairs shortest paths computation. Because there are $O(\log n)$ levels in the tree construction, the overall CLOCK2 time complexity in this case is $O(b^2 \cdot \log n)$.

4.2.4 Empirical Tests

The heuristics CLOCK1 and CLOCK2 were implemented in ANSI C; we now summarize the experimental results.

Results for Cell-Based Designs

Three basic variants of CLOCK1 were tested, corresponding to three efficient matching subroutines. The first variant, called **SP**, uses the $O(n)$ space par-

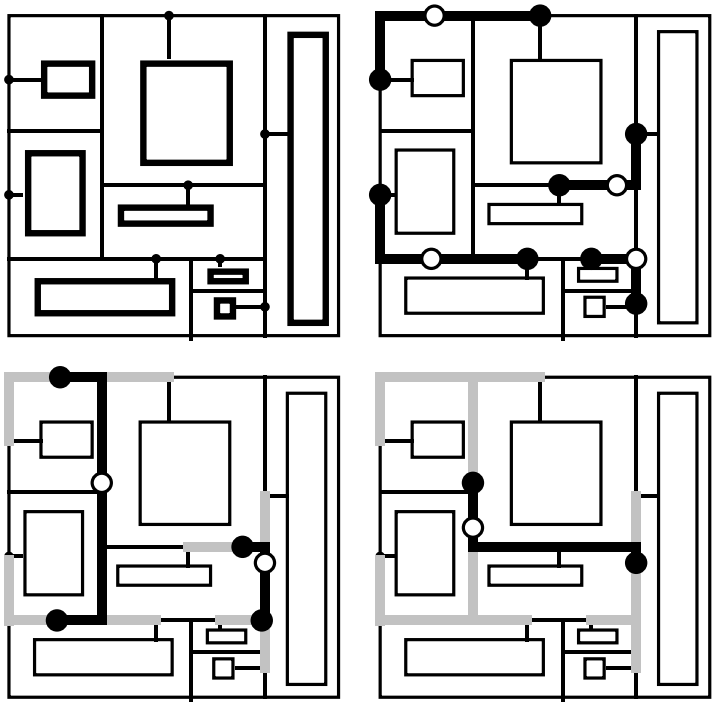


Figure 4.8 CLOCK2 execution on an 8-sink clock net in a random block placement. Solid dots are roots (CEPs) of subtrees in the previous level, and hollow dots are roots of new subtrees at the current level. The newly added routing is highlighted at each level.

titioning heuristic of [232] to induce a heuristic matching through recursive bisection of the layout region (by contrast, the MMM method of Jackson et al. is based on bisection of the set of terminal locations). The second variant, called **GR**, uses an $O(n \log^2 n)$ greedy matching heuristic [231] which always adds the shortest edge between unmatched terminals. The third variant, called **SFC**, uses an $O(n \log n)$ spacefilling curve-based method [22] to map the layout plane to a circle, thus inducing an ordering of the terminal locations. The SFC variant then chooses the better of the two embedded matchings (i.e., either all odd edges or all even edges in the induced tour through the terminals). Although each of these methods was originally proposed for Euclidean planar match-

ing, each also performs well in Manhattan geometry. Each of these matching variants was tested both with and without the following two refinements: (i) removing all edge crossings in the heuristic matching, and (ii) performing H-flipping as necessary. Since either refinement can be used independently with any matching variant, twelve distinct versions of CLOCK1 result. These are summarized as follows.

- **SP, GR, SFC**.
- **SP+E, GR+E, SFC+E** – Same as SP, GR and SFC, respectively, except that the heuristic matching cost is improved by edge-uncrossing.
- **SP+H, GR+H, SFC+H** – Same as SP, GR and SFC, respectively, except that pathlength skew and/or tree cost is improved by H-flipping.
- **SP+E+H, GR+E+H, SFC+E+H** – Same as SP, GR, and SFC, respectively, except that both edge-uncrossing and H-flipping are performed.

For comparison, we also implemented

- **MMM** – The method of means and medians, similar to the implementation described by Jackson et al. [143].

These 13 algorithms were tested on random clock nets with up to 1024 sinks, generated from a uniform distribution in the 1000×1000 grid. Results averaged over 50 random instances of each size are summarized below: Tables 4.1 and 4.2 give the average tree costs and Tables 4.3 and 4.4 give the average pathlength skews for all heuristics. All data in the tables are in grid units.

From the tables, we see that the edge-uncrossing and H-flipping refinements each improve tree cost and pathlength skew. When the refinements are combined, average pathlength skew is close to zero, and tree cost is generally superior to that of MMM. The best variant appears to be GR+E+H, i.e., CLOCK1 with a greedy matching heuristic, edge-uncrossing and H-flipping.[2] (Coincidentally, of the three matching heuristics used, only the greedy method has

[2] Any set of approximation heuristics will induce a *meta-heuristic* which for any given instance returns the best solution found by any heuristic in the set. Interestingly, in our experience the meta-heuristic of all 12 CLOCK1 variants always returns a perfect pathlength-balanced tree. This is potentially useful since our heuristics are all of similar complexity; for example, we can solve the Primary1 benchmark with all twelve variants using approximately 180 seconds of Sun SPARC-1 CPU time.

$\|S\|$	MMM	SP	GR	SFC	SP+E	GR+E	SFC+E
4	1197	1155	1136	1140	1129	1129	1130
8	2136	2075	2032	2031	1990	1990	1992
16	3506	3582	3409	3527	3343	3326	3343
32	5598	5922	5481	5788	5342	5277	5326
64	8377	9184	8526	9048	8100	8032	8068
128	12276	13793	12632	13656	11912	11725	11976
256	17874	20765	18625	20354	17573	17024	17768
512	25093	30443	27055	29618	25341	24548	25720
1024	36765	44304	38688	42750	36444	35086	37056

Table 4.1 Average clock tree cost for the various heuristics.

$\|S\|$	SP+H	GR+H	SFC+H	SP+E+H	GR+E+H	SFC+E+H
4	1125	1125	1125	1125	1125	1125
8	2027	2028	1994	1971	1979	1980
16	3502	3416	3428	3333	3322	3329
32	5860	5628	5577	5329	5273	5304
64	9226	8794	8748	8076	7982	8047
128	13997	3315	13159	11871	11697	11914
256	21307	19611	19713	17457	16955	17629
512	31646	29175	28688	25188	24465	25483
1024	46417	42110	41540	36276	34965	36814

Table 4.2 Average clock tree cost (continued).

worst-case cost that is asymptotically of the same order as the optimal matching cost [224].) Tables 4.5 and 4.6 highlight the contrast between GR+E+H and MMM, showing minimum, maximum and average values of both tree cost and pathlength skew.

Finally, Figure 4.9 depicts the GR+E+H output for the Primary2 test case, using the same sink placement as in [143]. Edges in the figure are depicted as straight lines, but are actually routed rectilinearly. For this instance,MMM results were: tree cost = 406.3 and pathlength skew (measured as standard deviation of pathlengths) = 0.74 [226]. By contrast, GR+E+H results were: tree cost = 376.7 and pathlength skew = 0.00. HSPICE simulations confirm sub-

$\|S\|$	MMM	SP	GR	SFC	SP+E	GR+E	SFC+E
4	112.31	3.98	15.52	0.00	0.00	0.00	0.00
8	186.10	45.79	76.71	4.26	0.66	0.66	0.66
16	234.72	70.93	141.22	19.47	4.01	3.54	3.66
32	262.61	143.85	200.33	28.29	8.14	7.85	6.14
64	229.15	179.83	273.04	51.36	6.93	8.65	5.29
128	201.55	226.61	314.05	64.86	11.52	14.18	11.26
256	183.28	286.90	324.57	85.10	17.25	13.85	15.04
512	153.90	321.23	399.29	85.46	14.79	15.26	15.73
1024	125.34	339.34	402.59	89.75	17.14	16.71	15.35

Table 4.3 Average pathlength skew for the various heuristics.

$\|S\|$	SP+H	GR+H	SFC+H	SP+E+H	GR+E+H	SFC+E+H
4	0.00	0.00	0.00	0.00	0.00	0.00
8	3.38	0.12	0.00	0.00	0.00	0.00
16	1.80	3.80	0.12	0.00	0.00	0.00
32	3.53	8.64	0.00	0.00	0.00	0.00
64	13.17	27.69	1.26	0.00	0.00	0.00
128	20.79	40.34	3.18	0.00	1.02	0.24
256	41.79	51.87	7.49	0.00	0.92	0.00
512	76.35	90.66	13.51	0.39	0.62	0.39
1024	75.92	94.99	16.62	0.44	0.08	0.38

Table 4.4 Average pathlength skew (continued).

nanosecond skew for the GR+E+H routing solution, using MOSIS 2.0μ CMOS parameters and 0.3pF gate loading capacitance [60]. It is somewhat surprising that clock skew can be controlled simply by balancing root-leaf pathlengths; as discussed below, this phenomenon may be due to the matching-based approach somehow providing an inherently robust topology for clock routing trees.

Results for Building-Block Designs

The CLOCK2 heuristic was tested on random clock nets of sizes 4, 8, and 16 sinks, using random layouts that contained 16 or 32 blocks. Layouts were

	MMM cost			GR+E+H cost		
$\|S\|$	Min	Ave	Max	Min	Ave	Max
4	656	1197	1823	555	1125	1668
8	1089	2136	2943	1123	1979	2810
16	2841	3506	4221	2793	3322	3993
32	4813	5598	6216	4695	5273	5866
64	7624	8377	9266	7372	7982	8556
128	11439	12276	13136	11052	11697	12243
256	17220	17874	18549	16379	16955	17543
512	25093	25666	26291	23866	24465	25325
1024	36126	36765	37561	34231	34965	36179

Table 4.5 Minimum, average and maximum tree cost for MMM and GR+E+H.

	MMM skew			GR+E+H skew		
$\|S\|$	Min	Ave	Max	Min	Ave	Max
4	2	112.31	379	0	0.00	0
8	46	186.10	407	0	0.00	0
16	86	234.72	416	0	0.00	0
32	118	262.61	540	0	0.00	0
64	141	229.15	337	0	0.00	0
128	120	201.55	282	0	1.02	30
256	127	183.28	250	0	0.92	46
512	103	153.90	203	0	0.62	31
1024	94	125.34	167	0	0.08	4

Table 4.6 Minimum, average and maximum pathlength skew for MMM and GR+E+H.

generated by creating the prescribed number of non-overlapping blocks with length, width, and lower-left x- and y-coordinates all chosen from a uniform distribution over the interval $[1, L]$ with $L = 1000$.

For each combination of net size and block cardinality, 100 instances were tested; Table 4.7 compares pathlength skew and tree cost of T_{CLOCK2} against the output of the KMB heuristic [159] for the Steiner problem in weighted

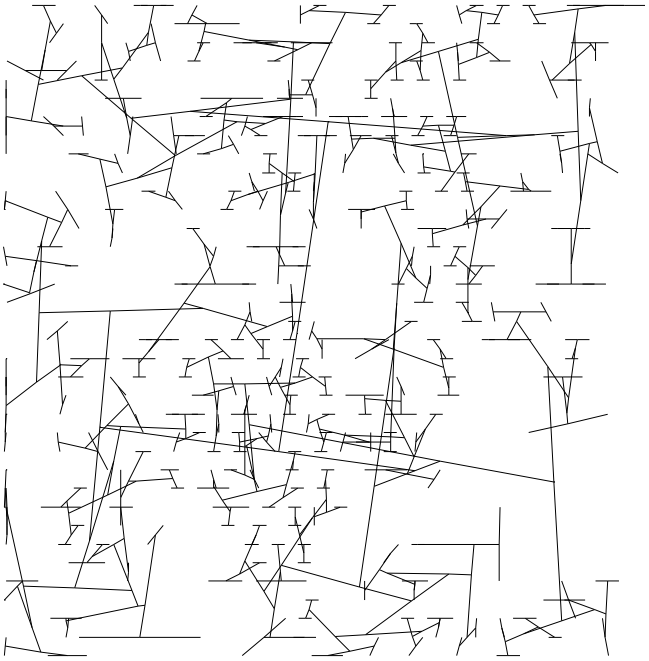


Figure 4.9 Output of variant GR+E+H on the Primary2 layout.

graphs (see Section 2.7). The average CLOCK2 pathlength skew is near zero, and is never more than 2% of the pathlength skew in the heuristic Steiner minimal tree. This skew reduction comes at the expense of between 24% and 77% increase in tree cost, versus the heuristic SMT. All data in the table are given in grid units.[3]

Remarks

In retrospect, the matching-based construction of pathlength-balanced trees remains interesting not for its skew-minimization properties, but rather for

[3][60] notes that HSPICE simulations confirm the low skew of the CLOCK2 construction for building-block layouts. Also, the "average density" in any routing channel, computed as the average of non-zero local column densities over all columns in all channels, is close to 1. Thus, although up to $\log n$ paths can possibly overlap in a given channel, such overlaps seem to occur only rarely.

		Pathlength skew		Tree cost	
# blocks	$\|S\|$	KMB	CLOCK2	KMB	CLOCK2
16	4	511.0	0.8	1537	1921
16	8	794.9	12.9	2328	3478
16	16	1101.5	22.1	3332	5873
32	4	445.0	0.4	1401	1729
32	8	804.4	4.4	2261	3407
32	16	1136.9	12.0	3357	5847

Table 4.7 Average tree costs and pathlength skews, in grid units, for both the KMB heuristic Steiner minimal tree and the CLOCK2 output tree. Each value is an average over 100 random instances in the 1000×1000 grid.

the directions it leaves open for subsequent work. The original discussion in [60, 144] stated the following "extensions":

1. **Toward Exact Zero Skew:** Instead of the linear delay model, the matching-based approach could use the more accurate Elmore delay model to select balance points (CEPs). The matching construction could also incorporate varying load capacitances and other design constraints which are ignored by the linear delay model.

2. **Loci of Balance Points:** In the Manhattan metric, the "balance point" of a wire connecting two terminals is not unique but is rather a locus of many possible locations (Figure 4.10), with the extremes corresponding to the two L-shaped wire orientations. The simulations above set the balance point of an edge to be its "Euclidean" midpoint, but there is no methodological justification for this.

3. **Lookahead and Deferral:** At each level of the matching construction, it is possible to use *lookahead* of one or more levels. For example, if pathlength skew cannot be eliminated by H-flipping, we could "go back" down one or two levels, and attempt the alternate "H" configurations within these subtrees. More generally, "lookahead" is simply a way of *deferring* commitment to specific elements of either the topology or the geometric embedding until more reasoned choices can be made.

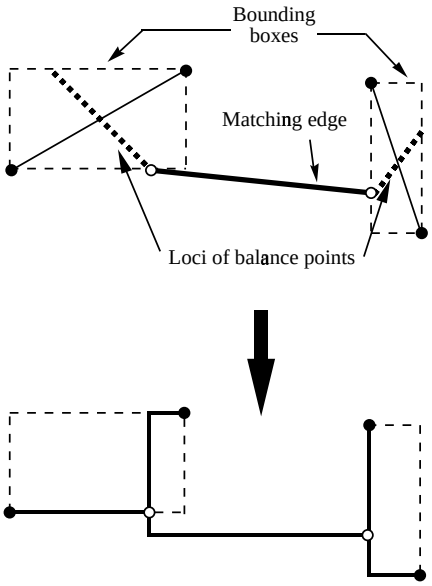


Figure 4.10 Further optimizations are possible by matching over the *loci* of balance point candidates.

For all practical purposes, the work of Tsay [240], solved the question of achieving exact zero Elmore delay skew. Approaches using "deferral" were proposed by Li and Jabry [172] and by Edahiro [84], in the sense that each of these works uses only a partial greedy matching over an existing set of CEPs to further the bottom-up generation of the clock tree. We now describe the "Deferred-Merge Embedding" (DME) approach [29, 44, 82], which combines the notions of (i) balance point loci, and (ii) deferred embedding of the topology.

4.3 DME: EXACT ZERO SKEW WITH MINIMUM WIRELENGTH

The geometric matching approach addresses skew minimization only with respect to linear delay, and does not guarantee a zero-skew solution. Tsay [240] provided a major advance via a method that guarantees exact zero skew accord-

ing to Elmore delay. [4] Tsay's algorithm combines pairs of zero-skew subtrees at "tapping points" (analogous to the "balance points" in CLOCK1) to yield larger zero-skew subtrees, with additional wire introduced as needed to maintain the exact zero-skew property. The method is efficient due to the linear-time evaluation of Elmore delay at all leaves of a given tree.

Both the top-down method of [143] and the bottom-up methods of [60, 144, 240] center on computing a clock tree topology, and leave unaddressed the minimum-cost embedding of the topology. In general, these methods fix the embedding of each internal node of the topology as soon as the node is defined [144], or with just one level of lookahead in the tree construction [143, 240]. However, as was demonstrated by "H-flipping" in the CLOCK1 algorithm, the ability to undo or "defer" embedding choices can lead to substantial cost reductions. Certainly, both skew *and cost* must be considered in a successful clock routing scheme.

This section describes the *Deferred-Merge Embedding* (DME) algorithm, which for any given topology substantially reduces the tree cost while guaranteeing exact zero skew, i.e., a ZST solution. DME was discovered independently by three groups – Boese et al. [29], Chao et al.[44], and Edahiro [82] – with the earliest of these being Edahiro.[5]

Given a set of sink locations S and topology G, the DME algorithm embeds the internal nodes of G in two phases, with the main precept being to defer the embedding of each node for as long as possible. First, a bottom-up phase constructs a tree of line segments, with each line segment being the locus of possible placements for some $v \in G$ within an optimal ZST. Once this bottom-up phase has determined the loci of possible placements for two siblings in G, the corresponding locus for their parent (i.e., the "merging segment") can then be determined. Second, a top-down phase resolves the exact locations of these internal nodes of the clock tree.

In the linear delay regime, DME produces an *optimal* (i.e., minimum-cost) ZST with respect to the prescribed topology [29], and this tree will also have optimal

[4] "Exact zero skew" is of course a somewhat redundant notion. However, since the publication of [240] the phrase has permeated the clock routing literature, where it connotes "guaranteed zero skew". Our discussion uses the phrase in this accepted sense.

[5] While the work of Edahiro [82] was clearly earliest, its existence was not realized by the other two groups. Chao, Hsu and Ho [44] applied DME to the Elmore delay model and also proposed the "balanced-bipartition" (BB) technique to generate an underlying clock tree topology. Boese and Kahng [29] treat both the Elmore and linear models, and establish many of the theoretical results for DME (see also the results for linear delay in [82]), as well as counterexamples to Elmore-delay optimality of DME (cf. [44]).

source-sink delay (i.e., minimum radius). In the Elmore delay regime, DME is also effective but does not guarantee that the output ZST is optimal for its given topology. Since DME must be given a prescribed topology G, the question of generating the best input topology for DME has become an active area of research. The method can be extended to prescribed-skew formulations, as well as more general routing optimizations (e.g., DME-like approaches are promising for global routing with upper and lower bounds on sink delays). Furthermore, by generalizing the concept of a placement locus from a merging segment to a merging area, DME can also address a *bounded-skew routing tree* formulation.

4.3.1 Bottom-Up Phase: The Tree of Merging Segments

For prescribed sink locations S and connection topology G, DME constructs a *tree of merging segments*. The merging segment of a node $v \in G$ represents the set of placements of v that are compatible with an optimal ZST solution. A merging segment will always be a line segment tilted at 45 degrees from the coordinate axes (recall Figure 4.10). It is possible for a merging segment to have zero length, i.e., be a single point. The merging segment of a node depends on the merging segments of its two children, so G must be processed in bottom-up order. If node v has children a and b, then edges e_a, e_b in the topology are assigned the minimum possible lengths such that it is possible to balance all the sink delays in the merged subtree rooted at v. These lengths must be enforced by the top-down embedding phase, when the ZST is created. We now develop more precisely the construction of the tree of merging segments.

In our discussion, the distance between two points p and q is the Manhattan distance $d(p, q)$, and the distance between two sets of points P and Q, written $d(P, Q)$, is $min\{d(p, q) \mid p \in P \text{ and } q \in Q\}$. Let a and b be the children of node v in G. We use TS_a and TS_b to denote the subtrees of merging segments rooted at a and b, respectively, and we seek placements of v which allow TS_a and TS_b to be merged with *minimum* added wire while preserving zero skew. Define the *merging cost* between TS_a and TS_b to be $|e_a| + |e_b|$, where $|e_a|$ and $|e_b|$ denote the lengths to be assigned to edges e_a and e_b (recall that e_a is the edge from node a to its parent). These lengths are chosen to minimize merging cost while balancing delays at $pl(v)$. (There is a unique optimal solution for $|e_a|$ and $|e_b|$ as long as delay is a monotone increasing function of wirelength.)

A *Manhattan arc* is a line segment, possibly of zero length, with slope +1 or -1. (It will turn out that all merging segments are Manhattan arcs.) The collection

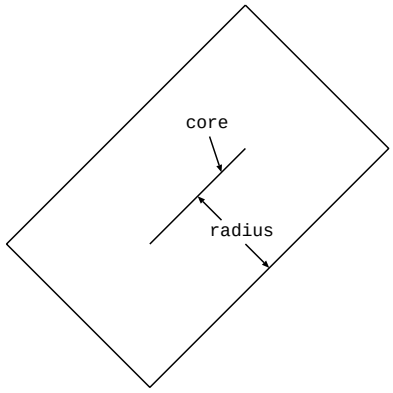


Figure 4.11 A TRR with core and radius as indicated.

of points within a given distance of a Manhattan arc is a *tilted rectangular region*, or *TRR*, whose boundary is composed of Manhattan arcs (see Figure 4.11). The Manhattan arc at the center of the TRR is called its *core*; the *radius* of a TRR is the distance between its core and its boundary. The concept of a TRR will be used to construct the tree of merging segments, and to determine embedding points in the top-down phase.

A formal recursive definition of the *merging segment* of node v, $ms(v)$, is as follows. If v is a sink s_i, then $ms(v) = \{s_i\}$. If v is an internal node of G with children a and b, then $ms(v)$ is the set of all placements $pl(v)$ which allow minimum merging cost. That is to say, $ms(v)$ is the set of all points within distance $|e_a|$ of $ms(a)$ and within distance $|e_b|$ of $ms(b)$, where $|e_a|$ and $|e_b|$ are as small as possible while still balancing source-sink delays. If $ms(a)$ and $ms(b)$ are both Manhattan arcs, then $ms(v)$ is simply the intersection of two TRRs, trr_a with core $ms(a)$ and radius $|e_a|$, and trr_b with core $ms(b)$ and radius $|e_b|$; i.e., $ms(v) = trr_a \cap trr_b$ (see Figure 4.12).

The merging cost at v is at least equal to $\kappa = d(ms(a), ms(b))$. If the merging cost is greater than κ, i.e., more wirelength is needed to balance the delays, then one edge length will equal zero and the other will equal the merging cost. Figure 4.12(a) illustrates the algorithm for the case where the merging cost is equal to κ, and Figure 4.12(b) illustrates the case where the merging cost is greater than κ. An entire tree of merging segments is shown in Figure 4.13; the

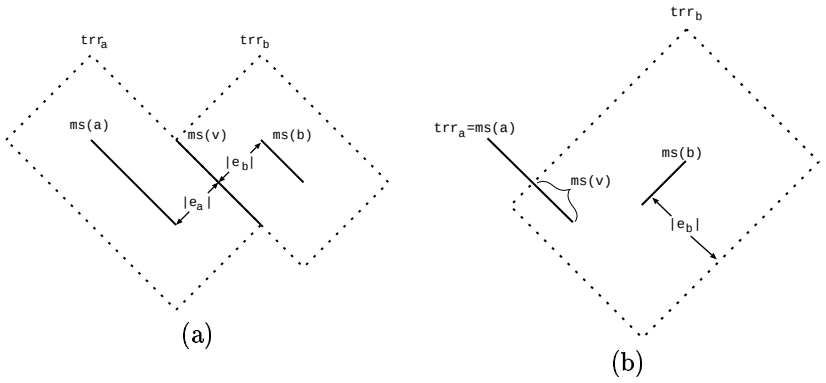


Figure 4.12 Two cases in construction of merging segment $ms(v)$. (a) Merging cost equals $\kappa = d(ms(a), ms(b))$. (b) Merging cost is greater than κ (note that in this example $radius(trr_a) = |e_a| = 0$).

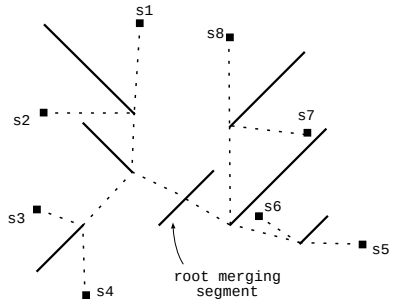


Figure 4.13 An example of a tree of merging segments with sinks $s_1, \ldots, s_8$. Solid lines are merging segments; dotted lines are edges between merging segments.

leaves of the tree of segments are all single points representing the sink locations $s_1, \ldots, s_8$, and the internal nodes (solid line segments) are Manhattan arcs.

Lemma 4.3.103 *Given two TRRs R_1 and R_2, their intersection is also a TRR and can be found in constant time. If R_1 and R_2 satisfy $radius(R_1) + radius(R_2)$ $= d(core(R_1), core(R_2))$, then $R_1 \cap R_2$ is a Manhattan arc.*

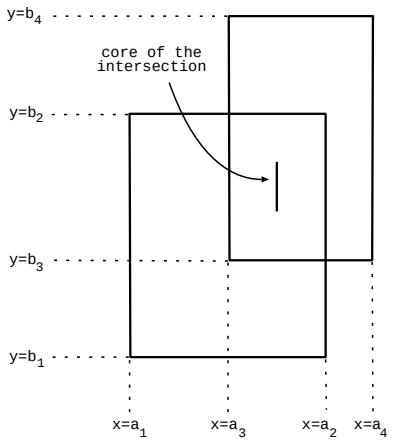


Figure 4.14 Intersecting two TRRs after 45-degree rotation.

Proof: Rotating the plane by 45 degrees, so that each TRR has its boundary segments parallel to the coordinate axes (see Figure 4.14), requires constant time. The intersection of the rotated TRR's will be either rectangular or empty, and can be found using a simple constant-time case analysis. Applying the inverse rotation to the intersection yields the TRR $R_1 \cap R_2$ and the first part of the claim.

If $radius(R_1) + radius(R_2) = d(core(R_1), core(R_2))$, then decreasing either radius will cause $R_1 \cap R_2 = \emptyset$. Hence, $R_1 \cap R_2$ must have zero width and be either a line segment or a single point. Since $R_1 \cap R_2$ is also a TRR, it must be a Manhattan arc. □

Lemma 4.3.103 can be used to show that all merging segments are Manhattan arcs. First, we show that for any node $v \in G$ with children a and b, if $ms(a)$ and $ms(b)$ are both Manhattan arcs, then $ms(v)$ is a Manhattan arc. (Case 1) If the merging cost at v is equal to κ, we have $|e_a| + |e_b| = d(core(trr_a), core(trr_b))$; by definition, $|e_a| = radius(trr_a)$ and $|e_b| = radius(trr_b)$. According to the lemma, $d(core(trr_a), core(trr_b)) = radius(trr_a) + radius(trr_b)$ means that

$trr_a \cap trr_b$ is a Manhattan arc. (Case 2) If the merging cost is greater than κ, either trr_a or trr_b will be a Manhattan arc whose intersection with any convex set (e.g., another TRR) will also be a Manhattan arc. Finally, we note that for each sink s_i the merging segment $ms(s_i)$ is a single point, which is also a Manhattan arc. Inductively, all merging segments must be Manhattan arcs.

Procedure Build_Tree_of_Segments
Input: Topology G; set of sink locations S
Output: Tree of merging segments TS containing $ms(v)$ for each node v in G, and edge length $|e_v|$ for each $v \neq s_0$

For each node v in G (bottom-up order)
  If v is a sink node **Then**
    $ms(v) = \{pl(v)\}$
  Else
    Let a and b be the children of v
    Calculate_Edge_Lengths($|e_a|$,$|e_b|$)
    Create TRRs trr_a and trr_b as follows:
      $core(trr_a) = ms(a)$
      $radius(trr_a) = |e_a|$
      $core(trr_b) = ms(b)$
      $radius(trr_b) = |e_b|$
    $ms(v) = trr_a \cap trr_b$

Figure 4.15 Construction of the tree of merging segments.

Figure 4.15 describes the procedure Build_Tree_of_Segments, which constructs the tree of merging segments. The Calculate_Edge_Lengths subroutine depends on the delay model, and is described below for the separate cases of linear and Elmore delay. By Lemma 4.3.103, Build_Tree_of_Segments requires constant time to compute each new merging segment, and time linear in the size of S to construct the entire tree of merging segments.

4.3.2 Top-Down Phase: Embedding of Nodes

Once the tree of segments has been constructed, the exact embeddings of internal nodes in the ZST are chosen in a top-down manner. Initially, for the

root s_0 any point in $ms(s_0)$ can be chosen as $pl(s_0)$.[6] After node v's parent has been embedded, v can be embedded anywhere on $ms(v)$, as long as the distance $d(pl(v), pl(p))$ is not greater than $|e_v|$. Thus, we create a square TRR trr_p with core $\{pl(p)\}$ and radius $|e_v|$; node v can be placed anywhere in the intersection $ms(v) \cap trr_p$ (see Figure 4.16).

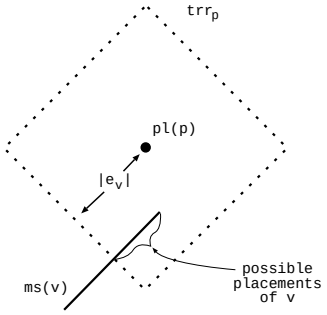


Figure 4.16 Procedure Find_Exact_Placements: finding the placement of v given the placement of its parent p.

Because $ms(p)$ was constructed such that $d(ms(v), ms(p)) \leq |e_v|$, the intersection $ms(v) \cap trr_p$ must be nonempty. For the tree of merging segments shown in Figure 4.13, the resulting placements are indicated by the points at which the segments are connected by dotted lines. Figure 4.17 describes the procedure Find_Exact_Placements, which uses the tree of merging segments to determine the final embedding of nodes in the ZST.

Since each instruction in Find_Exact_Placements is executed at most once for each node in G (and the intersection of TRRs $ms(v)$ and trr_p can be found in constant time, by Lemma 4.3.103), Find_Exact_Placements runs in $O(|S|)$ time. Procedure Build_Tree_of_Segments also runs in linear time, and hence DME is a linear-time algorithm.

[6] If a fixed source location s_0' is specified, we choose $pl(s_0) \in ms(s_0)$ with minimum distance from s_0' and connect a wire directly from s_0' to $pl(s_0)$.

Procedure Find_Exact_Placements
Input: Tree of segments TS containing $ms(v)$ and $\|e_v\|$ for each node v in G **Output:** ZST $T(S)$

For each internal node v in G (top-down order)
  If v is the root **Then**
  Choose any $pl(v) \in ms(v)$
  Else
  Let p be the parent node of v
  Construct trr_p as follows:
  $core(trr_p) = \{pl(p)\}$
  $radius(trr_p) = \|e_v\|$
  Choose any $pl(v) \in ms(v) \cap trr_p$

Figure 4.17 Construction of the ZST by embedding internal nodes of the topology.

4.3.3 Application of DME to Linear Delay

Calculating Edge Lengths

Calculating the edge lengths $|e_a|$ and $|e_b|$ is straightforward in the linear delay model. Let a and b be children of v with merging segments $ms(a)$ and $ms(b)$, and let $t_{LD}(a)$ and $t_{LD}(b)$ be the delays from a and b to the sinks in their respective subtrees. Then, zero skew at v requires that

$$t_{LD}(a) + |e_a| = t_{LD}(b) + |e_b|.$$

Again, let $\kappa = d(ms(a), ms(b))$. If $|t_{LD}(a) - t_{LD}(b)| \leq \kappa$, then the merging cost is minimized with $|e_a| + |e_b| = \kappa$, i.e.,

$$|e_a| = \frac{\kappa + t_{LD}(b) - t_{LD}(a)}{2}$$

and

$$|e_b| = \kappa - |e_a|$$

On the other hand, if $|t_{LD}(a) - t_{LD}(b)| > \kappa$, then the merging cost is minimized when one of the edge lengths is equal to zero. When $t_{LD}(a) > t_{LD}(b)$, we have

$|e_a| = 0$ and $|e_b| = t_{LD}(a) - t_{LD}(b)$; similarly, when $t_{LD}(a) < t_{LD}(b)$ we have $|e_a| = t_{LD}(b) - t_{LD}(a)$ and $|e_b| = 0$.

Optimality of DME for Linear Delay

In this section, we show two optimality results for the DME algorithm under the linear delay model. Our discussion will use the term *Manhattan disk* to denote the special case of a TRR whose core consists of a single point. In other words, a Manhattan disk is the set of all points within a given radius of a central point. In the Manhattan plane, such a "disk" is actually diamond-shaped (recall trr_p in Figure 4.16). Let $MD(s_i, r)$ denote the Manhattan disk with core $\{s_i\}$ and radius $r \geq 0$. The *diameter* of S is defined to be $max\{d(s_i, s_j) \mid s_i, s_j \in S\}$.

We first show that under the linear model, DME minimizes the source-sink delay in a ZST. Specifically, for any input topology DME constructs a ZST with delay equal to one-half the diameter of the sink set S, which is the minimum feasible radius for any tree connecting S. This result has also been shown by Edahiro [82, 83].

Lemma 4.3.104 *: Let d be the diameter of sink set S. Then*

$$\cap_{s_i \in S}[MD(s_i, d/2)] \neq \emptyset.$$

Proof: After a 45 degree rotation (and dilation), the Manhattan metric becomes equivalent to the L_∞ metric, where $d[(x, y), (x', y')] = max\{|x - x'|, |y - y'|\}$. Hence we need only prove the lemma for the L_∞ metric, where TRRs are equivalent to rectangles with vertical and horizontal boundaries. Consider the smallest rectangle R with vertical and horizontal boundary lines that contains all points in S (after rotation). Let d be the diameter of S. Then both the width and height of R must be less than or equal to d (otherwise there would be two sinks s_i and s_j with $d(s_i, s_j) > d$). Consequently, the point at the center of R is within distance $d/2$ of all sinks in S, and is contained in $\bigcap_{s_i \in S}[MD(s_i, d/2)]$. □

This shows the feasibility of constructing a ZST over S having linear source-sink delay equal to one-half the diameter of S. The next lemma states that increasing the radii of two TRRs by a constant δ will increase the radius of their intersection by δ but leave the core of the intersection unchanged. This

is obvious when the TRRs are rotated by 45 degrees as in the proof of Lemma 4.3.103 (see Figure 4.14).

Lemma 4.3.105 *: Let A and B be TRRs, and suppose $A \cap B = C \neq \emptyset$. Construct TRRs A' and B' such that for $\delta \geq 0$, $core(A') = core(A)$, $radius(A') = radius(A) + \delta$, $core(B') = core(B)$, and $radius(B') = radius(B) + \delta$. If $C' = A' \cap B'$, then $core(C') = core(C)$ and $radius(C') = radius(C) + \delta$.* □

Theorem 4.3.106 *: For any sink set S and topology G, the DME algorithm will return a ZST with minimum feasible source-sink delay under the linear model, equal to one-half the diameter of S.*

Proof: Let d equal the diameter of S. We assign a TRR, called $TRR(v)$, to each node $v \in G$ such that (i) if v is a sink, then $TRR(v) = MD(pl(v), d/2)$; and (ii) if v is an internal node with children a and b, then $TRR(v) = TRR(a) \cap TRR(b)$.

By Lemma 4.3.104, $TRR(s_0) = \cap_{s_i \in S}[MD(s_i, d/2)]$ is non-empty. Let s_j and s_k be two sinks in S with $d(s_j, s_k) = d$. The intersection of $TRR(s_j) = MS(s_j, d/2)$ and $TRR(s_k) = MS(s_k, d/2)$ must have radius $= 0$ (by Lemma 4.3.103), and so $TRR(s_0)$ must have radius $= 0$.

For any node v, let $t_{LD}(v)$ be the linear delay from v to each of the sinks in the subtree rooted at v in the DME output.

Fact 4.3.107 *For each node v in G, $core(TRR(v)) = ms(v)$ and $radius(TRR(v)) = d/2 - t_{LD}(v)$.*

Proof of Fact: We apply induction on the maximum number of edges between v and sinks in its subtree. If v is a sink, then $core(TRR(v)) = \{v\} = ms(v)$ and $radius(TRR(v)) = d/2 = d/2 - t_{LD}(v)$. If v is an internal node with children a and b, inductively assume that the Fact holds for a and b. In the linear delay model, we have $t_{LD}(a) = t_{LD}(v) - |e_a|$, implying

$$\begin{aligned} radius(TRR(a)) &= d/2 - t_{LD}(a) \\ &= d/2 - t_{LD}(v) + |e_a| \end{aligned}$$

and similarly $radius(TRR(b)) = d/2 - t_{LD}(v) + |e_b|$.

The TRRs trr_a and trr_b constructed by Build_Tree_of_Segments will have $core(trr_a) = ms(a)$ and $radius(trr_a) = |e_a|$, and $core(trr_b) = ms(b)$ and $radius(trr_b) = |e_b|$. This implies that

$$\begin{aligned} radius(TRR(a)) &= d/2 - t_{LD}(v) + radius(trr_a) \\ radius(TRR(b)) &= d/2 - t_{LD}(v) + radius(trr_b) \end{aligned}$$

whence $TRR(a)$ and $TRR(b)$ can be constructed from trr_a and trr_b, respectively, by adding the constant $d/2 - t_{LD}(v)$ to their radii. Lemma 4.3.105 then implies that $core(TRR(v)) = ms(v)$ and $radius(TRR(v)) = d/2 - t_{LD}(v)$. This proves the Fact.

Since $radius(TRR(s_0)) = 0$, we have $t_{LD}(s_0) = d/2$, proving the theorem. □

In the linear delay regime, DME also has optimal tree cost. The following lemma directly implies this result.

Lemma 4.3.108 *Suppose that ZST T has minimum wirelength for sink locations S and topology G. Let v be a node in G with children a and b. Also, let L_a denote the edge length assigned to e_a by DME and $L(T, e_a)$ denote the length of edge e_a in T. Then (i) $pl(T, v) \in ms(v)$ and (ii) $L(T, e_a) = L_a$.*

Proof: (See Figure 4.18.) The proof is by contradiction. Suppose that T has minimum wirelength for S and topology G and that either (i) or (ii) does not hold for some v in G. Let v be the node at the lowest level of G for which either (i) or (ii) is violated, i.e., the subtrees of T rooted at v's children a and b can be constructed by DME. We will first construct a tree T_{new} with source at $q = pl(T, v)$, and then construct a ZST T' by replacing the subtree of T rooted at v with part of T_{new}. We will have $pl(T', v) = q'$ as in Figure 4.18, and using Theorem 4.3.106 will show that $cost(T') < cost(T)$ if either (i) or (ii) are violated in T.

Let G_v be the subtree of topology G rooted at v, and let S_v be the set of sinks in G_v. Suppose that sink s_i is the sink in S_v furthest from q. Create a new sink z that is located at a point directly opposite of q from s_i; i.e., $d(q, s_i) = d(q, z)$ and $d(s_i, z) = 2 \cdot d(q, s_i)$. Consider the new set of sinks $S_{new} = S_v \cup \{z\}$. We create a topology G_{new} for S_{new} that merges G_v and z at its root, s_{new0}. We

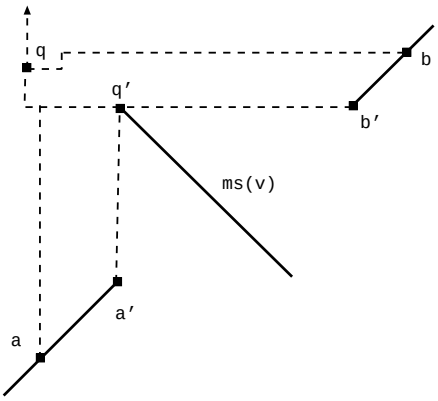


Figure 4.18 Optimal placement of siblings a and b must satisfy the distance constraint in the top-down phase Find_Exact_Placements. Here, $pl(T, a) = a$ and $pl(T', a) = a'$, etc.; and $cost(T') < cost(T)$. In the example shown, changing the placements of nodes a and b to locations a' and b' allows the a'–q and b'–q connections to share wire on the segment from q' to q. The delay at point q remains unchanged.

then run DME on S_{new} using topology G_{new} to create ZST T_{new}. By Theorem 4.3.106, T_{new} will have the minimum feasible delay at each sink, which is equal to one-half the diameter of S_{new}, i.e., $d(q, s_i)$.

By Fact 4.3.107, $ms(s_{new0})$ is the set of all points within distance $d(q, s_i)$ of every sink in S_{new}. Therefore, $q \in ms(s_{new0})$ and T_{new} can be constructed so that $q = pl(T_{new}, s_{new0})$. Let $a' = pl(T_{new}, a)$, $b' = pl(T_{new}, b)$, and $q' = pl(T_{new}, v)$. We now construct ZST T' for S by cutting off the subtree of T rooted at q and replacing it with T_{new} minus the edge between q and z. Since $t_{LD}(T', q) = d(q, s_i)$, it must be that $t_{LD}(T', q) \leq t_{LD}(T, q)$. If the strict inequality holds, we add extra wire between q and q' to enforce equality, and thereby retain zero skew.

For convenience, let us use $e_{a'}$ and $e_{b'}$ to represent the embeddings of edges e_a and e_b in T'. We also use $e_{q'}$ to denote the partial edge between q' and q in T'. Because the subtrees of T rooted at a and b were constructed according to DME, we have $t_{LD}(T, a) = t_{LD}(T', a')$ and $t_{LD}(T, b) = t_{LD}(T', b')$. Let $|e_a|$

and $|e_b|$ respectively represent the lengths of e_a and e_b in T. Then, because $t_{LD}(T, q) = t_{LD}(T', q)$

$$|e_a| = |e_{a'}| + |e_{q'}| \quad \text{and} \quad |e_b| = |e_{b'}| + |e_{q'}|.$$

Because the subtrees of a in T and T' can both be constructed using DME, they must have the same cost. Similarly, the subtrees of b in T and T' have the same cost. Consequently, the costs of T and T' differ only in edges e_a, e_b, and $e_{q'}$, i.e.,

$$cost(T) - cost(T') = |e_a| + |e_b| - (|e_{a'}| + |e_{b'}| + |e_{q'}|) = |e_{q'}|.$$

If T is optimal, then $|e_{q'}| = 0$ and hence (i) $q \in ms(v)$ and (ii) $L(T, e_a) = |e_a| = |e_{a'}| = L_a$. □

The DME optimality result follows directly from Lemma 4.3.108, because DME places only two constraints on the placement of a node v in G: (i) $pl(v) \in ms(v)$ and (ii) $d(pl(v), pl(p)) \leq L_v$, where p is the parent of v and L_v is the edge length assigned by DME to e_v.

Theorem 4.3.109 *Given a set of sink locations S and a connection topology G, the DME algorithm produces a ZST T with minimum cost over all ZSTs for S having topology G.* □

We note that the DME output also has optimal cost for any given topology when the source location is predefined (cf. the construction described in the previous footnote). Any tree rooted at a location $q \notin ms(s_0)$ will have minimum cost only if the two subtrees of G directly below the root are merged at a point $q' \in ms(s_0)$ which is then connected to s_0' by a single edge.

4.3.4 Application to Elmore Delay

Calculating Edge Lengths in the Elmore Delay Model

In the following, $\bar{r}$ and $\bar{c}$ again denote the resistance and capacitance per unit length of interconnect. We let T_v denote the subtree of $T(S)$ rooted at v, and let

c_v denote the node capacitance of v. We assume $c_v = 0$ for each non-sink node in all of our examples and test cases; however, each sink s_i can have loading capacitance dependent on the design of the corresponding functional unit. To calculate the edge lengths needed to merge two trees of merging segments TS_a and TS_b with minimum merging cost in the Elmore model, we use the analysis of Tsay [240], which we now review.

Let TS_a and TS_b have delays $t_1 = t_{ED}(a)$ and $t_2 = t_{ED}(b)$, and capacitances C_1 and C_2, respectively (we know the capacitance values from the edge lengths and sink capacitances in subtrees T_a and T_b). Let $pl(v)$ be a merging point with minimum merging cost. From the definition of Elmore delay, we have $t_{ED}(v,a) = r_{e_a}(\frac{1}{2}c_{e_a} + C_1)$. Thus, $pl(v)$ satisfies:

$$r_{e_a}(\frac{1}{2}c_{e_a} + C_1) + t_1 = r_{e_b}(\frac{1}{2}c_{e_b} + C_2) + t_2 \tag{4.1}$$

Let $d(ms(a), ms(b)) = \kappa$. Suppose that TS_a and TS_b can be merged with merging cost κ; in other words, $|e_a| = x$ and $|e_b| = \kappa - x$ for $0 \leq x \leq \kappa$. Then we have resistances $r_{e_a} = \bar{r}x$ and $r_{e_b} = \bar{r}(\kappa - x)$ and capacitances $c_{e_a} = \bar{c}x$ and $c_{e_b} = \bar{c}(\kappa - x)$. Substituting into (1) and solving for x yields:

$$x = \frac{t_2 - t_1 + \bar{r}\kappa(C_2 + \frac{1}{2}\bar{c}\kappa)}{\bar{r}(C_1 + C_2 + \bar{c}\kappa)} \tag{4.2}$$

Case 1: If $0 \leq x \leq \kappa$, then there exists a feasible zero skew merging point of TS_a and TS_b with merging cost κ, $|e_a| = x$ and $|e_b| = \kappa - x$.

Case 2: If $x < 0$ or $x > \kappa$, then the assumption of merging cost κ results in a negative edge length for either e_a or e_b. In this case, an extended distance $\kappa' > \kappa$ is required to balance the delays of the two trees. If $x < 0$, which means $t_1 > t_2$, we choose $pl(a)$ as the merging point and set $|e_a| = 0$ and $|e_b| = \kappa'$. Then:

$$t_1 = \bar{r}\kappa'(\frac{1}{2}\bar{c}\kappa' + C_2) + t_2 \tag{4.3}$$

and we use the quadratic formula to solve for κ':

$$\kappa' = \frac{((\bar{r}C_2)^2 + 2\bar{r}\bar{c}(t_1 - t_2))^{\frac{1}{2}} - \bar{r}C_2}{\bar{r}\bar{c}} \tag{4.4}$$

Similarly, if $x > \kappa$, we set $|e_b| = 0$ and

$$|e_a| = \kappa' = \frac{((\bar{r}C_1)^2 + 2\bar{r}\bar{c}(t_2 - t_1))^{\frac{1}{2}} - \bar{r}C_1}{\bar{r}\bar{c}} \tag{4.5}$$

The above analysis shows that a zero skew merging point between two ZSTs can always be found. The merging cost depends on the distance between the two roots of the ZSTs, the delay of each ZST, and the tree capacitance of each ZST. However, the DME algorithm is not optimal for all topologies under the Elmore delay approximation.

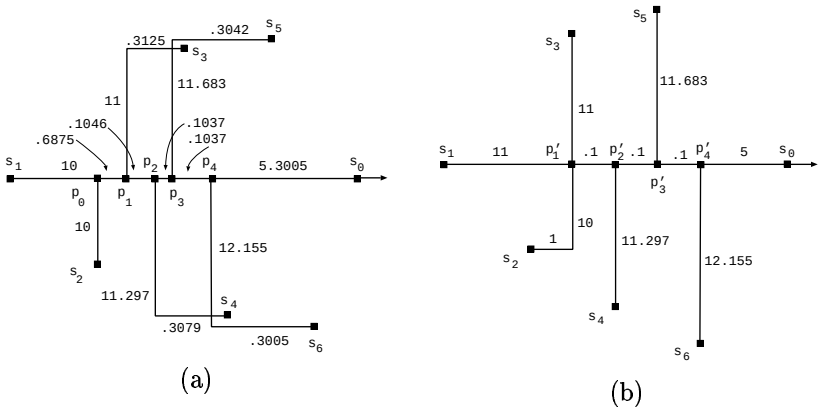


Figure 4.19 (a) ZST T which is constructed by the DME algorithm, and which has sub-optimal cost for the given topology. (b) ZST T' which has optimal cost for the topology in (a), but which violates the DME algorithm. In T', the internal nodes placed at p_0 and p_1 in T are placed at the same point, p'_1. (Trees are not drawn to scale; lengths of horizontal and vertical segments are as indicated.)

Suboptimality of DME for Elmore Delay

The ZSTs T and T' in Figure 4.19 show that DME will not always give a minimum-cost zero-skew embedding under the Elmore delay model. T and T' connect terminal points $s_1, ..., s_6$ to source s_0. Both trees are assumed to extend to the right side of s_0, with their subtrees on the right of s_0 being mirror images of the subtrees to the left of s_0; this ensures that the source will be at s_0 in the optimal tree). We normalize both the unit resistance $\bar{r}$ and unit capacitance $\bar{c}$ to one, and assume without loss of generality that the loading capacitance of each sink is zero.[7]

The ZST T' was constructed so that if points s_1 and s_2 are merged at point p'_1, then vertical wires from points s_3 through s_6 will merge along the horizontal wire from s_1 to s_0 with exactly zero skew. If, however, s_1 and s_2 are merged on their merging segment as in tree T, the delay at p'_1 will increase, and jogs will be required in the edges e_{s_3} through e_{s_6}. In this example, the four required jogs are each of length greater than 0.3. Thus, their sum is greater than 1, which was the amount of wire saved initially by merging s_1 and s_2 at p_0. Table 4.8 contains the calculated delay and capacitance at each of the internal nodes of T and T'. For example, in T' the capacitance at p'_1, $C_{p'_1}$, is 33; and the delay at node p'_2 is

$$t_{ED}(p'_2) = t_{ED}(p'_1) + 0.1 * (\frac{0.1}{2} + C_{p'_1}) = 60.5 + 3.305 = 63.8 = \frac{(11.297)^2}{2}$$

Because the unit resistance and capacitance are both equal to one, and because the loading capacitances at the leaves are zero, the tree capacitance of each node equals the amount of wire in its subtree. Thus, we see from Table 4.8 that $cost(T) - cost(T') \approx 0.44$. It should be noted that Chou and Cheng [54, 50] have recently demonstrated the cost suboptimality of DME in the octolinear and Euclidean geometries.

4.3.5 Experimental Results and Discussion

The DME algorithm has been implemented in C on Sun SPARC-1 workstations. To distinguish the effects of DME from the effects of various heuristics

[7] The example can be easily altered to have non-zero sink loading capacitances: shorten the edge adjacent to a given sink s_i by a small value $c_i > 0$, and then set the loading capacitance of the sink to c_i.

Tree T			Tree T'		
node	delay	capacitance	node	delay	capacitance
p_0	50	20			
p_1	64.0	32.0	p'_1	60.5	33.0
p_2	67.3	43.7	p'_2	63.8	44.4
p_3	71.9	55.8	p'_3	68.2	56.2
p_4	77.6	68.4	p'_4	73.9	68.4
s_0	454.0	2×73.66	s_0	428.6	2×73.44

Table 4.8 Delay and capacitance at each internal node in ZSTs T and T'.

for generation of clock tree topologies, we have applied DME to each of several previous constructions in the literature: the MMM method of [143], the KCR method of [60, 144], the method of Tsay [240], and the BB ("balanced bipartition") method of [44, 45]. These comparisons have been made for both the linear and Elmore delay models. Two sets of test cases were used: (i) the layouts of Primary1 and Primary2 studied in [143] and provided by Jackson et al. [226]; and (ii) the sink placements for circuits r1 - r5 studied by Tsay [240]. These seven test cases, which have sizes ranging from 267 to 3101 sinks, have emerged as a *de facto* benchmark suite for clock tree constructions in the recent literature.

Results for the Linear Delay Model

Experimental results for linear delay are shown in Table 4.9. The cost reduction afforded by DME can be substantial, e.g., KCR+DME averages more than 9% cost reduction over the original KCR construction. No data for BB in isolation is possible, since BB produces only an unembedded binary tree topology.

Results for the Elmore Delay Model

DME was tested under the Elmore delay model, using the same benchmark sink sets and initial topologies. Results are shown in Table 4.10, and again indicate that DME can obtain substantial improvements in tree cost. The results also show a clear synergy between the topology generation and embedding phases. For example, Tsay's construction does not yield a good initial topology for DME: we believe that this is because it already allows deferral of the choice of

Test Case	number of sinks	MMM cost	KCR cost	KCR+DME cost	BB+DME cost
Prim1	269	161.7	153.9	140.3	140.5
Prim2	603	406.3	376.7	350.4	360.8
r1	267	1815	1627	1497	1500
r2	598	3625	3349	3013	3010
r3	862	4643	4360	3902	3908
r4	1903	9376	8580	7782	8000
r5	3101	13805	12928	11665	11757

Table 4.9 Effect of DME on the KCR and BB constructions, under the linear delay model.

Test Case	Tsay cost	KCR cost	Tsay+DME cost	KCR+DME cost	BB+DME cost
Prim1	*	153.9	*	140.1	140.5
Prim2	*	376.7	*	345.2	360.8
r1	1697	1627	1658	1487	1535
r2	3432	3349	3368	3020	3065
r3	4407	4360	4333	3867	3962
r4	8866	8580	8694	7713	8054
r5	13199	12928	12926	11606	11837

Table 4.10 Comparison of algorithms for the Elmore delay model. Results for [Tsay] are not available for the Primary1 and Primary2 benchmarks.

placements for one level in the tree (the two endpoints of each merging segment are selected and carried to the next level, where the actual embedding is chosen to be the point which allows the minimum connection cost). Indeed, the Table shows clearly that the KCR topology is "more promising" vis-a-vis the subsequent application of DME.[8] Experimental results reported in [45] also indicate

[8]The strong performance of the KCR topologies is surprising. For instance, the top-down BB topology construction [44, 45] carefully considers capacitances and delays of subtrees, in addition to the proximity of their CEPs; this would seem better-suited to the Elmore delay model than the bottom-up KCR approach, which was designed for the linear delay model. Nevertheless, KCR+DME slightly outperforms BB+DME on the seven benchmarks.

a very significant reduction in source-sink Elmore delay, e.g., KCR+DME reduces phase delay by 22% over the trees of Tsay. Finally, [45] notes that that DME constructions with exact zero Elmore delay skew have essentially zero skew (i.e., only a few picoseconds) when evaluated using SPICE.[9] Figure 4.20 shows the output of KCR+DME for the same sink placement of Primary2 depicted in Figure 4.9 above.

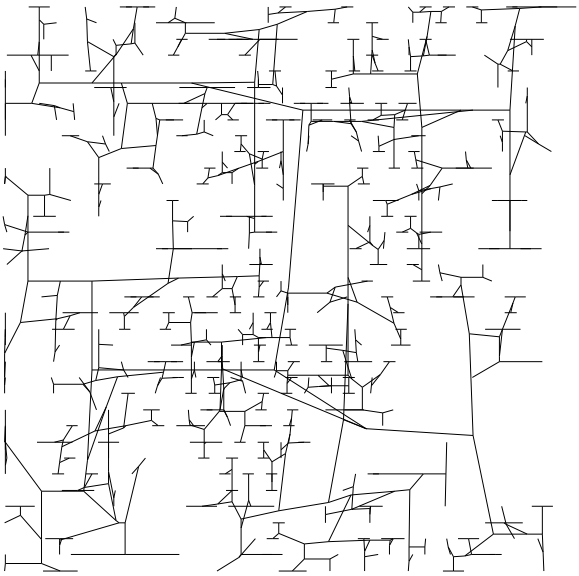


Figure 4.20 Output of KCR+DME on the Primary2 benchmark layout.

[9]The SPICE2G.6 simulations reported in [45] were for the BB+DME construction, and used the following methodology. Random sink sets were generated, with cardinalities ranging from 8 to 64. The routing area was assumed to be $0.5cm \times 0.5cm$, and interconnect and device parameters corresponded to a $1.2\mu m$ CMOS technology. An input clock frequency of 100 MHz and a single buffer were assumed; delays were measured at the output node of each inverter driving a sink node. Quite possibly, an alternate modeling and simulation methodology could change this assessment. An interesting aspect of the SPICE simulations in [45] is that they confirm essentially zero skew, but also show a smaller improvement in phase delay than is indicated by the Elmore delay model. This reflects the studies in the Appendix: Elmore delay is good for predicting skew (i.e., fidelity), but is less useful for predicting absolute delay (i.e., accuracy).

Remarks

DME may be integrated into clock routing design in a number of ways.

- The tree of merging segments allows a choice among alternative minimum-cost zero-skew embeddings of the clock tree. This is useful in design flows where blockages may be introduced before clock routing takes place.
- DME produces a tree with exact zero skew for any input topology, and may thus be applied to previously generated clock trees in order to improve both wirelength and delay.
- DME readily applies to problems of *prescribed* skew (i.e., "useful" skew) [19], where the arrival times of the clocking signal must differ by prescribed amounts. This is handled by setting initial delays at the sinks to non-zero values.
- DME can also be used for problems with *allowed* skew [19, 93, 240], where the signal must arrive at each sink within a prescribed time window. Huang et al. [133] have extended the concept of a merging segment to a *merging area*, and thus address the problem of minimum-cost *bounded-skew* routing; this has applications to both clock distribution and delay-constrained global routing.
- Since both the geometric embedding and the topology generation impinge on solution quality, studies of clock distribution topologies hold renewed interest for research.

For integrated topology generation and embedding, a promising approach is to run DME concurrently with matching-based and other bottom-up topology generating heuristics. Currently, the best DME-based algorithm so far is the Greedy-DME approach of Edahiro (the "CL" algorithm in [84]), which determines the connection topology greedily in bottom-up order, such that each merging segment entails minimum increase in total wirelength. Greedy-DME achieves nearly 17% wirelength reduction over KCR+DME; while it can result in unbalanced leaf depths, it also seems to leave very little room for improvement with respect to total tree cost after DME is applied.

The work of Chou and Cheng [54, 50] proposes a "grafting" operation which perturbs an existing topology by swapping two subtrees. When grafting is used as a neighborhood operator, simulated annealing can be used to optimize the

topology. Chou and Cheng extend DME to the octolinear and Euclidean geometries, and observe that DME no longer returns a minimum-cost ZST: in these geometries, a larger solution space – outside of the merging segments – must be searched to embed the internal nodes of the ZST. Thus, the second phase of their method applies a Gauss-Seidel iteration to embed the topology that was found by simulated annealing. In practice, this approach yields excellent results.

4.4 PLANAR-EMBEDDABLE TREES

Often, it is not easy to realize the preceding "exact zero skew" clock routing solutions by actually placing the wires into the layout plane. Typically, many vias must be introduced, which is undesirable. This difficulty was first noted by Zhu and Dai [259], who gave compelling reasons to seek a *single-layer*, or "planar-embeddable", clock routing solution.

- The clock routing layer may be prescribed, or we may prefer the layer with smallest RC delay.
- Routing on fewer distinct layers (i.e., having fewer distinct electrical parameters to consider) makes the layout more independent of process variation. Uniform electrical parameters also simplify buffering optimizations.
- Single-layer routing eliminates the delay and attenuation of the clock signal through vias, thus improving both performance and signal integrity.

Given these observations, the **Planar Zero-Skew Clock Routing** problem is of interest, i.e., given sink set S, find a *planar-embeddable* ZST $T(S)$ with minimum cost.

Notice that "planar-embeddable" intuitively means that the tree "can be drawn in the plane without edges crossing". However, this concept is not easily characterized in the Manhattan plane; existing works [259] implicitly rely on Euclidean planar-embeddability being sufficient for Manhattan planar-embeddability (a line segment in the Euclidean plane can be approximated to any desired accuracy by a monotone staircase in the Manhattan plane). Thus, we define two tree edges as crossing each other precisely when the corresponding *open* line segments in the Euclidean plane properly intersect (i.e., share exactly one point). This definition is necessitated by possible degenerate optimal planar

clock routing solutions, where the embeddings of edges are superposed. Figure 4.21 shows this phenomenon: four sinks that are collinear will have an optimal "planar" clock tree whose edges pass over each other. Since this sort of overlapping can be made planar with minimum increase in wirelength, we accept such a degenerate solution as planar. This is also the convention of [259].

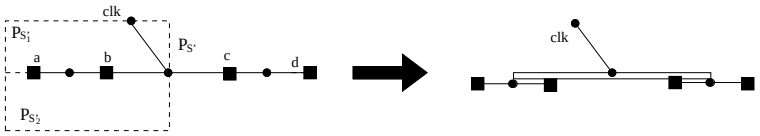


Figure 4.21 For these four sinks on a line, edges of the optimal planar ZST will overlap. We accept this since the ZST can be made non-overlapping with minimal increase in wirelength. The convex polygon $P_{S'}$ and the labels $P_{S'_1}, P_{S'_2}, a$ and b pertain to the correctness proofs of the Planar-DME algorithm that we develop below.

The planar clock routing method of [259] is as follows. The method starts with a tree containing only a connection from the source node to the furthest sink. At each iteration, a sink outside the current tree is connected to a "balance point" in the tree, i.e., via a connection to an existing edge such that zero pathlength skew is maintained and no tree edges are crossed (the Euclidean embedding is assumed). Two rules are applied: (i) the "Min-Rule": a new sink is always connected to the balance point which requires the least wirelength added to the tree; and (ii) the "Max-Rule": the new sink added to the tree is the one which has the greatest distance to its closest balance point. An elegant analysis shows that this method always yields a planar-embeddable zero-skew solution with minimum possible source-sink pathlength. The time complexity of the method is between $\Omega(n \log n)$ and $O(n^2)$.

For the case of four sinks at the corners of the unit square, with the clock source at the center of the square, the method of [259] will create an "X" clock tree with cost = 4, while the optimal "H" solution has cost = 3. A larger 400-point example is shown in Figure 4.22: the method of Zhu and Dai returns an "X-based" configuration, while the Greedy-DME [84] and H-tree [18] constructions are essentially optimal. Khan et al. [155] have observed this limitation, and have proposed applying the MMM top-down partitioning method [143] for a user-specified number of levels, followed by the Zhu-Dai method within each of the resulting regions. When the user-specified number of levels is zero, the

output is the same as that of the Zhu-Dai method. The authors of [155] claim that their algorithm guarantees minimum source-sink pathlength delay; for the Primary1 and Primary2 test cases, approximately 10% tree cost reduction over [259] is obtained.

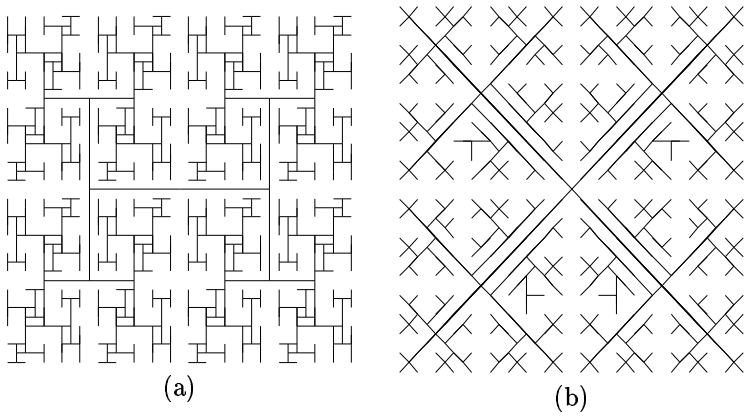


Figure 4.22 Contrast between (a) an H-tree-like solution and (b) the solution of Zhu and Dai. The solution in (a) is actually the output of the Planar-DME method described below.

In the following, we describe an approach due to Kahng and Tsao [153, 154] which naturally unifies the DME embedding strategy and the guaranteed-planar routing objective. The method exploits geometric observations to show that under the linear delay model, the bottom-up and top-down phases of DME can be replaced by a single top-down pass. Whereas DME nominally requires a prescribed topology as input, the "Single-Pass DME" result allows a clock tree topology to be determined *dynamically*, and flexibly, at the same time that it is being optimally embedded (i.e., with minimum cost and minimum source-sink delay).

Building on this observation, the top-down *Planar-DME* algorithm determines a topology that is guaranteed to be *planar-embeddable*, and simultaneously embeds this topology in the Manhattan plane. Beyond being planar, the resulting tree has provably minimum cost and minimum source-sink delay for its topology, since the single top-down pass achieves the same effect as DME.

4.4.1 Single-Pass DME

In this section, we show that under the linear delay model, the tree of merging segments constructed in the bottom-up DME phase can actually be generated in a top-down manner. This result follows from properties of the minimum-pathlength zero-skew subtree over any sink set S' (in particular, that the root of the subtree over S' must be located at the "center" of S').

For any sink subset $S' \subseteq S$, recall that $diameter(S') = \max\{d(s_i, s_j) | s_i, s_j \in S'\}$. Define the *radius* of S' to be $radius(S') = diameter(S')/2$, and let $center(S')$ denote the merging segment of v, where v is the root of the tree of merging segments constructed by DME over S'. (We will see that the distance from $center(S')$ to any sink in S' is at most $radius(S')$, hence this name.) Finally, let $c(S')$ denote the midpoint of $center(S')$.

Recall the following two facts from the above discussion of the DME algorithm (cf. Theorem 4.3.106 and Fact 4.3.107):

Fact 4.4.103 *For any sink set S and topology G, let S_v be the set of sinks in the subtree rooted at v in the DME solution. Let $t_{LD}(v)$ be the linear delay (i.e., pathlength) from v to each sink in S_v. Then $t_{LD}(v) = radius(S_v)$.*

Fact 4.4.104 *Let G be the connection topology of the ZST $T(S)$ that is produced by DME. Let $d = diameter(S)$ and let $TRR(v)$ denote the special tilted rectangular region that corresponds to either $TRR(v) = MD(pl(v), d/2)$ if v is a sink node, or $TRR(v) = TRR(a) \cap TRR(b)$ if v is an internal node of G with children a and b. Then for each node $v \in G$, $core(TRR(v)) = ms(v)$ and $radius(TRR(v)) = d/2 - t_{LD}(v)$.*

Fact 4.4.104 states that for any node v, the merging segment $ms(v)$ is given by the *center* of S_v; Lemma 4.3.103 then implies that $ms(v)$ can be constructed in $O(|S_v|)$ time. Together, facts 4.4.103 and 4.4.104 imply that neither the computation of $ms(v)$, nor the delay time $t_{LD}(v)$, will depend on v's children; this will be the key to constructing the tree of merging segments in top-down order. A third fact is useful in the time complexity analysis:

Fact 4.4.105 *For any sink subset $S' \subseteq S$ in the Manhattan plane, $radius(S') = diameter(S')/2$ can be computed in linear time.*

Theorem 4.4.106 *Given a set of sinks S and a connection topology G, we can produce the same output ZST $T(S)$ that the DME algorithm will produce under the linear delay model, using only a single top-down phase with time complexity $O(|S|^2)$.*

Proof: We will show that for any node v in G, $ms(v)$ can be found in time linear in the total number of descendants of v. Let the terms d and $TRR(v)$ be defined as in the statement of Fact 4.4.104. The value of d can be found in $O(|S|)$ time (Fact 4.4.105), and in $O(|S|)$ time we can build $TRR(s)$ for all sinks s (leaf nodes in G). According to Fact 4.4.104, $ms(v) = core(TRR(v)) = core(\bigcap_{u \in S_v} TRR(u))$, where S_v is the set of descendants of node v in G. Since the intersection of any two TRR's can be found in constant time and is also a TRR, we can compute $TRR(v)$ and its core in time proportional to the number of v's descendants (cf. Lemma 4.3.103).

If v is not the root of G, let p be its parent. By Fact 4.4.103, the length of the edge incident to node v in G, $|e_v|$, is equal to $t_{LD}(p) - t_{LD}(v) = radius(S_p) - radius(S_v)$, where S_v and S_p are the sets of descendants of node v and p, respectively. Thus, $ms(v)$ and $|e_v|$ can be computed in $O(|S_v|)$ time, and we now have the information that would have been provided by the bottom-up phase of Deferred-Merge DME. In the best case, the height of the tree of merging segments is $O(\log |S|)$, so that the overall time complexity is $\Omega(|S| \log |S|)$. In the worst case, the height of the tree of merging segments is $\Theta(|S|)$, implying $O(|S|^2)$ overall time complexity. □

Thus, under the linear delay model $ms(v)$ is independent of the connection topology over S_v. Furthermore, $t_{LD}(v) = radius(S_v)$ implies that all sinks in S_v are within distance $radius(S_v)$ of $center(S_v)$, i.e., $center(S_v)$ is the merging segment of the root of *any* ZST over S_v that has minimum source-sink pathlength delay. This immediately yields what [153, 154] call the *Single-Pass DME* method. Because Single-Pass DME results in the same optimum ZST that DME would achieve, established properties of the output tree (i.e., minimum source-sink pathlength and minimum total tree cost with respect to the generated connection topology) are maintained.

4.4.2 The Planar-DME Algorithm

The impact of Theorem 4.4.106 may not be immediately apparent, since DME can already accomplish the same construction as Single-Pass DME in linear time. However, the theorem's proof showed that as soon as Single-Pass DME

has been given a partitioning of S_v into S_a and S_b, it can immediately find the $ms(a)$ and $ms(b)$ that are compatible with an optimal ZST having this "top part" of the clock topology. Thus, Single-Pass DME allows the connection topology to be determined dynamically in a top-down fashion, yet still finds a minimum-pathlength, minimum-cost embedding of whatever topology is eventually determined. If Single-Pass DME chooses the connection topology and embeds it carefully, then a planar routing can be achieved.

The Planar-DME algorithm [153, 154] is essentially a version of Single-Pass DME wherein the connection topology is determined based on the existing routing, such that future routing cannot interfere with this existing routing. To describe the algorithm, we require two terms that are defined in the Euclidean plane: (i) $P_{S'}$ denotes any *convex polygon* containing S', and (ii) *convex-hull*(S') is the $P_{S'}$ with minimum area. Also, we say that a point p lies *inside* $P_{S'}$ if p is on the boundary of, or lies strictly interior to, $P_{S'}$. The convex polygon concept is used to guide the top-down partitioning of both the routing area and the set of sinks, as follows. Given a Euclidean convex polygon that contains a given set of sinks, we will divide this polygon into two smaller convex polygons, in such a way that a minimum-cost ZST is still possible and the routing within one polygon cannot interfere with the routing in the other polygon. This will be done recursively until every polygon contains exactly one sink.

More precisely, in each recursive call of Planar-DME, we start with a Euclidean convex polygon $P_{S'}$ containing $S' \subseteq S$. The existing routing is outside or on the boundary of $P_{S'}$, and terminates at some node p of the topology that will eventually connect to its child node v. As long as two properties are maintained – (i) v is embedded at a point that is compatible with the DME solution, and (ii) $P_{S'}$ is partitioned into two smaller convex polygons such that the routing from p to v is on the boundary between the polygons – a planar DME-like solution will remain possible. Recall that the topology is determined dynamically: $S' = S_v$, and partitioning S' into S'_1 and S'_2 yields the sink sets S_a and S_b for v's children a and b.

Finally, the Planar-DME algorithm of [154] is derived from Single-Pass DME by introducing the following rules for *embedding* the internal nodes of the ZST, and for top-down *partitioning* of the sinks in each subtree.

The **embedding rules** embed v inside $P_{S'}$ such that the embedding is compatible with the DME solution, i.e., they maintain the first property above (see Figure 4.23). In each recursive call, Planar-DME accepts a subset of sinks $S' \subseteq S$, some convex polygon $P_{S'}$ containing S', and some point p inside $P_{S'}$ which is to connect to a point v on $ms(v) = center(S')$. The existing routing

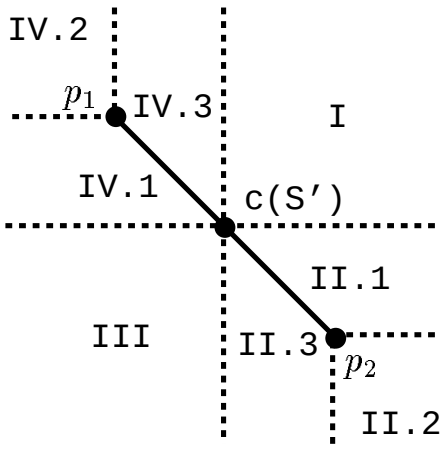


Figure 4.23 Rules to choose the embedding point of v (the root of the subtree over sink set $S' \subseteq S$ in *any* minimum-radius ZST), and to choose the the splitting line to partition the sink set S' based on the relative positions of v's parent p and $center(S')$. If we denote the coordinates of $c(S')$ by (x_c, y_c), then the regions are defined by the following inequalities: Region I: $x \geq x_c$, $y \geq y_c$; Region $II.1$: $y \geq -x + y_1 + x_1$, $y \leq y_c$, $y \geq y_2$; Region $II.2$: $x \geq x_2$, $y \leq y_2$; and Region $II.3$: $y \leq -x + y_1 + x_1$, $x \geq x_c$, $x \leq x_1$. Regions III, $IV.1$, $IV.2$,and $IV.3$ are defined similarly.

is outside $P_{S'}$, so if we can select a feasible embedding point v inside $P_{S'}$, then the routing from p to v will not interfere with any routing that is external to $P_{S'}$. As a consequence, the resulting routing will be planar. The point p is the embedding of v's parent, and has been determined earlier in the top-down pass.[10] The merging segment $ms(v) = center(S')$ has endpoints p_1 and p_2 in the figure. To be compatible with the DME solution, we choose to embed v on the portion of $center(S')$ that is closest to the location of p. Furthermore, to ensure that v is embedded inside $P_{S'}$, we embed v at a point on this chosen portion of $center(S')$, as close as possible to the midpoint of $center(S')$, denoted $c(S')$. This guarantees an embedding point inside $P_{S'}$ [153].

[10]We use, e.g., p to denote either a node in the tree topology or the point at which that node has been embedded in the Manhattan plane (that is to say, $pl(p)$).

The actual embedding rules for v depend on p's location, as follows (see Figure 4.23).

- Region I, III: $v = c(S')$ (since this is one of the points on $center(S')$ that is closest to p).
- Region $II.1, IV.1$: v is the point of intersection of $center(S')$ with the horizontal line through point p (in this case and the following cases, we embed v at a point on $center(S')$ that is closest to p, and as close to $c(S')$ as possible).
- Region $II.3, IV.3$: v is the point of intersection of $center(S')$ with the vertical line through point p.
- Region $II.2$: $v = p_2$.
- Region $IV.2$: $v = p_1$.

The **partitioning rules** for S' are also straightforward: the goal is to find an appropriate *splitting line* that divides $P_{S'}$ into two convex polygons and thus also partitions the sink set between the two subtrees that are below v. Essentially, we can use any line through p and v as a splitting line.

- If $p \neq v$ we extend the line segment $\overline{pv}$ to be a splitting line $\overleftrightarrow{pv}$ which divides $P_{S'}$ into two convex polygons $P_{S'_1}$ and $P_{S'_2}$. Any sink lying inside one of the convex polygons is assigned to that polygon, thus determining membership in either S'_1 or S'_2; a sink on $\overleftrightarrow{pv}$ can be assigned to either polygon as long as neither S'_1 or S'_2 is empty. For example, in Figure 4.21 $S' = \{a, b\}$ is divided into $S'_1 = \{a\}$ and $S'_2 = \{b\}$, and $P_{S'}$ is divided into $P_{S'_1}$ and $P_{S'_2}$ accordingly.
- The case where $p = v$ is resolved as follows: if $p \neq c(S')$ then the line segment $center(S')$ is extended to form the splitting line, otherwise we arbitrarily choose the vertical line through p as the splitting line.

These simple choices of embedding and partitioning rules guarantee a planar result for Single-Pass DME [154].

Theorem 4.4.107 *Given a subset $S' \subseteq S$, a convex polygon $P_{S'}$, and a point p inside $P_{S'}$, the embedding rules will select a feasible embedding point v inside $P_{S'}$ and the partitioning rules will divide S' into two nonempty subsets.* □

Theorem 4.4.108 *Planar-DME constructs a planar clock routing tree.* □

The Planar-DME algorithm is formally described in Figure 4.24. Steps 4 and 6 in Planar-DME-Sub are the crux of the difference between Planar-DME and the generic Single-Pass DME. There will be at most $n = |S|$ levels of recursion, since the maximum size of any sink subset decreases by at least one at each level. Thus, Planar-DME has the same $\Omega(n \log n)$ and $O(n^2)$ complexity bounds as Single-Pass DME and the method of Zhu and Dai. The example of Figure 4.25 illustrates the Planar-DME construction.

4.4.3 Experimental Results and Discussion

Planar-DME, with the simple polygon partitioning scheme described above, has been implemented in C and tested on the seven test cases described in Section 4.3.5. For the linear delay model, Table 4.11 compares Planar-DME against three other methods: the method of Zhu and Dai [259]; the KCR+DME method which gave the best results in the previous section; and the Greedy-DME method of Edahiro [84], which gives the best-known wirelength results for zero-skew trees. Recall that the method of [259] is planar; the KCR+DME method yields a height-balanced tree topology; and Greedy-DME can yield a height-unbalanced solution.

Planar-DME obtains an average of 15.5% reduction in tree cost versus the previous planar method of [259]. Surprisingly, the Planar-DME solution has lower cost than the *non-planar* KCR+DME solution for Primary1 and r5, and indeed Planar-DME has tree cost very comparable to that of KCR+DME for the other test cases. For the Primary1 test case, Planar-DME surpasses the Greedy-DME result; this may be due to the very regular arrangement of sinks in the Primary1 layout. On the other hand, results are worst for the r2 example, perhaps due to the highly irregular distribution of sinks in this test case. Kahng and Tsao [153] have described extensions to Elmore delay and alternate partitioning rules, and Huang et al. [133] have applied a variant as the core of a bounded-skew clock routing heuristic.

Finally, Figure 4.26 shows the planar clock routing solutions constructed by Planar-DME and the algorithm of [259] for the benchmark placement of the Primary2 circuit.

Algorithm Planar-DME (S,clk)
Input: Set of sinks S; clock location clk in P_S
Output: Planar ZST $T(S)$ with root s_0; $cost(T(S))$

1. $r = radius(S)$
2. Build $TRR(u) = MD(u, r)$ for all sinks $u \in S$
3. $center(S) = core(\bigcap_{u \in S} TRR(u))$
4. **If** clk not specified
5. Embed s_0 at $c(S)$ (i.e., $pl(s_0) = c(S)$);
6. **Else**
7. Embed s_0 at clk (i.e., $pl(s_0) = clk$)
8. $t_{LD}(s_0) = r + d(pl(s_0), center(S))$
9. Planar-DME-Sub(S,P_S,s_0)
10. $cost(T) = \sum_{v \in T} |e_v|$

Procedure Planar-DME-Sub (S',$P_{S'}$,p)
Input: Set of sinks $S' \subseteq S$; convex polygon $P_{S'}$ containing S'; parent node p lying inside $P_{S'}$
Output: Planar ZST $T(S')$ with root v

1. $t_{LD}(v) = radius(S')$
2. $ms(v) = center(S') = core(\bigcap_{u \in S'} TRR(u))$
3. $|e_v| = t_{LD}(p) - t_{LD}(v)$
4. Use embedding rules to embed node v at $pl(v) \in ms(v)$
5. Connect a wire from $pl(p)$ to $pl(v)$
6. Use the partitioning rules to divide S' and $P_{S'}$ into S'_1 and S'_2, and $P_{S'_1}$ and $P_{S'_2}$
7. $parent(v) = p$
8. **If** $|S'| = 1$ **Return**
9. Planar-DME-Sub(S'_1,$P_{S'_1}$,v)
10. Planar-DME-Sub(S'_2,$P_{S'_2}$,v)

Figure 4.24 Planar-DME Algorithm.

4.5 REMARKS

Clock distribution is now one of the most actively studied areas of physical design. As noted at the outset, clock distribution is also highly intractable: it impinges on system architecture, circuit design, and discrete algorithms, and

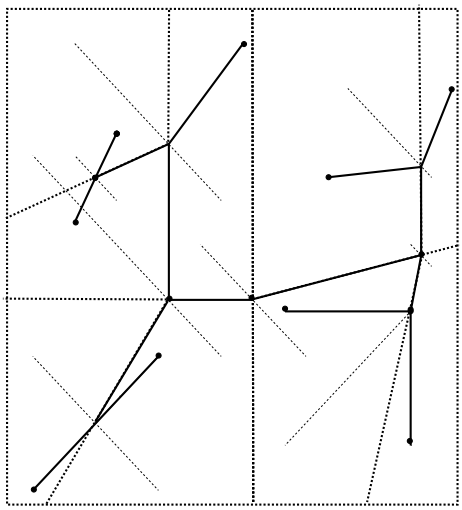


Figure 4.25 Example with 9 sinks (circular dots at leaf nodes), illustrating execution of Planar-DME. The routing region is recursively divided into convex polygons according to the partitioning rules (boundaries of polygons are indicated by thick dotted lines). Also shown is the tree of merging segments (thin dashed lines), from which application of the embedding rules is apparent.

is an area where "theory" and "practice" can diverge to a disconcerting extent (e.g., contrast the present abstractions of "exact zero skew" with the formulations surveyed in [100]). Fortunately, the existing literature has established a number of fundamental techniques for clustering in topology generation, optimal (planar) embedding of prescribed topologies, and achieving prescribed Elmore delay skew. Furthermore, CAD researchers are now beginning to address more realistic problem formulations. The following are just a few examples.

- To reduce power requirements, interconnect optimization to reduce capacitance and architecture design to reduce switching frequency are both of interest, by virtue of the $C \cdot V^2 \cdot f$ dependence of dynamic power dissipation. Increasing the system performance, e.g., in pipelined architectures, requires accurate management of latency. To address these issues, *wiresizing and buffer insertion* optimizations have been proposed by [217, 237, 197, 261] and others.

	#sinks	Greedy-DME	KCR+DME	Planar-DME	Zhu-Dai
Prim1	269	137.0	140.3	136.0	167.9
Prim2	594	311.4	350.4	353.7	422.5
r1	267	1,331.9	1,497	1,511.8	1,778.3
r2	598	2,590.8	3,013	3,363.5	3,580.1
r3	862	3,317.8	3,902	3,943.9	4,635.9
r4	1,903	6,780.2	7,782	7,835.7	9,577.1
r5	3,101	9,890.5	11,665	11,491.1	14,119.4
Complexity		$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(n^2)$
Planarity		NO	NO	YES	YES

Table 4.11 Comparison of Planar-DME with other algorithms under the linear delay model. Clock tree costs for Greedy-DME are quoted from [84], and costs for KCR+DME are quoted from [45]. Planar-DME is executed without any prescribed clock source location.

- *Skew control*, as opposed to "exact zero skew", often represents a reasonable engineering solution that can save wiring cost while maintaining acceptable performance (cf., [260, 209]). Prescribed-delay routing can be accomplished by DME variants or by the method of [214]. Bounded-skew routing has been addressed in [133].

- For high-speed systems, process variations during manufacturing can easily introduce the several hundred picoseconds of skew needed to cause system failure. Thus, *process variation independence* has emerged as a new design criterion. Wiresizing and buffer sizing methods were developed by [56, 197]; the single-layer routing methods of [259, 153] nominally also address this issue.

- When very large "superbuffers" or mesh topologies are used in the clock distribution network (recall the example of the *Alpha* microprocessor [76], where 3% of the active area is occupied by a single clock driver), there are effectively *multiple sources* in the clock routing problem. For both general signal net routing (where multiple drivers are required to drive fanins of a large signal net at high speeds) and clock distribution, considering multiple sources seems to be an emerging research area [167, 175].

- Finally, for multi-chip packaging technologies with area-array pads, the clock distribution problem becomes *hierarchical*. The area-array pads en-

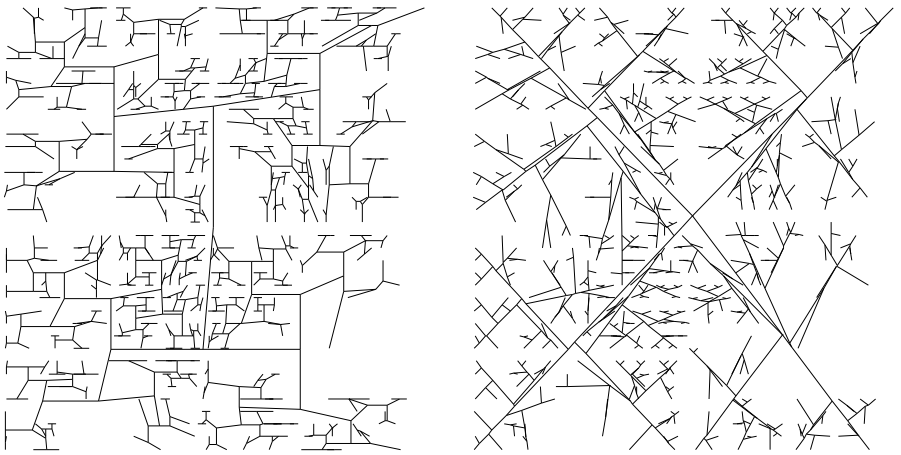


Figure 4.26 Planar zero-skew clock trees with minimum source-sink pathlength delay, for the Primary2 benchmark: (a) the solution produced by Planar-DME; and (b) the solution produced by the algorithm of Zhu and Dai.

able signals to be brought onto the die from the substrate at any location; this can lead to many clock "sources" on a given die. Then, the problem is to partition the clock distribution between the underlying substrate and the on-chip routing areas. Zhu and Dai [259] have performed the first investigations of this problem.

5

MULTIPLE OBJECTIVES

Overview

In previous chapters we have explored constructions that optimize the three main design objectives of wirelength, skew, and delay. However, in practice we often seek to optimize multiple objectives simultaneously. This chapter explores ways of representing and addressing multiple competing objectives. We begin with a *minimum density* formulation for balancing the utilization of horizontal and vertical routing resources and describe heuristics with expected performance bounded by constants times optimal. This enables the simultaneous optimization of up to three objectives (e.g., radius/density/wirelength, or skew/density/wirelength at once), without degrading solution quality with respect to any of the objectives. We also discuss a non-uniform lower bound schema that affords tighter estimates of solution quality for a given problem instance.

Next, we develop a general framework of multiple-objective optimization, based on multi-weighted graphs (i.e., where edge weights are vectors rather than scalars). This formulation captures distinct criteria such as wirelength, jogs and congestion, and enables effective routing in graph-based regimes (i.e., routing in building-block designs, field-programmable gate arrays, and where obstacles are present). Finally, we discuss a network-flow based approach to prescribed-width routing where multiple objectives induce an arbitrarily costed region; applications of this include, e.g., circuit-board routing, and routing with respect to reliability or thermal considerations. This methodology departs from conventional shortest-path or graph-search based methods in that it applies to routing regions with a continuous cost function, as well as to regions containing

solid polygonal obstacles. Extensions address the minimum-surface problem of Plateau, which is of independent interest.

5.1 MINIMUM DENSITY TREES

In Chapter 2 the minimum-area objective was approximately captured by minimizing the tree cost: since wires have a fixed width and must be routed at a fixed separation from each other, the total tree edgelength provides an obvious lower bound on the routing area that must be added to the layout. However, the grid-based structure of integrated-circuit routing resources provides additional information for determining the impact of a given interconnection topology on the chip area.

This section discusses the *minimum density* objective of [10, 11] for spanning and Steiner tree constructions. This formulation is motivated by the minimum-area layout objective, which is best achieved through balancing the usage of horizontal and vertical routing resources [194]. We present two efficient heuristics for constructing low-density spanning trees, and prove that their outputs are within small constants of optimal with respect to both tree cost and density. The proof techniques suggest a non-uniform lower bound schema which affords tighter estimates of solution quality for a given problem instance. Furthermore, the minimum density objective can be transparently combined with a number of previous interconnection objectives (e.g., minimizing tree radius or skew) without affecting solution quality with respect to these previous metrics. Section 5.2 details a more general scheme for the simultaneous optimization of multiple objectives.

Consider the four-terminal signal net shown in Figure 5.1; the interconnection tree of Figure 5.1(a) forces at least three wires to cross the dashed line, meaning that the horizontal dimension of the chip must increase enough to accommodate these three *routing grids*.[1] In contrast, the tree of Figure 5.1(b) forces the horizontal chip dimension to grow by only one routing grid (however, the vertical chip dimension will grow by two grids, as indicated by the horizontal dashed line). Manufacturing and Packaging costs suggest that the most effective layouts are generally those which are roughly square, and this suggests balancing the horizontal and vertical routing requirements induced by the interconnection tree.

[1] We adopt "routing grid" as a generic term that is independent of layout methodology. The term encompasses, e.g., vertical feedthroughs or horizontal routing tracks in a channel [194].

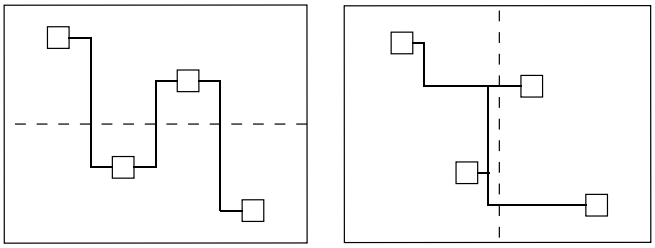


Figure 5.1 A four-terminal signal net for which the tree on the left increases the required layout dimension by three routing grids, while the tree on the right requires only two routing grids.

Recall that a *signal net* S is a set of $n+1$ terminals $s_0, s_1, s_2, \ldots, s_n \in S$ in the Manhattan plane, and *routing tree*, $T(S)$ is a tree which spans S. The *cost* of a tree edge is the Manhattan distance between its endpoints, and the cost of a routing tree is the sum of the costs of its edges. A line *properly* intersects an edge if and only if it intersects the edge at a single point which is not an endpoint of the edge.

Definition 5.1.1 *The* density *of an interconnection tree is the maximum number of tree edges that can be properly intersected by a horizontal or vertical line in the plane.*

Definition 5.1.2 *For a given net S, the* minimum density *of S is the minimum density achievable by an interconnection tree $T(S)$, and a* minimum density tree *is any $T(S)$ that achieves this minimum density.*

Minimum Density Tree (MDT) Problem: Given a net S, construct a minimum density tree $T(S)$ having minimum cost.

The density criterion (see figure 5.2) recalls the notion of trees with "low stabbing number", which are used in the computational geometry literature to speed up dynamic "ray shooting" queries [1, 46, 85, 86, 247]. However, spanning trees with low stabbing number minimize the number of tree edges that can be intersected by a line of *any* orientation, while the MDT formulation

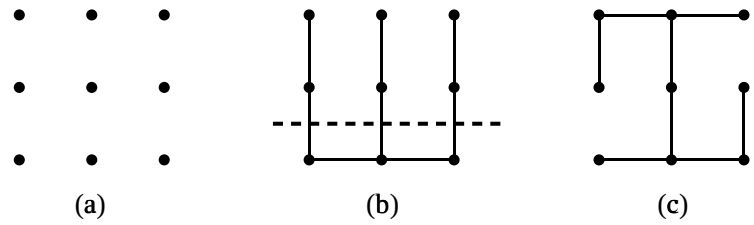


Figure 5.2 (a) Example of a signal net, along with (b) an interconnection tree with density = 3, and (c) a minimum density tree with density = 2.

above is concerned only with horizontal or vertical intersecting lines; this difference enables tighter bounds and simultaneous minimization of both tree cost and density.

In light of the minimum area, delay and skew objectives discussed earlier, the MDT heuristics discussed below provide interesting multiple optimizations wherein up to three competing objectives may be optimized *simultaneously*. As a result, the area minimization objective of minimum-density routing can be attained without sacrificing performance-driven criteria. In particular, below we describe how tree cost, radius, and density can be simultaneously optimized; we also show how tree cost, skew and density can be simultaneously addressed.

5.1.1 Heuristics for Minimum Density Trees

The following discussion will assume that all terminals lie inside the unit square.

The COMB Construction

Our first basic algorithm sorts the terminals by increasing x-coordinate (ties are broken to favor the larger y-coordinate), and then partitions the terminals into $\frac{\sqrt{n}}{\sqrt{2}}$ vertical *strips*, each containing $\sqrt{2n}$ terminals (Figure 5.3(a)). (Note that the discussion implicitly assumes use of the floor and ceiling functions as appropriate; this does not affect any of the asymptotic results.) We then connect all the terminals of each strip into a path, in order of decreasing y coordinate (Figure 5.3(b)). We complete the routing topology by connecting the termi-

nals with lowest y coordinate in each strip, in order from left to right (Figure 5.3(c)). This algorithm is described in Figure 5.4. The complexity of this algorithm, which we call COMB, is clearly dominated by the partitioning/sorting step (Step 1 of Figure 5.4), and is therefore $O(n \log n)$.

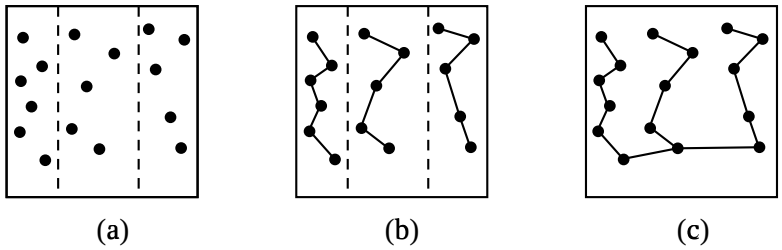


Figure 5.3 Execution of the COMB spanning tree construction on a net of size $n = 16$.

Algorithm: COMB	
Input: a net S, containing $\|S\| = n$ terminals	
Output: a low-density, low-cost tree spanning S	
1:	**Partition** S into $\frac{\sqrt{n}}{\sqrt{2}}$ vertical strips each containing $\sqrt{2n}$ terminals
2:	**Connect** in monotone y-order the terminals within each strip
3:	**Connect** in monotone x-order the bottom terminals of all strips
4:	**Output** resulting spanning tree

Figure 5.4 Algorithm COMB: heuristic minimum-density spanning tree construction.

If the introduction of Steiner points is allowed in constructing the tree, we can reduce the worst-case density as well as the worst-case cost of the construction as follows: (i) partition the net S into $\frac{\sqrt{n}}{\sqrt{2}}$ vertical strips, each containing $\sqrt{2n}$ terminals (Figure 5.5(a)); (ii) within each strip, connect the terminals in the strip to a central *spine*, i.e., a vertical line which passes through the median terminal of the strip when the terminals are sorted by x-coordinate (Figure 5.5(b)); then (iii) join all the spines using segments of a single horizontal line (Figure 5.5(c)). This variant, which we call COMB_ST, is described in Figure 5.6 and has complexity $O(n \log n)$, again reflecting the complexity of the partitioning/sorting step.

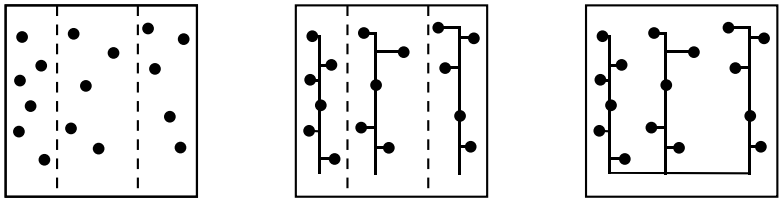


Figure 5.5 Execution of the COMB_ST Steiner tree construction on a net of size $n = 16$. Note that density = 3 is achieved by the construction, while the COMB construction yielded density = 5 for the same instance.

Algorithm: COMB_ST
Input: a net S, containing $\|S\| = n = k^2$ terminals
Output: a low-density low-cost Steiner tree connecting S
1: **Partition** S into $\frac{\sqrt{n}}{\sqrt{2}}$ vertical strips each containing $\sqrt{2n}$ terminals
2: **Connect** the terminals within each strip to a central spine
3: **Connect** the bottoms of all spines
4: **Output** resulting Steiner tree

Figure 5.6 Algorithm COMB_ST: heuristic minimum-density Steiner tree construction.

A Chain-Peeling Method

A different, "chain-peeling" approach to density minimization iteratively computes and superposes chains or antichains (i.e., sets of terminals through which a staircase routing exists). A *chain* is a sequence of terminals with coordinates that are monotone nondecreasing in both x and y; an *antichain* has coordinates monotone nondecreasing in x and monotone nonincreasing in y. According to Dilworth's theorem [75], every partially ordered set of size n must contain either a chain or an antichain of size at least $\sqrt{n}$.

The chain-peeling method, which we call PEEL (Figure 5.7), detects a maximal chain or antichain and then removes it from the net; the process is iterated over the remaining terminals until the net has been covered. Each chain contributes

at most 1 to the overall density, and the chains/antichains can be joined together into a tree (Step 7 of Figure 5.7) without increasing the density further (see Theorem 5.1.8 below). The PEEL method is attractive because it escapes such pathological examples as that of Figure 5.8, where COMB or COMB_ST will yield density an unbounded factor greater than that of PEEL. Section 5.1.2 shows that the time complexity of PEEL is $O(n^{\frac{3}{2}} \log\log n)$.

Algorithm: PEEL
Input: a net S, containing $\|S\| = n$ terminals **Output:** a low-density low-cost tree spanning S
1: $P = S$ 2: $T = \emptyset$ 3: **While** $P \neq \emptyset$ **Do** 4: $C =$ maximum chain or antichain of P 5: $T = T \cup C$ 6: $P = P - C$ 7: **Join** all chains/antichains in T and output resulting tree

Figure 5.7 Algorithm PEEL produces a low-density tree by iteratively computing maximum chains or antichains, then joining them into a tree.

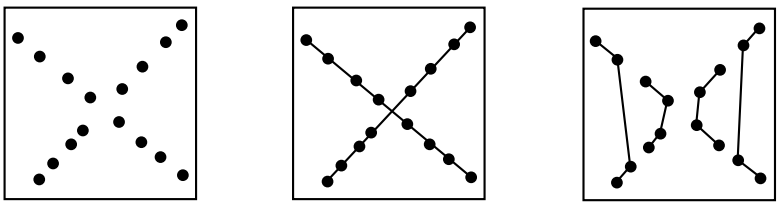


Figure 5.8 Illustration of class of examples (a) on which PEEL performs (b) an unbounded factor better than either COMB or COMB_ST (c). The connecting edges between the strips (Step 3 of COMB) are not shown in (c). For points in an "X" configuration, PEEL will always yield a constant density = 2, while COMB or COMB_ST density will grow as the square root of n.

5.1.2 Performance Bounds

Both the density and the total tree cost of the constructions are on average only small constant factors away from optimal.

Density Bounds

For a net S of n terminals, a lower bound of $\Omega(\sqrt{n})$ can easily be established for the worst-case minimum density of the spanning tree $T(S)$. Moreover, we can show $\Theta(\sqrt{n})$ expected density for the minimum density tree over S.

Theorem 5.1.3 *A net S containing the n distinct grid points of the $(\sqrt{n}-1)\times(\sqrt{n}-1)$ grid cannot be spanned by a tree having density $< \lceil\frac{\sqrt{n}+1}{2}\rceil$.*

Proof: In the square array, there are $2(\sqrt{n}-1)$ horizontal and vertical lines between the rows and columns of terminals. For the tree $T(N)$ to be connected, each tree edge must cross at least one horizontal or vertical line. Hence, there are at least $n-1$ line crossings, and the pigeonhole principle implies that at least one of the lines is crossed $\lceil\frac{n-1}{2(\sqrt{n}-1)}\rceil = \lceil\frac{\sqrt{n}+1}{2}\rceil$ times. □

The next theorem requires the following lemma:

Lemma 5.1.4 *If n indistinguishable balls are independently placed at random into n indistinguishable boxes, $(1-\frac{1}{e})\cdot n$ boxes are expected to be non-empty.*

Proof: The probability of a given ball ending up in a given box B_i is $\frac{1}{n}$, thus the probability of the ball missing box B_i is $1-\frac{1}{n}$. By the independence of the placements, the probability that all n balls miss box B_i is $(1-\frac{1}{n})^n$. Therefore, as n increases, the probability that any given box remains empty is $\lim_{n\to\infty}(1-\frac{1}{n})^n = \frac{1}{e}$. By linearity of expectation, it follows that a constant fraction $\frac{1}{e}$ of the n boxes are expected to remain empty, proving the lemma. □

Theorem 5.1.5 *For n terminals chosen randomly from a uniform distribution in the unit square, the minimum density tree has expected density $\Theta(\sqrt{n})$.*

Proof: Partition the unit square into n identical square cells, each of size $\frac{1}{\sqrt{n}}$ by $\frac{1}{\sqrt{n}}$, using $2\sqrt{n}-2$ vertical and horizontal lines (Figure 5.9(a)). If we regard

cells as "boxes" and terminals as "balls" then by Lemma 5.1.4 the expected number of cells containing at least one terminal is $(1 - \frac{1}{e}) \cdot n$. For the spanning tree to be connected, each of these non-empty cells must contain at least one terminal s_i which has an incident tree edge e_j that crosses a boundary of the cell (Figure 5.9(b)). By the pigeonhole principle, at least one of the horizontal or vertical lines will intersect $\frac{(1-\frac{1}{e})\cdot n}{(2\sqrt{n}-2)} > (1 - \frac{1}{e}) \cdot \frac{\sqrt{n}}{2}$ tree edges, yielding the $\Omega(\sqrt{n})$ lower bound on the expected minimum density. Since the algorithms always yield trees with density $O(\sqrt{n})$ (see the following sequence of results), the expected minimum routing density for a net of n terminals uniformly chosen in the unit square is $\Theta(\sqrt{n})$. □

Interestingly, the proof schema of Theorem 5.1.5 suggests a computational lower bound for individual instances of the MDT problem, as follows. Given a net S, select integers i and j and partition the unit square into an i by j (not necessarily uniform) rectangular grid such that the greatest number P of the resulting $i \cdot j$ rectangles contain terminals (see Figure 5.10). By the pigeonhole principle (recall the proof of Theorem 5.1.5), this induces an immediate lower bound of $\lceil \frac{P-1}{(i-1)+(j-1)} \rceil = \lceil \frac{P-1}{i+j-2} \rceil$ on the minimum routing density of S. Various schemes can be used to find a partition which maximizes the quantity $\frac{P-1}{i+j-2}$: for example, one could place $i = \sqrt{n}$ horizontal lines such that at most $\sqrt{n}$ terminals lie between each consecutive pair of horizontal lines, and then place the $j = \sqrt{n}$ vertical lines using a similar criterion. It is open whether there exists a polynomial-time algorithm which computes a rectangular partition that maximizes $\frac{P-1}{i+j-2}$, even for fixed i and j (e.g., $i = j = \sqrt{n}$).

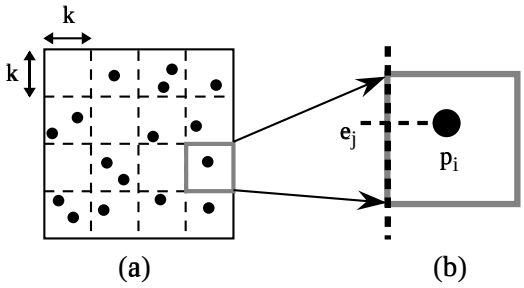


Figure 5.9 Expected minimum density of a net: (a) the unit square is partitioned into n congruent cells; (b) each non-empty cell contains some terminal s_i which contributes at least one edge e_j that crosses a cell boundary.

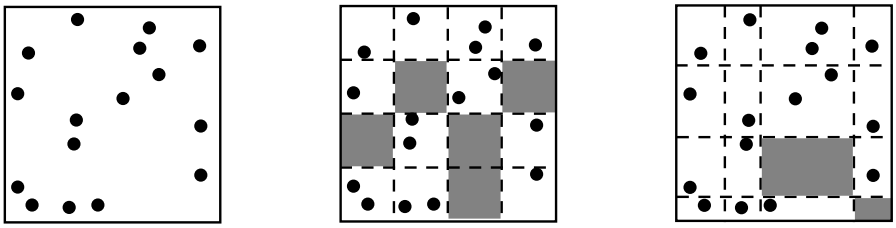


Figure 5.10 Computing a non-uniform lower bound on the density. For the net in (a), a uniform partition of the unit square into 16 squares of size $\frac{1}{4} \times \frac{1}{4}$ each, shown in (b), yields 11 non-empty cells which imply a density lower bound of $\lceil \frac{11-1}{(4-1)+(4-1)} \rceil = 2$. On the other hand, the non-uniform partition shown in (c) yields 14 non-empty cells, which imply an improved density lower bound of $\lceil \frac{14-1}{(4-1)+(4-1)} \rceil = 3$.

Theorem 5.1.6 *Algorithm COMB constructs a spanning tree with density* $\leq \sqrt{2n}$.

Proof: Since each strip contains no more than $\sqrt{2n}$ terminals, a vertical line passing through any strip cannot intersect more than $\sqrt{2n}$ tree edges. Since any given horizontal line cannot intersect more than two edges within a strip (one edge from Step 2 and one from Step 3 in Figure 5.4), the maximum horizontal density is $2 \cdot \frac{\sqrt{n}}{\sqrt{2}} = \sqrt{2n}$. Thus, the density of the COMB output is at most $\sqrt{2n}$. □

Theorem 5.1.7 *Algorithm COMB_ST constructs a Steiner tree with density* $\leq \frac{\sqrt{n}}{\sqrt{2}} + 1$.

Proof: In the construction of Figure 5.5, a strip can contain at most $\frac{\sqrt{n}}{\sqrt{2}}$ terminals on each side of its spine, so no vertical line can intersect more than $\frac{\sqrt{n}}{\sqrt{2}}$ of the edges created in Step 2 of Figure 5.6. No horizontal line can intersect any of the $\frac{\sqrt{n}}{\sqrt{2}}$ vertical spines more than once. Thus, the density of the COMB_ST

output is at most $\frac{\sqrt{n}}{\sqrt{2}} + 1$, when we consider the edges added to join the spines together (Step 3). □

A density bound for the chain-peeling algorithm PEEL follows two lemmas, namely, (i) at most $O(\sqrt{n})$ chains or antichains will be "peeled" during the construction, and (ii) these chains/antichains can be connected to form a single component which has density at most the number of chains/antichains.

Theorem 5.1.8 *Algorithm PEEL constructs a Steiner tree with density most* $\leq 2 \cdot \sqrt{n}$.

Proof: We first show that PEEL computes at most $2 \cdot \sqrt{n}$ chains and/or antichains. Let a_i denote the number of points remaining after we have peeled off i chains and/or antichains. Assume that the algorithm stops when we have peeled off k chains and/or antichains, i.e., $a_k = 0$. We want to show that $k \leq 2 \cdot \sqrt{n}$. According to Dilworth's Theorem [75], the size of the $(i+1)^{\text{th}}$ chain/antichain is at least $\sqrt{a_i}$. Thus, $a_{i+1} \leq a_i - \sqrt{a_i}$. It is easy to verify that $\sqrt{x - \sqrt{x}} \leq \sqrt{x} - \frac{1}{2}$. Therefore,

$$\sqrt{a_k} \leq \sqrt{a_{k-1} - \sqrt{a_{k-1}}} \leq \sqrt{a_{k-1}} - \frac{1}{2} \leq (\sqrt{a_{k-2}} - \frac{1}{2}) - \frac{1}{2} \leq \ldots \leq \sqrt{a_0} - \frac{k}{2}$$

This implies that $k \leq 2 \cdot (\sqrt{a_0} - \sqrt{a_k}) = 2 \cdot \sqrt{n}$. To complete the proof, we need to show the chains and antichains can be "joined" into a spanning tree without increasing density. This can be accomplished by extending each chain to the top-right corner of the unit square and each anti-chain to the top-left corner; this clearly will not increase total density beyond k (see Figure 5.11). A simple case analysis shows that the set of chains can then be connected to the set of antichains with no further increase in density, yielding the overall density bound of $2 \cdot \sqrt{n}$. □

Note that when the chains and antichains are joined into a Steiner tree as described in the proof, the tree density will always be *exactly* the total number of chains and antichains since a horizontal line near the top of the square will cut all (extended) chains and antichains. Clearly, lower density constructions might be attainable; however, the experimental results of Section 5.1.4 use this simple "joining" construction for Step 7 of the PEEL algorithm.

A result of Hunt and Szymanski [134] shows that the maximum chain or antichain in a pointset can be computed in $O(n \log\log n)$ time. Since PEEL requires $O(\sqrt{n})$ iterations, its time complexity is bounded by $O(n^{\frac{3}{2}} \log\log n)$.

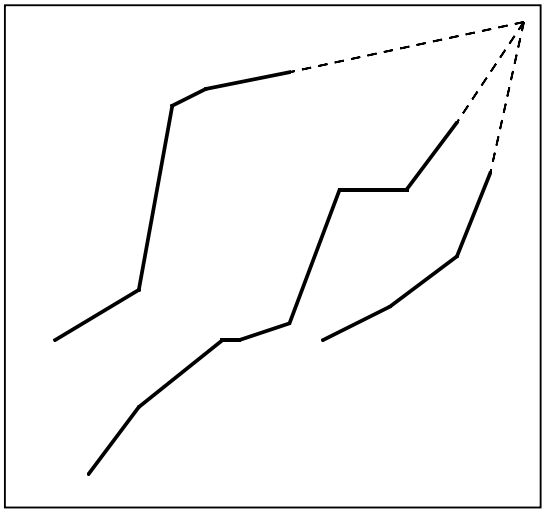


Figure 5.11 Combining chains into a low-density tree.

Cost Bounds

Probabilistic arguments show that on average, COMB and COMB_ST will produce trees with low cost.

Theorem 5.1.9 *For n terminals distributed arbitrarily in the unit square, algorithm COMB constructs a spanning tree with cost $\leq 2\sqrt{2} \cdot \sqrt{n}$.*

Proof: In the COMB construction, the sum of the vertical components of the edges within each strip is bounded by 1 (the height of each strip is one unit). Thus, the sum of the vertical components of all routing tree edges introduced in Step 2 of Figure 5.4 is bounded by $\frac{\sqrt{n}}{\sqrt{2}}$. Furthermore, the vertical components of edges introduced in Step 3 also sum to at most $\frac{\sqrt{n}}{\sqrt{2}}$. To bound the sum of horizontal components, note that if we pick an arbitrary edge from within each strip, these $\frac{\sqrt{n}}{\sqrt{2}}$ edges have total horizontal span bounded by 1. The horizontal components of all tree edges from Step 2 thus contribute at most $\sqrt{2n} - 1$ to the tree cost, and since the edges added in Step 3 have total horizontal span ≤ 1, we obtain the bound of $2\sqrt{2} \cdot \sqrt{n}$. □

Theorem 5.1.10 *For n terminals distributed arbitrarily in the unit square, algorithm COMB_ST constructs a Steiner tree with cost $\leq \sqrt{2n} + 1$.*

Proof: In the COMB_ST construction, the vertical spines contribute at most $\frac{\sqrt{n}}{\sqrt{2}}$ to the tree cost. As in the proof of Theorem 5.1.9, if we pick an arbitrary pair of horizontal edges in each strip, one from either side of the spine, the total cost of these edges is ≤ 1, so the sum of horizontal edge components is at most $\frac{\sqrt{2n}}{2} = \frac{\sqrt{n}}{\sqrt{2}}$. Finally, the horizontal connector which joins the spines (Step 3 of Figure 5.6) has cost ≤ 1, and the desired bound of $\sqrt{2n} + 1$ follows. □

Theorem 5.1.11 *For n terminals distributed arbitrarily in the unit square, algorithm PEEL constructs a Steiner tree with cost $\leq 4 \cdot \sqrt{n}$.*

Proof: According to Theorem 5.1.8, PEEL constructs at most $2 \cdot \sqrt{n}$ chains and antichains, which are extended and then joined to yield a Steiner tree over the net S. Each extended chain or antichain can have cost at most 2, yielding the desired bound. □

Theorem 5.1.12 *For n terminals chosen randomly from a uniform distribution in the unit square, the expected minimum spanning tree cost is $\Theta(\sqrt{n})$.*

Proof: While this claim is a consequence of results in the theory of subadditive functionals in the L_p plane [23, 229], we present the following simple proof. Again, we partition the unit square into an array of n square cells, each of size $\frac{1}{\sqrt{n}}$ by $\frac{1}{\sqrt{n}}$. Recall that the expected number of cells that will contain at least one terminal is $(1 - \frac{1}{e}) \cdot \sqrt{n}$. In any tree $T(N)$, each terminal will have at least one incident tree edge, and this edge must cross the boundary of the cell. It is easy to show that the expected distance from a terminal to the nearest side of its containing cell is lower-bounded by some constant times the length of the side of the cell (in the Manhattan norm, this constant is $\frac{1}{6}$). We therefore have an $\Omega(\sqrt{n})$ bound on the expected total tree cost. Since COMB always yields a spanning tree with cost $O(\sqrt{n})$, the minimum spanning tree cost for a set of n terminals uniformly distributed in the unit square is $\Theta(\sqrt{n})$ on average. □

From these results, we have:

Corollary 5.1.13 *For n terminals chosen randomly from a uniform distribution in the unit square, the algorithms COMB, COMB_ST and PEEL all con-*

struct trees which on average have both density and cost bounded by constants times optimal. □

As noted in Section 5.2, the notion of density is related to the computational geometric concept of "low stabbing number" which seeks spanning trees having few intersections with lines of *any* orientation [46, 85]. Welzl [247] has proved that there always exists a spanning tree with stabbing number $O(\sqrt{n})$. Edelsbrunner et al. [86] have shown that $\Omega(\sqrt{n})$ is a lower bound for the stabbing number of a pointset; Theorems 5.1.3 and 5.1.5 above show that this lower bound holds even when only horizontal and vertical stabbing lines are allowed, and establish an *average* case $\Omega(\sqrt{n})$ density lower bound. The authors of [86] also give three spanning-tree constructions with low stabbing number, trading off between space, stabbing number, and the use of randomization. These methods obtain bounds on stabbing number ranging from $O(n^{\frac{1}{2}+\epsilon})$ to $O(n^{\frac{1}{2}} \cdot \mathit{polylog})$, and typically run in $O(n^3)$ time and $O(n^2)$ space. By contrast, the algorithms above guarantee $O(n^{\frac{1}{2}})$ density, and run in $O(n \log n)$ time and $O(n)$ space. Finally, Agarwal [1] showed that there always exists a *family* of $O(\log n)$ trees such that for an arbitrary given line, *one* of the trees will have a stabbing number of $O(\sqrt{n})$; this family can be computed in time $O(n^{\frac{3}{2}} \cdot \mathit{polylog})$.

5.1.3 Triple Optimization

In VLSI routing it is often desirable to simultaneously minimize more than one objective. However, this is difficult: it is unusual for even two competing criteria to be treated effectively (e.g., the simultaneous tree radius and tree cost minimization of [63]). In this section, we show that the minimum-density objective is "compatible" with existing performance-driven routing objectives; indeed, we may simultaneously address up to three separate routing-tree criteria. Section 5.2 outlines a general scheme for the simultaneous optimization of multiple objectives.

Minimizing Skew, Density, and Total Wirelength

Recall from Chapter 4 that construction of a tree with minimum difference among the various source-sink pathlengths captures both minimum-skew clock routing [19] and global routing with min-max timing constraints. Chapter 4 gave a general interconnection scheme that achieves extremely small pathlength skew, while keeping the total wirelength on average within a constant factor

of optimal, and always bounded by $O(\sqrt{n})$. This clock routing construction, which we refer to as CLOCK1, begins with a forest of n isolated terminals, each of which is considered to be a (trivial) tree. An optimal geometric matching on these n points yields $\frac{n}{2}$ segments, each of which defines a tree with two nodes. A tree is rooted at its balance point, i.e., the point that minimizes the pathlength skew to the leaves of its two subtrees. Trees continue to be paired up by geometric matching of their roots so that at each level of the construction only half as many points are matched as in the previous level. Thus, after $\lceil \log n \rceil$ matching iterations, a complete tree topology is obtained, as described in Section 4.2.2.

In order to construct clock-routing trees with low density, we construct a low-density geometric matching via the following variant of algorithm COMB: partition the net into $\frac{\sqrt{n}}{\sqrt{2}}$ strips of $\sqrt{2n}$ terminals each and connect the terminals of each strip from top to bottom as before (Figure 5.12(a)). However, instead of connecting the bottom terminals of all strips, connect the terminals in a *serpentine* fashion, i.e., alternate between connecting the bottoms and tops of adjacent pairs of strips as shown in Figure 5.12(b). Arguments similar to those above show that this procedure (which we call COMB_SERP) will connect all of the terminals in a single long path topology that has both total cost and overall density simultaneously bounded by $O(\sqrt{n})$ in the worst case.

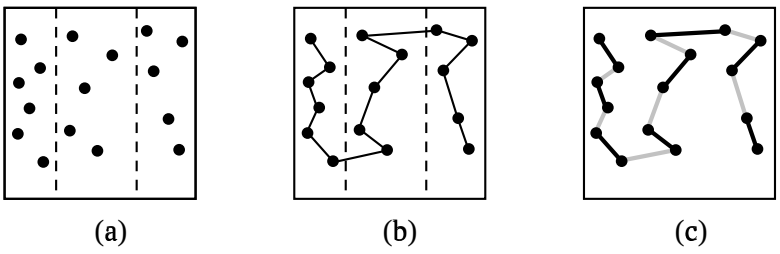


Figure 5.12 (a) Partitioning a net into strips/chains; (b) a serpentine tour with low density, low average cost, and low density; and (c) an embedded geometric matching which also has low density and low average cost.

Taking only every other edge of the tour produced by COMB_SERP will constitute a geometric matching (Figure 5.12(c)) having both total cost and overall density simultaneously bounded by $O(\sqrt{n})$. We may iteratively use such matchings within the CLOCK1 algorithm to yield a clock routing tree that si-

multaneously address three competing objectives: pathlength skew, total wirelength, and density. In particular, the latter two quantities are both bounded on average by constants times optimal.[2]

Minimizing Radius, Density, and Total Wirelength

In Chapter 3 a method was proposed to uniformly trade off total routing tree cost with tree radius (i.e., the longest source-sink pathlength in the tree) and simultaneously optimize both parameters to within constants times optimal in the worst case. This "bounded-radius, bounded-cost" (BRBC) construction [63] starts with a low-cost tour of the net terminals (e.g., a depth-first tour of a minimum spanning tree) and then augments this tour by adding shortest paths to the source from certain regularly spaced locations along the tour. The algorithm returns the shortest-paths tree over the resulting augmented graph, as detailed in Section 3.2.2.

We can combine the minimum-density objective with with the radius/cost tradeoff of the BRBC algorithm to obtain another "triple optimization". Specifically, we may execute the BRBC algorithm with an initial tour L (see Figure 3.10) that is based on, e.g., the COMB_SERP spanning tree (instead of the minimum spanning tree); recall from Section 5.1.3 that the COMB_SERP output has total cost and density both bounded by $O(\sqrt{n})$. Aside from this choice of initial traversal, the remainder of the construction proceeds exactly as the BRBC algorithm (see Figure 3.10).

Given an arbitrary real parameter $\epsilon \geq 0$, the resulting BRBC spanning tree will have radius bounded by $(1+\epsilon)$ from optimum in the *worst case*, cost bounded by $(1+\frac{2}{\epsilon}) \cdot 2\sqrt{2n}$, and density bounded by $(1+\frac{4}{\epsilon \cdot R}) \cdot \sqrt{2n}$, where $R \leq 2$ is the distance from the source to the farthest sink.[3] Note that for any fixed value of ϵ, all three of the above measures (i.e., radius, cost, and density) are on average

[2] This follows from the fact that at each level of the tree construction, only half as many points are being matched as in the previous iteration. Thus, for example, the density of the resulting clock tree will be bounded by $O(\sqrt{n}) + O(\sqrt{\frac{n}{2}}) + O(\sqrt{\frac{n}{4}}) + ... = O(\sqrt{n})$.

[3] The density of the combined COMB_SERP / BRBC construction is bounded by the sum of $\sqrt{2n}$ (the density of the COMB_SERP tree Q) plus the number of shortest paths to the source taken during the traversal of Q in the BRBC algorithm (since any shortest path is necessarily monotone, it cannot contribute more than 1 to the density). The latter quantity is determined by noting that the depth-first tour of Q has length equal to twice the COMB tree cost $2\sqrt{2n}$, and that BRBC adds shortest paths to the source at intervals of at least $\epsilon \cdot R$ along the traversal of Q. Thus, the density of the overall construction is given by $\sqrt{2n} + \frac{2 \cdot 2\sqrt{2n}}{\epsilon \cdot R} = (1 + \frac{4}{\epsilon \cdot R}) \cdot \sqrt{2n}$.

constants times the respective optimal values. Indeed, the radius bound is a constant times optimal in the *worst* case as well.

5.1.4 Experimental Results

We have implemented the COMB_SERP variant of the COMB algorithm, the COMB_ST, and the PEEL algorithms using ANSI C for the Sun environment. Results are presented in Tables 5.1 and 5.2. For each pointset cardinality, each algorithm was executed on 100 pointsets randomly chosen from a uniform distribution in the unit square. Table 5.1 reports the minimum, average, and maximum densities of the resulting trees. Note that for algorithm PEEL, the number of chains and antichains computed by the algorithm is reported; this gives the spanning-tree density when we use the simple joining method described in the proof of Theorem 5.1.8. The tree cost of PEEL will be somewhat higher than shown in Table 5.2, since the data does not include the extra edgelength needed to join the chains together.

The average density of the tree produced by the COMB_SERP algorithm is on par with the density of the simple minimum spanning tree. However, the density of the minimum spanning tree has higher variance, and in the worst case can be as large as $\Omega(n)$. Thus, the COMB or COMB_SERP constructions have practical utility due to their predictable performance. The average density of the trees produced by the COMB_ST algorithm is lower than the average density of the corresponding minimum spanning trees: for example, with signal nets of size 10, COMB_ST yields trees with average density = 3.00, in contrast to average minimum spanning tree density = 3.82. For $n = 10$, this 21% decrease in average density is achieved with a corresponding 21% increase in the tree cost over the MST cost, shown in Table 5.2. There is essentially no variance in the density of the COMB_ST output.

As discussed in Section 5.1.2, for a given net S, any partition of the unit square into an i by j rectangular grid, such that P of the resulting $i \cdot j$ rectangles contain terminals of S (Figure 5.10), induces a lower bound $\lceil \frac{P-1}{i+j-2} \rceil$ on the minimum routing density of S. Recall that a simple version of this lower bound schema schema places $i = \sqrt{n}$ horizontal lines so as to leave at most $\sqrt{n}$ terminals between consecutive lines, and then places $j = \sqrt{n}$ vertical lines using the same criterion. A comparison of the COMB_ST density versus the results of this computational lower bound are given in the rightmost three columns of Table 5.1 (any fractional computational lower bound values are rounded up to the

net size	MST			COMB_SERP		
	min	ave	max	min	ave	max
5	2	2.57	4	2	2.70	3
10	2	3.82	6	3	3.71	4
15	3	4.35	6	3	4.95	5
20	4	4.98	8	4	4.98	5
30	4	5.99	8	6	6.00	6
50	5	7.11	10	7	7.79	8

net size	PEEL			COMB_ST			COMB_ST / LB		
	min	ave	max	min	ave	max	min	ave	max
5	2	2.00	2	2	2.00	2	1.00	1.05	2.00
10	2	3.08	4	3	3.00	3	1.00	1.46	1.50
15	3	3.93	5	3	3.00	3	1.00	1.44	1.50
20	4	4.76	6	4	4.00	4	1.33	1.72	2.00
30	5	5.88	7	5	5.00	5	1.67	1.76	2.50
50	7	7.85	9	6	6.00	6	1.50	1.92	2.00

Table 5.1 Tree density statistics for minimum spanning tree and for the three heuristic constructions. Averages are taken over 100 instances for each net size. The rightmost columns give the ratio of COMB_ST density to the instance-wise computational lower bound of Section 5.1.2.

nearest integer, since density takes on only integer values). This lower bound can be used in assessing algorithm quality on an instance-wise basis.

It is still an open question whether there exists a polynomial-time algorithm that constructs a routing tree with both cost and density bounded by constants times optimal in the *worst* case. It is also unknown whether the MDT problem is NP-complete. The chain-peeling method, PEEL, holds some promise in the sense that there exist examples where it outperforms COMB and COMB_ST by a factor of $\Theta(\sqrt{n})$ (Figure 5.8); it is conjectured that PEEL can be shown to yield worst-case density that is within a small constant factor of optimal. Two closely related conjectures are: (i) that the minimum density of a spanning tree over net S is at least the minimum of the number of chains or the number of antichains needed to cover S; and (ii) the PEEL algorithm will use at most twice the minimum possible number of chains/antichains that cover S.

net size	MST			COMB_SERP		
	min	ave	max	min	ave	max
5	804	1658.39	2554	1010	2154.82	4233
10	1781	2662.36	3462	2287	3682.77	4766
15	2296	3224.41	4045	2663	4692.03	6465
20	2766	3789.89	4558	3819	5265.67	6567
30	4107	4651.00	5403	5524	6841.33	8529
50	5190	5945.47	6668	7542	8708.12	10177

net size	PEEL			COMB_ST		
	min	ave	max	min	ave	max
5	758	1495.57	2481	1063	2260.09	3229
10	1595	2776.06	4080	2307	3224.01	3974
15	2562	3721.27	5071	3143	4216.83	4941
20	2871	4720.69	6350	3692	4823.63	5649
30	4873	6318.27	8085	5594	6570.46	7740
50	7447	9298.73	11629	7070	8029.99	8945

Table 5.2 Tree cost statistics.

5.2 MULTI-WEIGHTED GRAPHS

While previous chapters focused on optimizing a single design criterion (e.g., wirelength, signal skew, tree radius), secondary routing optimization goals might entail congestion avoidance, jog minimization, and circuit reliability. In Section 5.1.3 we observed that certain combinations of multiple objectives may be simultaneously optimized. Continuing in this direction, this section develops a general framework of multi-weighted graphs, where multiple *competing* objectives are optimized simultaneously under a smooth designer-controlled tradeoff. This framework enables effective routing in graph-based regimes, e.g., in building block design, in FPGAs, around obstacles, etc., and is also applicable to many other areas of combinatorial optimization (e.g., traveling salesman, matching, and partitioning). This work was first described in Alexander and Robins [5, 6, 8].

A multi-weighted graph is a weighted graph where each edge weight is a *vector* rather than a scalar; that is, the graph has several distinct sets of edge weights, corresponding to the various objectives that we seek to optimize. For example, one set of edge weights may represent wirelengths, a second set of

edge weights can represent congestion information, and a third set of edge weights can model jog penalties, etc. Searches in such multi-weighted graphs are guided by a weighted average of the values corresponding to the different competing criteria, relative to given designer-selected tradeoff parameters. Such a framework subsumes, e.g., "alpha-beta" routing, which has been used for jog minimization in circuit design [58, 132].

Let $V = \{v_1, v_2, \cdots, v_n\}$ be a set of vertices and $E \subseteq V \times V$ be a set of $|E| = m$ edges. We define a *k-weighted graph* $G = (V, E)$ to be a weighted graph with a vector-valued weight function $\vec{w} : E \rightarrow \Re^k$. In other words, associated with each edge $e_{ij} \in E$ is a *vector* of k real-valued weights $\vec{w}_{ij} = (w_{ij1}, w_{ij2}, \cdots, w_{ijk})$. Note that ordinary weighted graphs are a special case of k-weighted graphs, with $k = 1$.

Let $\vec{d} = (d_1, d_2, \cdots, d_k)$ be a vector of k real-valued *tradeoff parameters*, where $0 \leq d_i \leq 1$ for $0 \leq i \leq k$, and $\sum_{m=1}^{k} d_m = 1$. From the k-weighted graph $G = (V, E)$ and the tradeoff parameters $\vec{d}$ we construct a new weighted *tradeoff graph* $\widehat{G}(\vec{d}) = (V, E)$ with scalar weight function $w'_{ij} = \vec{d} \cdot \vec{w}_{ij} = \sum_{m=1}^{k} d_m \cdot w_{ijm}$. The tradeoff graph $\widehat{G}$ is an ordinary weighted graph having the same topology as G, but whose single edge weights represent the weighted averages of the multi-weights of G, with respect to the tradeoff-parameters vector $\vec{d}$.

Let $\vec{u} = (1, 1, \ldots, 1)$ be the unit vector, and for a given vector $\vec{z} = (z_1, z_2, \cdots, z_k)$, let $\vec{z}_i = (0, 0, \cdots, 0, z_i, 0, 0, \cdots, 0)$ denote the vector obtained from the vector $\vec{z}$ by using z_i in the i^{th} place, and the rest of the places being zero. Thus, $\vec{u}_i$ denotes the vector consisting of zeros everywhere except the i^{th} place, which will contain a "1". A k-weighted graph G naturally induces k distinct graphs, each with an identical topology but with edge weights restricted to only one of the k components of the weight function $\vec{w}$; these k induced graphs are denoted by $G_m = \widehat{G}(\vec{u}_m)$ for $1 \leq m \leq k$.

We define the minimum spanning tree (MST) for a multi-weighted graph G with respect to the tradeoff parameters $\vec{d}$ as the ordinary MST over the tradeoff graph $\widehat{G}(\vec{d})$ and denote it by $\text{MST}(\widehat{G}(\vec{d}))$. Similarly, we can compute the MST on each of the k induced graphs G_m and we denote these $\text{MST}(G_m)$. For convenience we will use $\overline{\text{MST}}$ to denote the cost of the MST.

As an example of an application of multi-weighted graphs, consider the following cost/performance tradeoff in circuit-board manufacturing. Let $k = 2$ and construct a 2-weighted graph G over n terminals on a circuit board, where w_{ij1} represents the *capacitance* of the edge e_{ij}, and where w_{ij2} represents the

manufacturing cost of that edge (see Figure 5.13). Minimizing the total tree capacitance corresponds to improving circuit performance, but we would rather avoid incurring a huge manufacturing-cost increase in return for only a tiny performance gain. The goal is therefore to trade off these two objectives in a smooth manner. Clearly, $\text{MST}(\widehat{G}((1,0)))$ denotes the tree with the least possible capacitance, while $\text{MST}(\widehat{G}((0,1)))$ denotes the tree that is cheapest to manufacture. On the other hand, $\text{MST}(\widehat{G}((\frac{1}{2},\frac{1}{2})))$ represents the tree that simultaneously optimizes both performance and manufacturing cost, with both objectives being equally important (i.e., $d_1 = d_2 = \frac{1}{2}$).

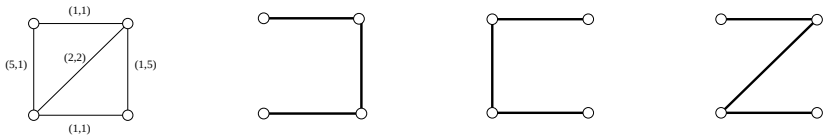


Figure 5.13 (a) A 2-weighted graph G, and MSTs over its two induced graphs: (b) $\text{MST}(\widehat{G}((1,0)))$ with cost $3+7=10$, and (c) $\text{MST}(\widehat{G}((0,1)))$ with cost $7+3=10$; (d) shows the MST over the tradeoff graph: $\text{MST}(\widehat{G}((\frac{1}{2},\frac{1}{2})))$ with cost $4+4=8$.

Given a k-weighted graph G and a parameter vector $\vec{d}$, it would be of interest to bound $\overline{\text{MST}}(\widehat{G}(\vec{d}))$ both from above and below in terms of $\overline{\text{MST}}(G_1)$ through $\overline{\text{MST}}(G_k)$, $\vec{d}$, and n. Below we derive the following upper and lower bounds for metric graphs (i.e., graphs with each weight set satisfying the triangle inequality $w_{ijm} + w_{jkm} \geq w_{ikm}$, $1 \leq i,j,k \leq n$, $1 \leq m \leq k$):

$$\sum_{m=1}^{k} d_m \cdot \overline{\text{MST}}(G_m) \;\leq\; \overline{\text{MST}}(\widehat{G}(\vec{d})) \;\leq\; (n-1) \cdot \sum_{m=1}^{k} d_m \cdot \overline{\text{MST}}(G_m)$$

While the lower bound shown above holds in general, for arbitrary (non-metric) weighted graphs there exists no upper bound strictly in terms of the $\overline{\text{MST}}(G_i)$'s, $\vec{d}$, and n. For metric graphs, the above lower bound is tight, while the upper bound can be improved; for example, we will show a tight upper bound of

$\overline{\text{MST}}(\widehat{G}(\vec{d})) \leq \frac{4}{3} \cdot \sum_{i=1}^{k} d_i \cdot \overline{\text{MST}}(G_i)$ for metric graphs over three nodes. This is significant since most nets in typical VLSI designs contain 3 or fewer pins [104, 162].

Theorem 5.2.105 *For any k-weighted graph G, and tradeoff parameters $\vec{d}$, $\sum_{m=1}^{k} d_m \cdot \overline{\text{MST}}(G_m) \leq \overline{\text{MST}}(\widehat{G}(\vec{d}))$.*

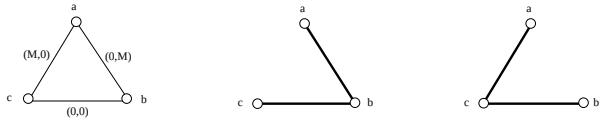
Proof: Consider an arbitrary edge e_{ij} in $\text{MST}(\widehat{G}(\vec{d}))$ with cost $\sum_{m=1}^{k} d_m \cdot w_{ijm}$. If *every* $\text{MST}(G_m)$, $1 \leq m \leq k$, also contains edge e_{ij}, then clearly the cost of edge e_{ij} in all k trees is $\sum_{m=1}^{k} w_{ijm}$, and the cost of this edge scaled by the tradeoff parameters $\vec{d}$ is $\sum_{m=1}^{k} d_m \cdot w_{ijm}$, which is equal to the cost of this edge in $\text{MST}(\widehat{G}(\vec{d}))$. Clearly, if *all* of the k $\text{MST}(G_m)$'s contained the same edges as $\text{MST}(\widehat{G}(\vec{d}))$, then equality holds and the theorem is true. On the other hand, if $\text{MST}(\widehat{G}(\vec{d}))$ contains an edge that is not in $\text{MST}(G_m)$ for some $1 \leq m \leq k$, the cost of $\text{MST}(\widehat{G}(\vec{d}))$ relative to $\sum_{m=1}^{k} d_m \cdot \overline{\text{MST}}(G_m)$ can only increase. □

Ideally we would like to bound the MST cost of arbitrary multi-weighted graphs in terms of only the costs of the $\text{MST}(G_i)$'s, $\vec{d}$, and n. Unfortunately, this property does not hold in general.

Theorem 5.2.106 *For any k-weighted graph G over n vertices, and tradeoff parameters $\vec{d}$, the tradeoff graph cost $\overline{\text{MST}}(\widehat{G}(\vec{d}))$ cannot be bounded from above by any function of only $\overline{\text{MST}}(G_i)$'s, $\vec{d}$, n, and k.*

Proof: Consider the 2-weighted complete graph $G = (V, E)$ over $n = 3$ nodes, where $k = 2$. Fix $\vec{d}$ by setting $0 < d_1, d_2 < 1$. Let M be some large constant, $V = \{a, b, c\}$, and $E = V \times V$, with $w_{ab1} = 0$, $w_{bc1} = 0$, $w_{ac1} = M$ and let $w_{ab2} = M$, $w_{bc2} = 0$, $w_{ac2} = 0$ (see Figure 5.14). Observe that $\overline{\text{MST}}(G_1) = \overline{\text{MST}}(G_2) = 0$, $k = 2$, $n = 3$, d_1, and d_2 are all constants. On the other hand, $\overline{\text{MST}}(\widehat{G}) = min(d_1 \cdot M, d_2 \cdot M)$, which can be made arbitrarily large for any fixed $\vec{d}$ by making M large enough. Since any expression in terms of only constants must also be constant, $\overline{\text{MST}}(\widehat{G})$ can not be bounded from above by any function strictly in terms of $\overline{\text{MST}}(G_1)$, $\overline{\text{MST}}(G_2)$, k, n, and $\vec{d}$. □

The negative result of Theorem 5.2.106 only applies to non-metric graphs; we now derive a general upper bound for metric graphs.



(a) (b) (c)

Figure 5.14 An example showing that $\overline{\text{MST}}(\widehat{G}(\vec{d}))$ cannot be bounded from above by any function strictly in terms of $\overline{\text{MST}}(G_i)$'s, $\vec{d}$, n, and k: (a) The 2-weighted graph G; (b) $\text{MST}(G((1,0)))$ has cost 0; (c) $\text{MST}(G((0,1)))$ has cost 0. On the other hand, $\text{MST}(G((\frac{1}{2},\frac{1}{2})))$ has cost $\frac{M}{2}$, which can be arbitrarily large.

Theorem 5.2.104 *If G is any metric k-weighted graph G over n vertices, and tradeoff parameters $\vec{d}$, then* $\overline{\text{MST}}(\widehat{G}(\vec{d})) \leq (n-1) \cdot \sum_{m=1}^{k} d_m \cdot \overline{\text{MST}}(G_m)$

Proof: Consider an arbitrary edge e_{ij} in $\text{MST}(\widehat{G}(\vec{d}))$ and its cost, $\sum_{m=1}^{k} d_m \cdot w_{ijk}$. Consider the m'^{th} element in this summation, and the corresponding MST of $G_{m'}$. $\text{MST}(G_{m'})$ spans vertices v_i and v_j, but does not necessarily contain the edge e_{ij}. However, a path must exist in $\text{MST}(G_{m'})$ from v_i to v_j, denoted $\text{minpath}_{MST(G_{m'})}(i,j)$, with cost denoted by $\text{dist}_{MST(G_{m'})}(i,j)$. By metricity, $w_{ijm'} \leq \text{dist}_{MST(G_{m'})}(i,j)$. Therefore:

$$\begin{aligned}
\text{cost of edge } e_{ij} \text{ in } \text{MST}(\widehat{G}(\vec{d})) &= \sum_{m=1}^{k} d_m \cdot w_{ijm} \\
&\leq \sum_{l=1}^{k} d_m \cdot \text{dist}_{MST(G_{m'})}(i,j) \\
&\leq \sum_{l=1}^{k} d_m \cdot \overline{\text{MST}}(G'_m)
\end{aligned}$$

Since e_{ij} is an arbitrary edge of $\text{MST}(\widehat{G}(\vec{d}))$, this holds for *all* $n-1$ edges in $\text{MST}(\widehat{G}(\vec{d}))$. Thus, $\overline{\text{MST}}(\widehat{G}(\vec{d}))) \leq (n-1) \cdot \sum_{m=1}^{k} d_m \cdot \overline{\text{MST}}(G_m)$. □

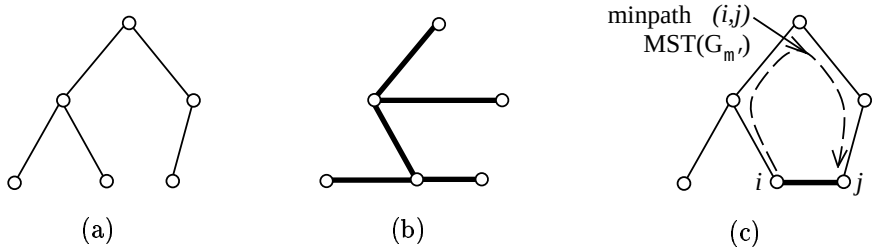


Figure 5.15 A general upper bound in the metric case: for $\overline{\text{MST}}(\widehat{G}(\vec{d}))$ in terms of $\overline{\text{MST}}(G_i)$'s, $\vec{d}$, n, and k: (a) depicts $\text{MST}(G_m)$; (b) depicts $\text{MST}(\widehat{G}(\vec{d}))$; and (c) shows how the cost of the m^{th} weight component of each e_{ij} can be bounded by $d_m \cdot \overline{\text{MST}}(G_m)$.

And finally, in the case of three-node metric graphs, the upper bound can be tightened somewhat:

Theorem 5.2.104 *For 2-weighted metric graphs with three nodes, and any scaling vector $\vec{d} = (d_1, d_2)$, the following holds:*

$$d_1 \cdot \overline{\text{MST}}(G_1) + d_2 \cdot \overline{\text{MST}}(G_2) \leq \overline{\text{MST}}(\widehat{G}(\vec{d})) \leq \frac{4}{3} \cdot [d_1 \cdot \overline{\text{MST}}(G_1) + d_2 \cdot \overline{\text{MST}}(G_2)]$$

Proof: Let $G = (V, E)$ be a complete 3-node 2-weighted graph, with edge weights (a, x), (b, y), and (c, z). Let $\vec{d} = (d_1, d_2)$ be an arbitrary constant vector, such that $0 \leq d_1, d_2 \leq 1$, and $d_1 + d_2 = 1$ (see Figure 5.16(i)).

The lower bound $d_1 \cdot \overline{\text{MST}}(G_1) + d_2 \cdot \overline{\text{MST}}(G_2) \leq \overline{\text{MST}}(\widehat{G}(\vec{d}))$ holds by Theorem 5.2.105. Assume without loss of generality that $a \leq b \leq c$, which implies that $\overline{\text{MST}}(G_1) = a + b$. The following three possibilities must be considered, corresponding to the cases (i) $x, y \leq z$, (ii) $x, z \leq y$, and (iii) $y, z \leq x$:

Case (i): assume $x, y \leq z$, which implies that $\overline{\text{MST}}(G_2) = x + y$ (see Figure 5.16(ii)). Thus:

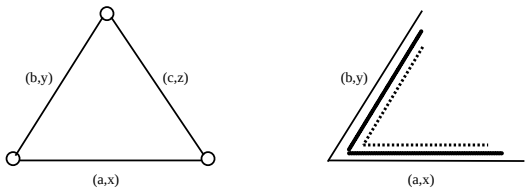


Figure 5.16 A tighter upper bound for 3-terminal nets; (a) a 3-node 2-weighted graph, with edge weights (a, x), (b, y), and (c, z), and (b) topology of the three spanning trees $\mathrm{MST}(G_2)$ (inner), $\mathrm{MST}(G_1)$ (middle) and $\mathrm{MST}(\widehat{G}(\vec{d}))$ (outermost) corresponding to case (i).

$$\begin{aligned} d_1 \cdot a + d_2 \cdot x &\leq d_1 \cdot c + d_2 \cdot z \quad \text{and} \\ d_1 \cdot b + d_2 \cdot y &\leq d_1 \cdot c + d_2 \cdot z \end{aligned}$$

$$\begin{aligned} \text{Now} \quad \overline{\mathrm{MST}}(\widehat{G}(\vec{d})) &= d_1 \cdot a + d_2 \cdot x + d_1 \cdot b + d_2 \cdot y \\ &= d_1 \cdot (a + b) + d_2 \cdot (x + y) \\ &= d_1 \cdot \overline{\mathrm{MST}}(G_1) + d_2 \cdot \overline{\mathrm{MST}}(G_2) \end{aligned}$$

and the theorem holds.

Case (ii): Assume $x, z \leq y$, which implies that $\overline{\mathrm{MST}}(G_2) = x + z$. Let $\overline{\mathrm{G}} = d_1 \cdot \overline{\mathrm{MST}}(G_1) + d_2 \cdot \overline{\mathrm{MST}}(G_2)$, and consider the three possible subcases illustrated in Figure 5.17.

- **Subcase (ii)a:** assume $\mathrm{MST}(\widehat{G}(\vec{d}))$ contains the "a/x" and "b/y" edges (See Figure 5.17(ii)a). Then:

$$\begin{aligned} \overline{\mathrm{MST}}(\widehat{G}(\vec{d})) &= d_1 \cdot (a + b) + d_2 \cdot (x + y) \\ &\leq d_1 \cdot (a + b) + d_2 \cdot (x + x + z) \end{aligned}$$

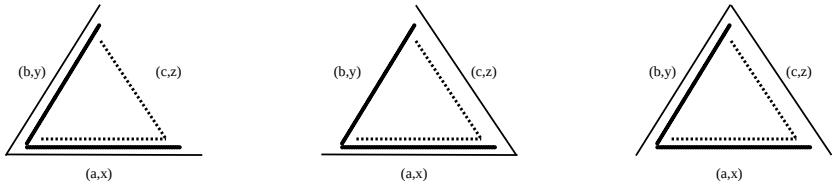


Figure 5.17 Topology of the three spanning trees MST(G_2) (inner), MST(G_1) (middle) and MST($\widehat{G}(\vec{d})$) (outer) corresponding to case (ii)a, case (ii)b, and (ii)c.

$$\begin{aligned} &= d_1 \cdot \overline{\text{MST}}(G_1) + d_2 \cdot \overline{\text{MST}}(G_2) + d_2 \cdot x \\ &= \overline{\text{G}} + d_2 \cdot x \end{aligned}$$

- **Subcase (ii)b:** assume MST($\widehat{G}(\vec{d})$) contains the "a/x" and "c/z" edges (See Figure 5.17(ii)b). Then:

$$\begin{aligned} \overline{\text{MST}}(\widehat{G}(\vec{d})) &= d_1 \cdot (a + c) + d_2 \cdot (x + z) \\ &\leq d_1 \cdot (a + a + b) + d_2 \cdot (x + z) \\ &= d_1 \cdot \overline{\text{MST}}(G_1) + d_1 \cdot a + d_2 \cdot \overline{\text{MST}}(G_2) \\ &= \overline{\text{G}} + d_1 \cdot a \end{aligned}$$

- **Subcase (ii)c:** assume MST($\widehat{G}(\vec{d})$) contains the "b/y" and "c/z" edges (See Figure 5.17(ii)c). Then:

$$\begin{aligned} \overline{\text{MST}}(\widehat{G}(\vec{d})) &= d_1 \cdot (b + c) + d_2 \cdot (y + z) \\ &\leq d_1 \cdot (b + a + b) + d_2 \cdot (x + z + z) \\ &= d_1 \cdot \overline{\text{MST}}(G_1) + d_1 \cdot b + d_2 \cdot \overline{\text{MST}}(G_2) + d_2 \cdot z \\ &= \overline{\text{G}} + d_1 \cdot b + d_2 \cdot z \end{aligned}$$

Now, since $\mathrm{MST}(\widehat{G}(\vec{d}))$ is a *minimum* spanning tree, it is the minimum of the bounds produced by subcases (ii)a, (ii)b, and (ii)c:

$$\begin{aligned}\overline{\mathrm{MST}}(\widehat{G}(\vec{d})) &= \overline{\mathrm{G}} + \min(d_2 \cdot x, d_1 \cdot a, d_1 \cdot b + d_2 \cdot z) \\ &\leq \overline{\mathrm{G}} + \frac{1}{3} \cdot (d_2 \cdot x + d_1 \cdot a + d_1 \cdot b + d_2 \cdot z) \\ &= \frac{4}{3} \cdot \overline{\mathrm{G}}\end{aligned}$$

Case (iii): Assume $y, z \leq x$, which implies that $\overline{\mathrm{MST}}(G_2) = y + z$. Again, let $\overline{\mathrm{G}} = d_1 \cdot \overline{\mathrm{MST}}(G_1) + d_2 \cdot \overline{\mathrm{MST}}(G_2)$, and consider the three possible subcases corresponding to whether $\mathrm{MST}(\widehat{G}(\vec{d}))$ contains (3a) "a/x" and "b/y", (3b) "a/x" and "c/z", or (3c) "b/y" and "c/z", which are handled using similar arguments to those in case (ii) above.

The bound $\overline{\mathrm{MST}}(\widehat{G}(\vec{d})) \leq \frac{4}{3} \cdot [d_1 \cdot \overline{\mathrm{MST}}(G_1) + d_2 \cdot \overline{\mathrm{MST}}(G_2)]$ holds in each one of the three possible cases (i), (ii), and (iii). The example $a = \epsilon, x = 2 - \epsilon, b = c = y = z = 1$ (where ϵ is an arbitrarily small value) shows that this bound is tight. □

For 4-terminal nets the general upper bound of Theorem 5.2.104 implies that the constant in the upper bound is $n - 1 = 3$. However, an exhaustive computer search indicates that this constant is actually $\frac{3}{2}$. We therefore conjecture that the upper bound can be tightened considerably. Tighter bounds on the combined MST cost over multi-weighted graphs were recently developed in [105].

5.3 PRESCRIBED-WIDTH ROUTING

This section addresses prescribed-width routing, which seeks a least-cost source-destination path having a given minimum width; this problem arises in the routing of, e.g., circuit boards, or where thermal constraints, blockages, or congestion induce a continuously costed routing region. We describe an approach which optimally solves prescribed-width routing using an efficient network flow formulation. This method departs from conventional shortest-path or graph search based methods, in that it not only handles regions with solid polygonal obstacles but also generalizes to arbitrary cost maps which may arise in modeling incomplete or uncertain knowledge of the routing region. The approach was

originally proposed to address path planning for a mobile agent in a general environment[4] and was subsequently extended in [129, 130] to solve Plateau's classic problem on minimum surfaces.

We focus on prescribed-width routing subject to two practical extensions: (i) the need to incorporate uncertainty into the formulation, and (ii) the requirement of an error-tolerant solution. These respectively yield the notion of a general cost function in a given region, along with the notion of a minimum width path. The algorithmic approach to prescribed-width routing in regions with arbitrary cost functions employs a general combinatorial approach involving network flows [67]. The crucial observation is that a minimum-cost *path* which *connects* two locations s and t corresponds to a minimum-cost *cut-set* which *separates* two other locations s' and t'. The prescribed-width path is obtained by applying efficient network flow algorithms to exploit this duality between connecting paths and separating sets.

This approach is guaranteed to find optimal solutions to the minimum-cost prescribed-width routing formulation which we define below. The algorithm runs in polynomial time, and can be implemented in $O(d^2 \cdot N^2 \cdot \log N)$ time where N is the number of nodes in a discrete mesh representation of the region, and d is the prescribed-path width. Experimental results confirm that this method can find optimal prescribed-width paths where current combinatorial methods are prohibitively expensive, and where variational or gradient heuristics only return locally optimum solutions.

5.3.1 Prescribed-Width Routing by Network Flows

We begin by establishing notation and terminology. The development focuses on the connection-separation duality which motivates the network flow approach.

[4]The robotics literature has addressed a problem related to prescribed-width routing, namely the *motion planning problem* for an autonomous mobile agent, which asks for a minimum-cost feasible path between a source and a destination in a configuration space [39, 66, 161, 211]. The cost of a given solution may depend on many factors, including distance traveled, time or energy expended, and hazard probabilities encountered along the path (see Canny [39], Latombe [161], or Mitchell [185] for a survey).

Problem Formulation

We say that a subset of the plane is *simply connected* if it is homeomorphic to a disk (i.e., contains no holes); a subset of the plane is *compact* if it is closed and bounded.

Definition 5.3.1 *A* region *is a simply-connected, compact subset of* $\Re^2$.

Given a region R, we know by the Jordan curve theorem [68] that the boundary B of R partitions the plane into three mutually disjoint sets: B itself; the interior of R; and the exterior of R. We consider the problem of computing a path in R from source S to destination T, where S and T are disjoint connected subsets of the boundary B. A *path* is defined as follows:

Definition 5.3.2 *Given a region R with boundary B, a* path *between two disjoint connected subsets $S \subset B$ and $T \subset B$ is a non self-intersecting continuous curve $P \subseteq R$ which connects some point $s \in S$ to some point $t \in T$.*

Clearly the path P partitions R into three mutually disjoint sets: (i) the set of points of R lying strictly on the left side of P, which we denote by R_l (we assume that P is oriented in the direction from s toward t); (ii) the set of points of R lying on the right side of P, denoted by R_r; and (iii) points of P itself. This is illustrated in Figure 5.18. It is possible for at most one of R_l and R_r to be empty, and this happens exactly when P contains a subset of B between S and T. R_l (resp. R_r) is empty if $P \supseteq B_l$ (resp. $P \supseteq B_r$), where B_l and B_r respectively denote the subsets of the boundary B lying clockwise and counterclockwise between S and T, i.e., $B_l = (B \cap R_l) - (S \cup T)$ and $B_r = (B \cap R_r) - (S \cup T)$.

As noted above, our goal is to optimally solve prescribed-width routing when two practical constraints are incorporated: an *arbitrary* cost function defined over the region, and a *minimum width* path requirement. An arbitrary cost function corresponds to a *general region*, which is in many ways more realistic than a region consisting only of solid rectilinear obstacles, e.g. a continuous cost function models naturally constraints due to thermal and manufacturing considerations, etc. In this scenario, each point in the region will have an associated *weight*, or *cost* of traversal, corresponding to temperature. (In what follows, we use the terms "weight" and "cost" synonymously.) Multiple objectives may also be captured via this formulation: if the path to the destination

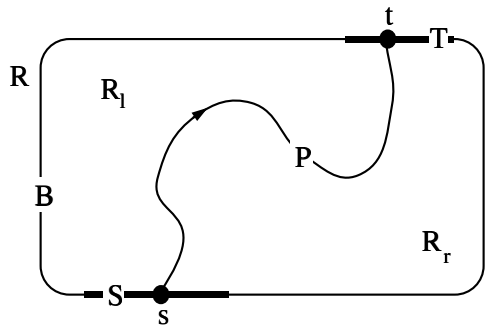


Figure 5.18 A path P between two points $s \in S$ and $t \in T$, where S and T are disjoint subsets of the boundary B of a region R.

must be short, then the weight function will take into account, say, both temperature level and the travel distance. Note that this formulation subsumes the binary cost function of a basic region with solid polygonal obstacles (cost = 1) and free space (cost = 0).

Formally, given a region R, we define a weight function $w : R \rightarrow \Re^+$ such that each point $s \in R$ has a corresponding positive weight $w(s)$. The *cost*, or *weight*, of a path $P \subseteq R$ is defined to be the integral of w over P. Optimal path planning entails minimizing this path integral. To find a minimum-cost path $P \subseteq R$ between two points on the boundary of R, one might guess that Dijkstra's shortest path algorithm [67] provides a natural solution. However, application of Dijkstra's algorithm relies on an implicit assumption that the solution can be cast as an *ideal* path, i.e., a path of *zero* width. The relevance of this caveat becomes clear when we consider the second extension to the basic formulation - the requirement of a *minimum-width* path solution.

We require the path to have a minimum width everywhere, e.g., corresponding to the width of a metal trace on a circuit board, or to the number of parallel wires in a bus. With this in mind, we define a *prescribed-width* path to be one which maintains a given minimum width. In general, the optimum path of width d_1 cannot be obtained by simply widening or narrowing the optimum path of width d_2.

We now establish the relationship between a prescribed-width requirement and the concept of *d-separation* [112]. In the following, we use $ball(x, d)$ to denote

the closed ball of diameter d centered at x, i.e., the set of all points at distance $\frac{d}{2}$ or less from x.

Definition 5.3.3 *Given two disjoint subsets S and T of the boundary of a region R, a set of points $\hat{P} \subseteq R$ is a* width-d path *between S and T if there exist $s \in S$, $t \in T$ and a path P connecting s to t such that $\hat{P} \supseteq \bigcup_{x \in P}\{ball(x,d) \cap R\}$, i.e., $\hat{P}$ contains the intersection of R with any disk of* diameter d *centered about a point of P.*

Just as the path P between S and T will partition R into R_l, R_r, and P, the width-d path $\hat{P} \subseteq R$ between S and T also partitions R into three sets: (i) the set of points $\bar{R}_l = ((R - \hat{P}) \cap R_l) \cup B_l$, that is, the union of the left boundary B_l and all points in R that are to the left of $\hat{P}$; (ii) the set of points $\bar{R}_r = ((R - \hat{P}) \cap R_r) \cup B_r$; and (iii) the pointset $\hat{P}$ itself. We now obtain the definition of a d-separating path (see Figure 5.19):

Definition 5.3.4 *Given two disjoint subsets S and T of the boundary of a region R, a set of points $\bar{P} \subseteq R$ is a* d-separating path *between S and T if $\bar{P}$ is a width-d path such that any point of $\bar{R}_l$ is distance d or more away from any point of $\bar{R}_r$.*

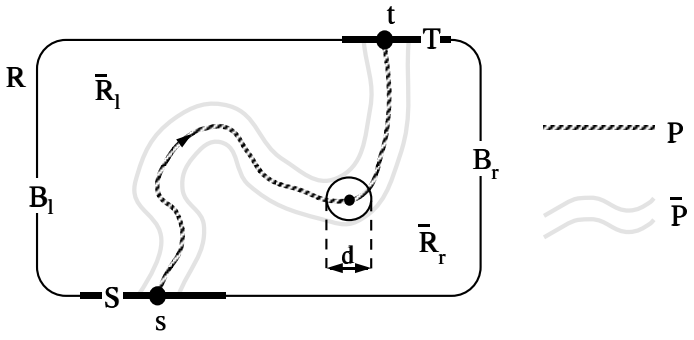


Figure 5.19 A d-separating path $\bar{P}$ of width d between two points $s \in S$ and $t \in T$ of the boundary of a region R. Here $\bar{R}_l$ is separated from $\bar{R}_r$ by a distance of d.

A d-separating path $\bar{P}$ between S and T is a *minimal d-separating path* between S and T if no subset of $\bar{P}$ satisfies the preceding definition. Because all points in R have positive cost and because we are interested in minimum-cost paths, the following discussion refers only to minimal d-separating paths. While the treatment thus far assumed a continuous routing region, in VLSI the region is typically discretized relative to a given fixed grid. Thus, we will assume a fixed-grid representation $\tilde{R}$ of the region R, where the cost of a path is defined to be the sum of the weights of the nodes covered by the path. Similarly, the notion of d-separation also naturally extends to the discrete grid:

Definition 5.3.5 *Given a region R, a* discrete *d-separating path $\tilde{P}$ in the gridded region $\tilde{R}$ is the subset of the gridpoints of $\tilde{R}$ that is contained in some d-separating path $\bar{P}$ in R (Figure 5.20).*

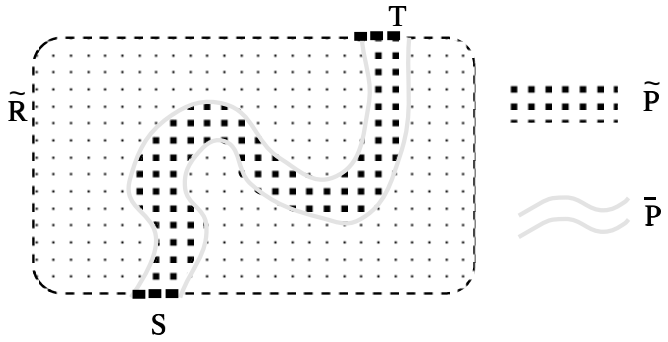


Figure 5.20 A discretized representation $\tilde{R}$ of a region R, and a discrete d-separating path $\tilde{P}$ in $\tilde{R}$. Note that $\tilde{P}$ is the set of lattice points covered by the continuous d-separating path $\bar{P}$ in R.

As before, a discrete d-separating path is *minimal* if no subset of it satisfies this definition. Analogously to the continuous case, a discrete d-separating path partitions the gridded region into two subsets, such that each gridpoint from one partition is a distance of at least d units away from any gridpoint in the other partition. We therefore have the following problem formulation:

Prescribed-Width Path (PWP) Problem: Given a weighted gridded region $\tilde{R}$ with boundary $B \subset \tilde{R}$, a source $S \subseteq B$, a destination $T \subseteq B$, and a width d, find a discrete d-separating path $\tilde{P} \subseteq \tilde{R}$ between S and T which has minimum cost.

Although PWP is a very natural problem formulation, it cannot be efficiently solved by traditional methods. Recall that in an n-node edge-weighted graph $G = (V, E)$ with identified source $v_0 \in V$, the k^{th} phase of Dijkstra's algorithm, $k = 1, \ldots, n$, finds another node v_k for which the shortest pathlength d_{0k} in G is known; we know the optimum s-t pathlength when $v_k = t$. Although the PWP formulation above assumes a node-weighted G, we may easily obtain an edge-weighted graph (for all $v \in V$, add $\frac{w(v)}{2}$ to the weight of each edge incident to v) to which we may apply Dijkstra's algorithm. However, this transformation is correct only for computing the optimal zero-width path: Dijkstra's algorithm relies on the fact that d_{ij} can never be strictly less than $min_k(d_{ik} + d_{kj})$, but this may not hold when paths have non-zero width.

In the special case where the cost function is binary, Dijkstra's algorithm is applicable via the following well-known technique [161]: augment the region by growing each obstacle (as well as the region boundary) isotropically by $\frac{d}{2}$ units, then set the weight of each node in the free area to some constant, while the weight of any node in an area covered by an obstacle is set to infinity. A minimum-cost prescribed-width path in such an augmented region would correspond to the center P of the d-separating path $\tilde{P}$ that we seek (Figure 5.19). Unfortunately, this simple transformation fails for arbitrary weight functions: a general region has no solid "obstacles" which can be "grown" in such a fashion.

A Network Flow Based Approach

To solve the prescribed-width path problem, we use ideas from network flows in continua [128].[5] We first review several key concepts from the theory of network flows [96, 163]. A *flow network* $\eta = (N, A, s, t, c, c')$ is a directed graph with node set N; a set of directed arcs $A \subseteq N \times N$; a distinguished *source* node $s \in N$ and a distinguished *sink* node $t \in N$; an *arc capacity* function $c : A \rightarrow \Re^+$ which specifies the capacity $c_{ij} \geq 0$ of each arc $a_{ij} \in A$; and a *node capacity* function $c' : N \rightarrow \Re^+$ which specifies the capacity $c'_i \geq 0$ of each node

[5] Mitchell [186] also extends the ideas of flows in continua, but in a very different way. The results in [186] develop a theory of flows in polyhedral domains, with a view to such practical applications as motion planning of *many* agents through a congested region (i.e., "rapid deployment"), etc.

$n_i \in N$. To handle undirected graphs, we may replace each undirected arc a_{ij} by two directed arcs a_{ij} and a_{ji}, each having capacity c_{ij}.

A *flow* in η assigns to each arc a_{ij} a value ϕ_{ij} with the constraint that $0 \leq \phi_{ij} \leq c_{ij}$. An arc a_{ij} is called *saturated* if $\phi_{ij} = c_{ij}$. We insist on *flow conservation* at every node except s and t, and we require that the flow through each node n_j does not exceed the capacity of that node:

$$\sum_i \phi_{ij} \;=\; \sum_k \phi_{jk} \;\leq\; c'_j \qquad n_j \neq s,t$$

A node n_j is called *saturated* if $\sum_i \phi_{ij} = c'_j$. Since flow is conserved at every node, the total amount of flow from the source must be equal to the total flow into the sink; we call this quantity the *value* Φ of the flow:

$$\Phi = \sum_i \phi_{si} \;=\; \sum_j \phi_{jt}$$

A flow with the maximum possible value is called a *maximum flow*. An *s-t cut* in a network is a set (N', A') of nodes $N' \subseteq N$ and arcs $A' \subseteq A$ such that every path from s to t uses at least one node of N' or at least one arc of A'. The *capacity* $c(N', A')$ of a cut is the sum of the capacities of all nodes and arcs in the cut. A classical result of linear programming duality states that the maximum flow value is equal to the minimum cut capacity; this is the *max-flow min-cut* theorem [96]:

Theorem 5.3.6 *Given a network $\eta = (N, A, s, t, c, c')$, the value of a maximum s-t flow is equal to the minimum capacity of any s-t cut. Moreover, the nodes and arcs of any minimum s-t cut are a subset of the saturated nodes and saturated arcs in some maximum s-t flow.*

Recall that any *s-t* path will separate, i.e., cut, R_l from R_r. In particular, an inexpensive *s-t* path will correspond to an inexpensive cut between two appropriately chosen nodes s' and t'. Since a subset of the nodes and arcs saturated by the maximum s'-t' flow will yield this s'-t' cut, it is natural for us to derive the desired *s-t* path via a maximum-flow computation in a network

where capacities correspond to travel costs in R. The remainder of this section describes how this is accomplished.

To transform prescribed-width routing in a region R into an instance of network flow, we first superpose a mesh network topology over R, then assign node weights in this network according to the weighting function $w : R \to \Re^+$. This yields a network that corresponds to the underlying PWP instance.

We guarantee a *minimum-width* path solution by ensuring that any separating node set in the mesh topology satisfies the prescribed width-d requirement. Toward this end, we define the *d-neighborhood* of a node v in the mesh to be the set of all nodes at distance d or less units away from v, and we then modify the mesh topology by uniformly connecting each node to all other nodes in its d-neighborhood, where d is the prescribed path width. The resulting network is called a *d-connected* Mesh, and has the property that no nodeset of width less than d is a d-separating set. An illustration of this construction for $d = 2$ is given in Figure 5.21. We note that the concept of a d-neighborhood was first investigated by Gomory and Hu [128].

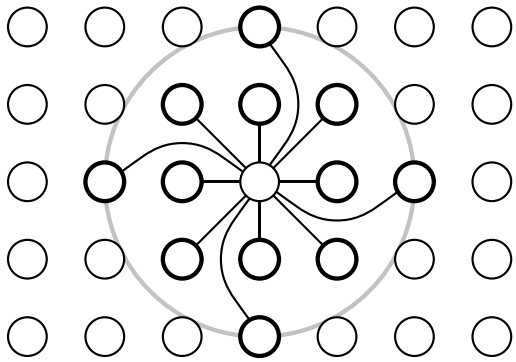


Figure 5.21 A node and its d-neighborhood ($d = 2$).

Finally, we choose nodes s' and t' such that the minimum s'-t' cut is forced to lie along some path between s and t. We accomplish this by making s' and t' respectively into a source and a sink, then connecting each to a contiguous set of nodes corresponding to part of the boundary of the original region R. This completes the transformation; Figure 5.22 gives a high-level illustration of the construction.

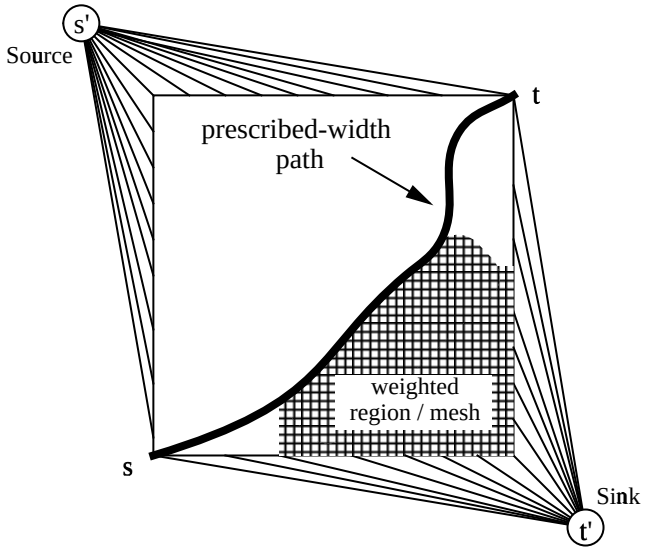


Figure 5.22 A prescribed-width path problem instance transformed into a network flow instance.

Observe that up to this point, we have converted a prescribed-width path instance into an undirected, *node-capacitated* (node-weighted) flow instance. However, network flow algorithms typically assume that the input is an *arc-capacitated* network (with infinite node capacities). Therefore, in order to use a standard maximum flow algorithm, we must transform an instance having both node and arc capacities into an equivalent arc-capacitated maximum flow instance. To accomplish this, we use the standard device of splitting each node $v \in N$ with weight $w(v)$ into two unweighted nodes v' and v'', then introducing a directed arc from v' to v'' with capacity $w(v)$. Also, each arc $(u, v) \in A$ of the original network is transformed into two infinite-capacity directed arcs (u'', v') and (v'', u'). Thus, each arc (v', v'') of the resulting directed network will, when saturated, contribute the original node weight $w(v)$ to the minimum cut value. This transformation is illustrated in Figure 5.23 [115]. The overall size of the network increases by only a constant factor via this last transformation, i.e., the final directed arc-capacitated network will have only $2|N|$ nodes and $|N| + 2|A|$ arcs. Therefore, a maximum flow computation in the transformed network will be asymptotically as fast as in the original network.

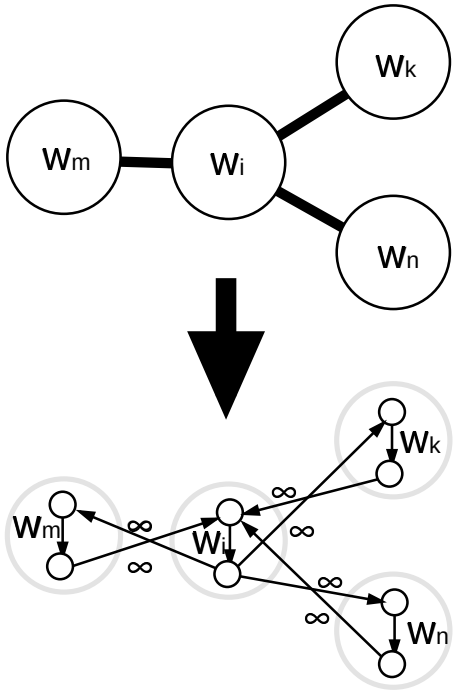


Figure 5.23 Transformation of a node- and arc-capacitated flow network to an arc-capacitated flow network: arc capacities c_{ij} remain infinite, while original node capacities (node weights) c_i' induce directed arc capacities in the transformed network.

Note that a maximum flow in the arc-capacitated transformed graph corresponds to a minimum arc-cut in the transformed graph (by the max-flow min-cut theorem). This in turn corresponds to a minimum node-cut in the original graph since the transformation preserves minimal cutset costs. The "width" of the cut can be no less than d since the connection of each node to all nodes in its d-neighborhood guarantees that any separating node set will have the prescribed width. A formal summary of the algorithm, which we call d-PATH, is given in Figure 5.24.

The max-flow min-cut theorem [96] and the existence of efficient algorithms for maximum flow (e.g., [67, 96]) together imply the following:

d-**PATH:** Finding a prescribed-width path in a weighted region
Input: Region R, weight function $w : R \rightarrow \Re^+$, width d, grid size g, source s and destination t on boundary of R **Output:** A discrete d-separating path $\bar{P} \subseteq R$ connecting s and t
Create a d-connected mesh topology η of size $g \times g$ over R with all arc capacities set to ∞ **Assign** node weights (capacities) in η according to weight function w **Set** boundary node weights (capacities) to ∞ **Transform** node/arc-capacitated network η into arc-capacitated network η' **Add** source node s' and sink node t' to η' **Connect** s' to B_l, the boundary nodes of R, clockwise from s to t **Connect** t' to B_r, the boundary nodes of R, clockwise from t to s **Set** capacities of all arcs adjacent to s' or t' to ∞ **Compute** maximum s'-t' flow in η' **Output** all nodes incident to arcs in the minimum s'-t' cut of η'

Figure 5.24 The d-PATH Algorithm: Finding a prescribed-width path of minimum cost in an arbitrary weighted region, i.e., an optimal solution to the PWP problem.

Theorem 5.3.7 *The d-PATH method of Figure 5.24 outputs an optimal solution to the PWP problem in time polynomial in size of the mesh representation of the region R.*

A Test Implementation

There are numerous algorithms for computing maximum flows in networks [3, 96, 128], and we have used an existing implementation of Dinic's network flow algorithm [111]. Starting with an empty flow, the Dinic algorithm iteratively augments the flow in stages; the optimal flow solution is achieved when no flow augmentation is possible. Each stage starts with the existing flow and attempts to "push" as much flow as possible along shortest paths from the source to the sink in a residue network wherein each arc has capacity equal to the difference between its original capacity and its current flow value. After the current flow has been thus augmented, newly saturated arcs are removed and the process iterates. Since there can be at most $|N| - 1$ such stages, each requiring time at most $O(|A| \cdot |N|)$, the total time complexity of the Dinic algorithm is $O(|A| \cdot |N|^2)$.

If we have a total of $|N|$ nodes in the mesh graph, the time complexity of the Dinic algorithm is $O(|N|^3)$. In practice, more efficient flow algorithms are available [3]. For example, by using the network flow algorithm of [110], we obtain the following:

Theorem 5.3.8 *For a given prescribed path width d, the d-PATH method solves the PWP problem in $O(d^2 \cdot |N|^2 \cdot \log \frac{|N|}{d^2})$ time, where $|N|$ is the number of nodes in the mesh representation of the region R.*

Proof: Each node in the mesh induced by the method has no more than d^2 adjacent arcs, so that $|A| = O(d^2 \cdot |N|)$. The network flow algorithm of [110] operates within time $O(|A| \cdot |N| \cdot \log(\frac{|N|^2}{|A|}))$. The overall time complexity of d-PATHk is therefore $O(d^2 \cdot |N|^2 \cdot \log \frac{|N|}{d^2})$. □

The time complexity of d-PATH is $O(|N|^2 \cdot \log |N|)$ for any fixed d, and may be further reduced in cases where the region cost function may only take on values from a fixed, bounded range. In this case, we may apply the maximum flow algorithm of [3] to obtain an overall time complexity of $O(|N|^2)$ for the d-PATH algorithm.

5.3.2 Simulation Results

The d-PATH implementation uses ANSI C code to transform an arbitrary prescribed-width routing instance into a maximum-flow instance; we then use the Fortran-77 Dinic code of [111] to compute the flow, and invoke Mathematica [248] to draw the resulting path. The implementation was tested on three classes of PWP instances: uniformly weighted regions, regions with polygonal obstacles, and smooth randomly-costed regions. For each of these input classes, the boundary of the region is a rectangle, and we look for a width-d path connecting s and t which are respectively in the top left and bottom right corners of the region. With each instance, we tested various values of d.

A uniformly weighted region has all node weights equal to the same constant. In such an instance we expect the solution path to resemble a straight line between s and t, with the straightness of the line improving as the mesh resolution and the width d both increase. Experimental results confirm this behavior.

The test regions with polygonal obstacles were populated by polygons of random sizes, located throughout the region. Nodes in the clear areas are uniformly

assigned a small constant weight, while nodes inside the obstacles have infinite weight. In such a region, changing the prescribed width d may dramatically affect the optimum path topology with respect to the obstacles, since long detours may be required in order to avoid narrow passages between objects. This phenomenon was indeed confirmed by the experiments, as illustrated in Figure 5.25.

Finally, the methodology was tested on randomly-costed regions, using a mesh resolution of 100 by 100 nodes and a range of d values. Each random region instance was generated as follows. All nodes in the mesh were initially assigned a weight of zero, except for a small random subset of the nodes which were each given a large random positive weight. Then, a weight redistribution step was iteratively used to increment each node's weight by a small random fraction of the total weight of its immediate neighbors until a smooth randomly-costed region was obtained. Figure 5.26 depicts typical d-PATH output for the PWP problem in a random region. Areas of greater weight are denoted by darker shades, and areas of smaller weight are depicted by lighter densities. The optimum width-d path is highlighted in black. Even though the Dinic algorithm is not ideal for a mesh topology, typical running times used to generate and solve all of the above classes of instances are on the order of at most a few minutes on a SUN SPARC IPC (15.7 MIPS).

Chief among future research goals is improvement of the time complexity of the network flow computation; substantial improvement is likely since the mesh is a highly regular and symmetric network that admits a concise representation. Additional research might also address more general path planning issues, such as (i) use of hierarchical approaches as a heuristic speedup, and (ii) addressing the case where the endpoints of the path are not necessarily on the boundary of the region.

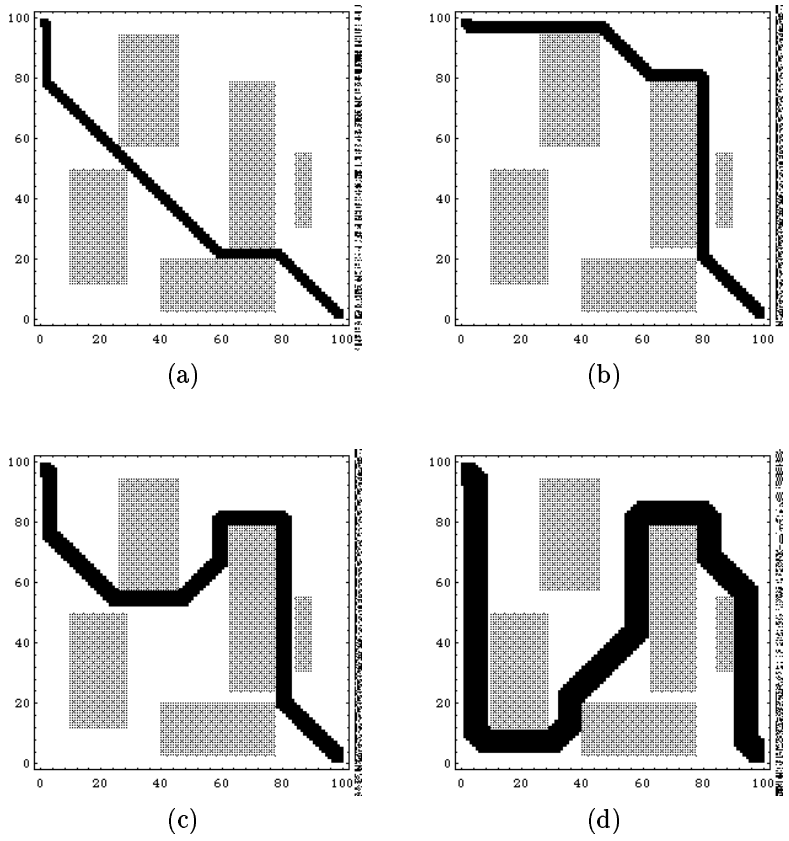


Figure 5.25 Prescribed-width paths in a region with polygonal obstacles. Note that the topology of the solutions changes as the prescribed width d is increased. The solutions shown correspond to widths (a) $d = 3$, (b) $d = 4$, (c) $d = 5$, and (d) $d = 6$.

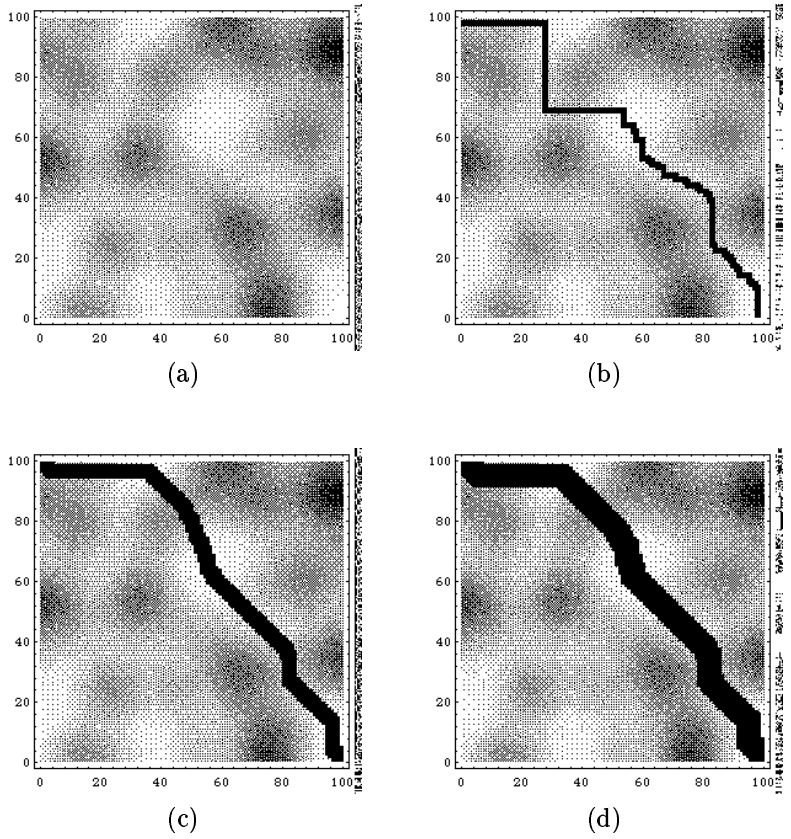


Figure 5.26 A randomly generated smooth region and its prescribed-width path solutions: (a) the region itself, and solutions corresponding to widths (b) $d = 2$, (c) $d = 5$, and (d) $d = 8$.

A

APPENDIX: SIGNAL DELAY ESTIMATORS

In this Appendix, we first describe the basic theory behind several efficient delay estimates, particularly the relationship between the moment representation and the system response in both the time-domain and the transform domain. We then give the formal basis of the Elmore and two-pole delay approximations. The second part of the Appendix describes a series of experimental investigations which characterize the accuracy and fidelity of the linear, Elmore, and two-pole delay approximations.

A.1 BASICS

The relationships among the moment representation, the Laplace transform of the response, and the time-domain response are discussed, e.g., in [180]. The following provides a brief review.

In a linear system, the transfer function $H(s) = \frac{V_{out}(s)}{V_{in}(s)}$ gives the relationship between the output response $V_{out}(s)$ and the input response $V_{in}(s)$. The system transfer function $H(s)$ is related to the impulse response $h(t)$ by the Laplace transform

$$H(s) = \int_0^\infty e^{-st} h(t) dt.$$

The transfer function for any linear system can be expressed as a ratio of polynomials in s, that is to say,

$$H(s) = K \frac{1 + a_1 s + a_2 s^2 + a_3 s^3 + ...}{1 + b_1 s + b_2 s^2 + b_3 s^3 + ...} \tag{A.1}$$

where K is the DC (zero-frequency) gain. The i^{th} moment of a linear system is defined to be

$$M_i = \frac{1}{i!}\int_0^\infty t^i h(t)dt = \frac{(-1)^i}{i!}H^{(i)}(0) \tag{A.2}$$

where $H^{(i)}(0)$ is the i^{th} derivative of $H(s)$ at $s = 0$.

Assuming $v_{out}(0) = 0$, the Laplace transform of the derivative of the output voltage response for a unit step input is $v'_{out}(t) \Leftrightarrow sV_{out}(s) = s \cdot \frac{1}{s}H(s) = H(s) \Leftrightarrow h(t)$. Therefore, the transfer function can also be written as

$$H(s) = \int_0^\infty e^{-st}v'_{out}(t)dt$$

Expanding e^{-st} into a Maclaurin series,

$$H(s) = \int_0^\infty v'_{out}(t)dt - \frac{s}{1!}\int_0^\infty tv'_{out}(t)dt + \frac{s^2}{2!}\int_0^\infty t^2v'_{out}(t)dt - \frac{s^3}{3!}\int_0^\infty t^3v'_{out}(t)dt$$

plus higher-order terms, and identifying the integral quantities as moments M_0, M_1, M_2, M_3 etc. from Equation (A.2) yields

$$H(s) = (M_0 - sM_1 + s^2M_2 - s^3M_3 + ...) \tag{A.3}$$

Therefore, the moments can also be defined as

$$M_i = \frac{1}{i!}\int_0^\infty t^i v'_{out}(t)dt \tag{A.4}$$

The moments of any system can be calculated using the definitions given in Equation (A.2) or in Equation (A.4), or by comparing with Equation (A.3). Applying the definition of moments in Equation (A.4) to $H(s)$, we obtain

$$\begin{aligned}
H(s) &= K(1 + a_1s + a_2s^2 + a_3s^3 + ...)(1 + b_1s + b_2s^2 + b_3s^3 + b_4s^4 + ...)^{-1} \\
&= K(1 + a_1s + a_2s^2 + a_3s^3 + ...) \\
&\quad \cdot \left(1 - b_1s + (b_1^2 - b_2)s^2 - (b_3 + b_1^2 - 2b_1b_2)s^3 + ...\right) \\
&= K[1 + s(a_1 - b_1) + s^2(a_2 - a_1b_1 + b_1^2 - b_2) \\
&\quad + s^3(a_3 - a_2b_1 + a_1(b_1^2 - b_2) - b_3 - b_1^3 + 2b_1b_2) + ...]
\end{aligned}$$

which yields

$$\begin{aligned} M_0 &= K \\ M_1 &= K(b_1 - a_1) \\ M_2 &= K(a_2 - a_1 b_1 + b_1^2 - b_2) \\ M_3 &= K\left(b_1^3 + b_3 - 2b_1 b_2 - a_3 + a_2 b_1 - a_1(b_1^2 - b_2)\right) y \end{aligned} \tag{A.5}$$

A.1.1 Elmore Delay

Elmore delay [87] is defined to be the first moment (M_1) of the system impulse response, i.e., the coefficient of s in the system transfer function $H(s)$. This is a first-order approximation of the delay and corresponds to a single dominant pole approximation of the response.

The $ABCD$ parameters of a distributed $RLCG$ transmission line are [81]:

$$\begin{pmatrix} V_1(s) \\ I_1(s) \end{pmatrix} = \begin{pmatrix} \cosh(\theta h) & Z_0 \sinh(\theta h) \\ \frac{1}{Z_0} sinh(\theta h) & \cosh(\theta h) \end{pmatrix} \begin{pmatrix} V_2(s) \\ I_2(s) \end{pmatrix} \tag{A.6}$$

where $\theta = \sqrt{(r + sl)(g + sc)}$; r, l, c, g are resistance, inductance, capacitance, conductance per unit length and h is the length of the line.[1] $V_1(s), I_1(s)$ and $V_2(s), I_2(s)$ correspond to the voltage and current at, respectively, the input port and output port of the interconnect line. $Z_0 = \sqrt{\frac{R+sL}{G+sC}}$ is the characteristic impedance of the distributed line, where R, L, C, G are the total resistance, inductance, capacitance and conductance of the distributed line of length h.

A case of special interest in the literature is the open-ended line, for which load impedance $Z_L \to \infty$, implying $I_2(s) = 0$. For this case, we may substitute the appropriate expressions for θ and Z_0 in the above expression, and obtain the transfer function for the open-ended distributed RLC line (i.e., with $g = 0$) as:

$$H(s) = \frac{1}{1 + \frac{RC}{2}s + (\frac{(RC)^2}{24} + \frac{LC}{2})s^2 + (\frac{(RC)^3}{720} + \frac{RLC^2}{12})s^3 + ...}$$

[1] The line parasitics will depend on the three-dimensional process geometry. Techniques for parasitic extraction are beyond the scope of this discussion.

Applying the Elmore delay definition to the distributed RLC line yields

$$T_{ED} = \frac{RC}{2}.$$

To obtain response and delay estimates in an interconnect tree, Rubinstein et al. [205] considered the *main path*, i.e., the unique path in the tree, between the source s_0 and the sink s_k of interest. Using the Elmore delay expression, [205] developed the following delay model for RC interconnect trees:

$$T_{ED} = r_d C_{s_0} + \sum_{\forall i \in MP(s_0, s_k)} r_{e_i} \left(\frac{c_{e_i}}{2} + C_i \right)$$

where C_i is the (sub)tree rooted at node i, e_i is the unique parent edge of node i when the tree is rooted at the source, and $MP(s_0, s_k)$ is the main path between source and sink. We use r_d to indicate the driver on-resistance at the source s_0; r_{e_i} and c_{e_i} respectively denote the lumped resistance and capacitance of the edge e_i. This is the same formula given in Section 3.1.2.

The above expression does not take into account the inductance of the interconnect line, which becomes more significant as feature sizes decrease, layout dimensions increase, and operating frequencies increase. Therefore, the delay estimation may not be accurate and the trees which minimize Elmore delay could be suboptimal. A paper by Kahng and Muddu [146] provides a more accurate methodology for response and delay calculations in general $RLCG$ interconnects.

A.1.2 Two-Pole Analysis

As noted in Chapter 3, many routing tree techniques are motivated by the simple nature of the Elmore delay model; indeed, the second part of this Appendix demonstrates that Elmore delay is a reasonable representation of delay in practice. However, Elmore delay does not afford any measure of delay at a given threshold voltage [205]. Methods which calculate more than one dominant pole from the moments of the system will lead to a second- or higher-order approximation of the response, with improved delay estimates. Two-pole methods by Horowitz [127], Zhou et al. [258, 256] and Kahng and Muddu [145, 149] have been used for simulating interconnect trees. In such methods, the transfer function of the system is approximated using the two dominant poles.

The transfer function of the system considering two dominant poles is

$$H(s) = \frac{k_1}{s - s_1} + \frac{k_2}{s - s_2}$$

where s_1, s_2 are the poles and k_1, k_2 are the coefficients corresponding to the poles. Assuming a step input, $v_{in}(t) = V_0 u(t)$ (where V_0 is the magnitude of the step input voltage), $V_{in}(s) = \frac{V_0}{s}$, and the output voltage is given by

$$V_{out}(s) = V_0 \frac{H(s)}{s}.$$

Applying partial fractions and taking the inverse Laplace transform:

$$v_{out}(t) = V_0 \cdot \left(-(\frac{k_1}{s_1} + \frac{k_2}{s_2}) + \frac{k_1}{s_1} e^{s_1 t} + \frac{k_2}{s_2} e^{s_2 t} \right).$$

Using the boundary conditions $v_{out}(t \to \infty) = V_0$ and $v'_{out}(t = 0) = 0$, one can solve for s_1, s_2, k_1 and k_2, i.e.,

$$-(\frac{k_1}{s_1} + \frac{k_2}{s_2}) = 1 \quad \text{and} \quad k_1 + k_2 = 0$$

and the time-domain response is

$$v_{out}(t) = V_0 \cdot \left(1 + \frac{k_1}{s_1} e^{s_1 t} + \frac{k_2}{s_2} e^{s_2 t} \right)$$

The delay at any given threshold (e.g., 50% or 90% of V_0) can then be computed from the response.

The poles s_1, s_2 and the coefficients k_1, k_2 corresponding to the poles are functions of only the first and second moments M_1, M_2, i.e., by applying the definition of moments to the two-pole transfer function, we have

$$\begin{aligned} (\frac{k_1}{s_1^2} + \frac{k_2}{s_2^2}) &= M_1 \\ (\frac{k_1}{s_1^3} + \frac{k_2}{s_2^3}) &= M_2 \end{aligned}$$

which yields

$$\begin{aligned} s_{1,2} &= \frac{2}{-M_1 \pm \sqrt{4M_2 - 3M_1^2}} \\ k_1 &= -k_2 = -\frac{1}{\sqrt{4M_2 - 3M_1^2}}. \end{aligned}$$

Note that the poles of the system should be always in the left half of the $s-$plane for stable systems. The voltage response can be calculated for both real and complex poles as follows:

Case 1: Real Poles.

From the above equations,

$$k_1 = -\frac{s_1 s_2}{s_2 - s_1}$$

and

$$k_2 = \frac{s_1 s_2}{s_2 - s_1}$$

The voltage response for real poles is

$$v_{out}(t) = V_0(1 - \frac{s_2}{s_2 - s_1}e^{s_1 t} - \frac{s_1}{s_1 - s_2}e^{s_2 t})$$

Case 2: Complex Poles.

Since the poles are complex we can express them in the form $s_1 = -\alpha + \jmath\beta$ and $s_2 = -\alpha - \jmath\beta$.

The voltage response is

$$\begin{aligned} v_{out}(t) &= V_0 \left[1 - \frac{s_2}{s_2 - s_1}e^{s_1 t} - \frac{s_1}{s_1 - s_2}e^{s_2 t}\right] \\ &= V_0 \left[1 - \frac{e^{-\alpha t}}{2}\left((1 + \frac{\alpha}{\jmath\beta})e^{\jmath\beta t} + (1 - \frac{\alpha}{\jmath\beta})e^{-\jmath\beta t}\right)\right] \\ &= V_0 \left[1 - e^{-\alpha t}(\cos(\beta t) + \frac{\alpha}{\beta}\sin(\beta t))\right] \\ &= V_0 \left[1 - \frac{\sqrt{\alpha^2 + \beta^2}}{\beta}e^{-\alpha t}\sin(\beta t + \rho)\right] \end{aligned}$$

where $\rho = \tan^{-1}(\frac{\beta}{\alpha})$ and

$$\alpha = \frac{M_1}{2(M_1^2 - M_2)}$$

$$\beta = \frac{\sqrt{3M_1^2 - 4M_2}}{2(M_1^2 - M_2)}$$

Before discussing the relative merits of the Elmore and two-pole approximations, two issues should be noted. First, to obtain the response $v_{out}(t)$ accurately up to second order, both the first and second moments need to be calculated exactly. Recent two-pole methods such as [256, 258] calculate the first and second moments by replacing the off-path admittance by the sum of total subtree capacitance; this provides a correct approximation only up to the coefficient of s in the subtree admittance. Thus, such methods underestimate the subtree impedance, and the response obtained is actually a lower bound on the true response. Consequently, the associated delay estimate is an upper bound on the actual delay. To calculate the second moment exactly, the admittance of off-path subtrees should be correctly approximated up to the second degree, i.e., up to the coefficient of s^2. The work of Kahng and Muddu [148, 149] achieves this, and hence affords exact expressions for the first and second moments for use in the two-pole methodology.

Second, the accuracy of the moment-based approximation of the system response will depend on the lumped segment models used for modeling the distributed interconnect lines. Traditionally, uniformly lumped segment models, e.g., ladders of **L**, **T** or **Π** circuits, have been used to model interconnect lines. For such *uniform* representations, the moments are perfectly captured only as the number of segments used approaches infinity, which is computationally unreasonable. Nevertheless, to achieve an accurate estimate of the system response, previous works on the two-pole approach (e.g., [205] or [256]) suggest using k uniform segments to model each interconnect line. For example, the work of [65] divides each line into $25\mu m$ segments and then models each segment using a fixed **L** type RLC circuit. For large layout dimensions, the value of k can be quite large. In [145, 147], Kahng and Muddu develop non-uniform equivalent circuits which exactly match the first and second moments of a distributed RLC line, i.e., such circuits have a transfer function which exactly matches that of the distributed RLC line up to the coefficient of s^2. These circuits have only two or three segments, but give the same simulation accuracy as an infinite number of uniform segments. Beyond the improvements in accuracy, the computational savings are substantial.[2]

[2]The idea of non-uniform equivalent circuits for interconnect modeling dates back to Rajput [199]. Gerzberg [108] surveyed different non-uniform models and proposed a model in

A.2 ACCURACY AND FIDELITY

Ideally, a routing algorithm will compute and optimize signal delays according to a detailed circuit simulation such as SPICE. However, since SPICE runtimes are generally too costly for this purpose, simpler delay estimates are used. Among the available estimators of voltage response and signal delay in interconnect structures, the most useful one for *efficient* construction of "optimal-delay routing trees" has not been determined. This section centers on a *fidelity* property which is necessary for any delay estimator to be effective in routing tree design. Studies by Boese et al. [30, 32] of the relative accuracies and relative fidelities of the linear, distributed RC, distributed RLC, and SPICE-computed delay approximations show that the Elmore distributed RC delay approximation has surprisingly high fidelity with respect to SPICE3e2. This is the motivation for direct optimization of Elmore delay in Sections 3.3 and 4.3.

The traditional minimum-cost Steiner tree objective, beyond minimizing wiring area, corresponds to a *lumped-capacitance* model of delay (i.e., signal delay is proportional to total tree capacitance, which is proportional to tree cost). Several methods discussed in Chapters 3 and 4 employ a *linear* model of delay: sink delays are proportional to source-sink path lengths, and the minimum-radius criterion results. The traditional lumped-capacitance approximations becomes less accurate as technology scales, since smaller wire geometries imply that resistive effects of the interconnect become more dominant (cf. the discussion of "resistance ratio" vis-a-vis Table 3.1 in Chapter 3). Thus, distributed RC delay approximations such as Elmore delay are of interest. Because greater system speeds and layout areas can expose inductive effects on delay, two-pole distributed RLC delay approximations such as [256] are also of interest. Each of these approximations will be more accurate than the linear or lumped-capacitance approximations, while requiring less computation time than SPICE. A two-pole estimate will be strictly intermediate between Elmore delay (a single-pole estimate) and SPICE in terms of both computation time and quality of the estimate.

---

which the RC values in each segment are in geometric progression; the "Uniform Distributed RC" (URC) line model in SPICE is derived from Gerzberg's model. The concept of non-uniform equivalent circuits has also been employed in many other areas, e.g., O'Brien and Savarino [188] obtain a non-uniform Π segment model for driving-point admittance at a gate output. Sakurai [206] has observed that the use of a non-uniform equivalent circuit is not always appropriate, since asymmetry implies that a correct response cannot be predicted when the line is driven bidirectionally. However, in the routing tree delay optimizations discussed here, source and sinks are fixed and the direction of signal flow is known.

A.2.1 Accuracy

The traditional measure of a delay estimator is its *accuracy*, which may vary with the circuit technology and the specifics of a net (for instance, the number of terminals it contains, or the size and aspect ratio of its bounding box). Table A.1 indicates the accuracy of the linear, Elmore and two-Pole models in predicting critical-sink delay for each of the interconnect technologies described in Table 3.1. Note that "Two-Pole" indicates a particular corrected version [187] of the two-pole simulator developed in [256]; this simulator was used to obtain many of the experimental results in Chapter 3. For each of the three estimators, the table gives the average ratio of SPICE delay to the estimated delay, and also shows the consistency of this ratio in terms of its standard deviation. This "SPICE-centric" analysis of the data reflects the use of SPICE estimates as "actual delay".[3] Each entry in the table represents an average over 100 random nets, with the source and critical sink chosen randomly. An MST routing was used so that the comparison would be for relatively good (but not necessarily optimal) routing solutions. This is because finding optimal-delay routing solutions according to SPICE is not computationally feasible. In all cases, the ratio of SPICE to Linear delay is the least consistent, having the largest standard deviation. On the other hand, the average ratios of SPICE to Elmore or Two-Pole delay also seem inaccurate (perhaps due to choice of delay threshold and other aspects of the experimental methodology [32]), and inconsistent (witness the high standard deviations).

Interestingly, similar experiments in [32] show that for *maximum* sink delay, the ratio between SPICE and both the Two-Pole and Elmore estimators is very consistent, with standard deviations less than 4% for 6-sink nets in all technologies. Thus, precomputed correction factors could compensate for any inaccuracy in these estimators. However, Table A.1 shows that the standard deviation of the accuracy ratio for *critical-sink* delay is consistently above 15%. This decreased consistency may indicate that the traditional net-dependent maximum delay objective is more "forgiving" of errors in the delay estimate than newer path-dependent, i.e., critical-sink, delay objectives.

[3]The SPICE delay estimation methodology [32] uses constant unit resistance and capacitance values for each interconnect technology. The root of the routing tree is driven by a resistor connected to the source, in order to separate driver attributes from the interconnect simulation. For the Two-Pole (i.e., corrected [256]) and SPICE (i.e., SPICE3e2) simulators, every interconnect segment is broken into uniform segments, each at most 1/100th the length of the layout dimension, connected in series. Sink loads are modeled by pure capacitive loads derived using minimum-size transistors. All delay estimates use the 50% rise time delay criterion. For the Two-Pole and SPICE estimate, time steps of 0.005ns for the IC technologies and 0.05ns for the MCM technology were used.

Accuracy of Linear, Elmore and Two-Pole Delay Estimates for Critical-Sink Delay					
		$\|S\| = 4$		$\|S\| = 7$	
	Delay Ratio	average	std dev	average	std dev
IC1	SPICE/Linear†	–	28.4%	–	32.7%
	SPICE/Elmore	0.72	13.5%	0.69	15.4%
	SPICE/2-Pole	1.27	13.5%	1.23	15.4%
	2-Pole/Elmore	0.568	0.45%	0.566	0.22%
IC2	SPICE/Linear†	–	33.9%	–	38.8%
	SPICE/Elmore	0.74	16.1%	0.70	17.8%
	SPICE/2-Pole	1.30	15.9%	1.23	17.8%
	2-Pole/Elmore	0.572	0.92%	0.568	0.45%
IC3	SPICE/Linear†	–	34.9%	–	40.3%
	SPICE/Elmore	0.78	16.0%	0.72	17.8%
	SPICE/2-Pole	1.36	15.7%	1.27	17.9%
	2-Pole/Elmore	0.574	1.39%	0.571	0.80%
MCM	SPICE/Linear†	–	57.1%	–	61.6%
	SPICE/Elmore	0.69	20.5%	0.65	25.1%
	SPICE/2-Pole	1.20	20.8%	1.14	25.2%
	2-Pole/Elmore	0.568	0.96%	0.566	0.44%

Table A.1 Accuracy of the Linear, Elmore and Two-Pole estimates for critical-sink delay. Standard deviations are reported as a percent of the average ratio. (†) Linear delay is defined as the source/sink pathlength; because this is a distance rather than a time, there is no SPICE/Linear "ratio". However, the percent standard deviation of this quotient is well-defined since it is independent of units.

A.2.2 Fidelity

The key observation in [30] is that precise accuracy or consistency are *not* really required of the delay estimates used to construct routing trees. In fact, the only practical requirement is that an estimator has a high degree of *fidelity*: an optimal or near-optimal solution according to the estimator should also be nearly optimal according to actual (SPICE-computed) delay. Boese et al. proposed a measure of fidelity vis-a-vis an exhaustive enumeration of all possible routing solutions: first rank *all* spanning tree topologies[4] by the given delay

[4]By an early theorem of Cayley [92], there are $|S|^{|S|-2}$ distinct spanning tree topologies for any given net S.

model, then rank the topologies again by SPICE delay, and find the average over all topologies of the absolute value of the difference between the two rankings. This measure of fidelity corresponds to a standard rank-ordering technique used in the social sciences [15].

		Linear vs SPICE		Elmore vs SPICE	
	Topologies	$\|S\| = 4$	$\|S\| = 5$	$\|S\| = 4$	$\|S\| = 5$
IC1	Best	2.30	16.3	0.54	5.9
	5 Best	2.52	18.1	1.02	7.2
	All	2.43	17.0	0.92	8.0
IC2	Best	2.52	19.4	0.58	6.4
	5 Best	2.66	20.2	0.99	7.2
	All	2.44	16.9	0.94	7.9
IC3	Best	2.60	19.8	0.58	5.6
	5 Best	2.68	20.9	0.93	6.5
	All	2.43	16.5	0.93	7.7
MCM	Best	3.04	24.6	0.72	5.1
	5 Best	2.81	24.4	0.89	4.7
	All	2.33	15.7	0.89	7.1

Table A.2 Average difference in rankings of topologies, in terms of 50% delay to a given random critical sink in each net. The sample consists of 50 random nets of each cardinality, with 50% rise time delay criterion. The number of topologies considered for each net is $4^{(4-2)} = 16$ for $|S| = 4$, and $5^{(5-2)} = 125$ for $|S| = 5$.

Table A.2 depicts this measure of fidelity for critical-sink delay and 4- and 5-terminal signal nets, using the linear and Elmore delay estimators. The table shows the average rank difference for the topology which has lowest delay according to the estimator; the average difference for the five topologies which have lowest delay according to the estimator; and the average difference in ranking over all topologies. Note that with the linear delay model, ties are broken in favor of trees with lower total wirelength. Ties also occur for SPICE-computed delay because of the finite time step used, and are also broken according to total wirelength. Since the accuracy ratio between Elmore and Two-Pole is nearly constant, the fidelity values for Two-Pole are essentially identical to those for Elmore, and are omitted from the table.

It is clear that Elmore delay has very high fidelity for the critical-sink criterion.[5] For example, with 5-terminal nets and IC3 technology parameters, optimal critical-sink topologies under Elmore delay average only 5.6 rank positions (out of 125) away from optimal according to SPICE. Kim, Owens and Irwin [157] have similarly established the fidelity of Elmore delay for circuit design: they plotted Elmore- versus SPICE-computed delays for a suite of 209 different place/route solutions of the same ripple-carry adder circuit, and also found a very high correlation between the two delay measures. The work of Vlach et al. [245] gives a theoretical motivation for this correlation, based on the concept of group delay.

Rank	IC1	IC2	IC3	MCM
1	1.000	1.000	1.000	1.000
2	1.006	1.003	1.002	1.001
3	1.011	1.005	1.005	1.001
4	1.014	1.006	1.006	1.002
5	1.016	1.007	1.006	1.003
6	1.017	1.007	1.006	1.004
7	1.026	1.012	1.007	1.005
8	1.040	1.021	1.014	1.005
9	1.074	1.046	1.036	1.014
10	1.160	1.138	1.120	1.047
11	1.180	1.155	1.134	1.049
12	1.224	1.207	1.182	1.058
13	1.246	1.218	1.191	1.060
14	1.288	1.254	1.225	1.064
15	1.306	1.269	1.233	1.066
16	1.327	1.309	1.283	1.103
17	1.351	1.344	1.326	1.427
18	1.380	1.376	1.354	1.431
19	1.417	1.427	1.413	1.475
125	8.04	10.36	10.81	18.34

Table A.3 Average SPICE delay ratios for the top 19 topologies ranked according to SPICE for $|S| = 5$. Values are averaged over 50 random nets and normalized to the average delay of the best topology. Also included is the average ratio for the *worst* topology (rank 125).

[5] Results in [32] show that Elmore delay has nearly perfect fidelity for the "easier" maximum sink delay criterion.

To see the relationship between SPICE rank suboptimality and actual percentage delay suboptimality, Table A.3 shows the average increase in SPICE delay from optimal for the 19 top-ranking topologies, i.e., the 19 lowest SPICE delays for $|S| = 5$. For IC2, the average distance of 6.4 rank positions for the optimal critical sink Elmore delay topology implies an expected difference of approximately 1.6% in actual SPICE-computed delay; for IC3 the distance of 5.6 rank positions implies approximately 0.7% SPICE delay suboptimality; and for MCM a difference of 5.1 rank positions implies 0.4% SPICE delay suboptimality.

One can compose the data in Tables A.2 and A.3 to obtain an estimate of the suboptimality, in terms of SPICE-computed delay, of the Elmore-optimal solution. The more direct measure is to compare SPICE delays of the Elmore-optimal and SPICE-optimal solutions, as shown in Table A.4 for both the critical-sink and maximum sink delay criteria. For critical sink delay and $|S| = 5$, the average SPICE suboptimality of the Elmore-optimal topology is between 3.1% for MCM and 9.9% for IC1. (These estimates are larger than would be inferred from Tables A.2 and A.3 due to the convexity of the relationship between SPICE rank and average SPICE delay.) With regard to the maximum delay criterion, the Elmore delay estimate affords essentially perfect results, in that the Elmore-optimal solution has SPICE delay suboptimality of between 0.1% and 0.2%.

	Critical Sink Delay		Maximum Delay	
Technology	$\|S\| = 4$	$\|S\| = 5$	$\|S\| = 4$	$\|S\| = 5$
IC1	2.9	9.9	0.9	0.1
IC2	3.9	9.6	0.5	0.2
IC3	3.8	7.8	1.3	0.2
MCM	1.9	3.1	0.1	0.1

Table A.4 Average SPICE suboptimality of the Elmore-optimal spanning tree topology (in percent).

REFERENCES

[1] P. K. Agarwal, *Intersection and Decomposition Algorithms for Planar Arrangements*, Cambridge University Press, Cambridge, England, 1991.

[2] P. K. Agarwal and M. T. Shing, *Algorithms for Special Cases of Rectilinear Steiner Trees: Points on the Boundary of a Rectilinear Rectangle*, Networks, 20 (1990), pp. 453–85.

[3] R. K. Ahuja, J. B. Orlin, and R. E. Tarjan, *Improved Time Bounds for the Maximum Flow Problem*, Tech. Rep. CS-TR-118-87, Dept. of Computer Science, Princeton University, 1987.

[4] M. J. Alexander, K. D. Boese, A. B. Kahng, and G. Robins, *A New Greedy Heuristic for the Rectilinear Steiner Arborescence Problem.* unpublished manuscript, January 1994.

[5] M. J. Alexander, J. P. Cohoon, J. L. Ganley, and G. Robins, *An Architecture-Independent Approach to FPGA Routing Based on Multi-Weighted Graphs*, in Proc. European Design Automation Conf., Grenoble, France, September 1994, pp. 259–264.

[6] M. J. Alexander and G. Robins, *An Architecture-Independent Unified Approach to FPGA Routing*, Tech. Rep. CS-93-51, Department of Computer Science, University of Virginia, October 1993.

[7] M. J. Alexander and G. Robins, *High-Performance Routing for Field-Programmable Gate Arrays*, in Proc. IEEE Intl. ASIC Conf., Rochester, NY, September 1994, pp. 138–141.

[8] M. J. Alexander and G. Robins, *A New Approach to FPGA Routing Based on Multi-Weighted Graphs*, in Proc. ACM/SIGDA Intl. Workshop on Field-Programmable Gate Arrays, Berkeley, CA, February 1994.

[9] M. J. Alexander and G. Robins, *New Graph Arborescence and Steiner Constructions for High-Performance FPGA Routing*, Tech. Rep. CS-94-12, Department of Computer Science, University of Virginia, April 1994.

[10] C. ALPERT, J. CONG, A. B. KAHNG, G. ROBINS, AND M. SARRAFZADEH, *Minimum Density Interconnection Trees*, in Proc. IEEE Intl. Symp. Circuits and Systems, Chicago, May 1993, pp. 1865–1868.

[11] C. ALPERT, J. CONG, A. B. KAHNG, G. ROBINS, AND M. SARRAFZADEH, *On the Minimum Density Interconnection Tree Problem*, VLSI Design, 2 (1994), pp. 157–169.

[12] 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.

[13] 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*, in Proc. IEEE Intl. Symp. Circuits and Systems, Chicago, IL, May 1993, pp. 1869–1872.

[14] C. J. ALPERT, T. C. HU, J. H. HUANG, A. B. KAHNG, AND D. KARGER, *Prim-Dijkstra Tradeoffs for Improved Performance-Driven Routing Tree Design.* unpublished manuscript, March 1993.

[15] T. G. ANDREWS, *Methods of Psychology*, John Wiley, New York, 1948.

[16] B. AWERBUCH, A. BARATZ, AND D. PELEG, *Cost-Sensitive Analysis of Communication Protocols*, in Proc. ACM Symp. Principles of Distributed Computing, 1990, pp. 177–187.

[17] B. AWERBUCH, A. BARATZ, AND D. PELEG, *Efficient Broadcast and Light-Weight Spanners.* unpublished manuscript, 1991.

[18] H. BAKOGLU, *Circuits, Interconnections and Packaging for VLSI*, Addison-Wesley, Reading, MA, 1990.

[19] H. BAKOGLU, J. T. WALKER, AND J. D. MEINDL, *A Symmetric Clock-Distribution Tree and Optimized High-Speed Interconnections for Reduced Clock Skew in ULSI and WSI Circuits*, in Proc. IEEE Intl. Conf. Computer Design, Port Chester, NY, October 1986, pp. 118–122.

[20] T. BARRERA, J. GRIFFITH, S. A. MCKEE, G. ROBINS, AND T. ZHANG, *Toward a Steiner Engine: Enhanced Serial and Parallel Implementations of the Iterated 1-Steiner Algorithm*, in Proc. Great Lakes Symp. VLSI, Kalamazoo, MI, March 1993, pp. 90–94.

[21] T. BARRERA, J. GRIFFITH, 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.

[22] J. J. BARTHOLDI AND L. K. PLATZMAN, *A Fast Heuristic Based on Spacefilling Curves for Minimum-Weight Matching in the Plane*, Inf. Proc. Letters, 17 (1983), pp. 177–180.

[23] J. BEARDWOOD, H. J. HALTON, AND J. M. HAMMERSLEY, *The Shortest Path Through Many Points*, Proc. Cambridge Philos. Soc., 55 (1959), pp. 299–327.

[24] P. BERMAN, U. FOESSMEIER, M. KARPINSKI, M. KAUFMANN, AND A. Z. ZELIKOVSKY, *Approaching the 5/4 - Approximation for Rectilinear Steiner Trees*, Tech. Rep. WSI-94-06, Wilhelm Schickard-Institut fur Informatik, 1994.

[25] P. BERMAN AND V. RAMAIYER, *Improved Approximations for the Steiner Tree Problem*, in Proc. ACM/SIAM Symp. Discrete Algorithms, San Francisco, CA, January 1992, pp. 325–334.

[26] M. W. BERN, *Two Probabilistic Results on Rectilinear Steiner Trees*, Algorithmica, 3 (1988), pp. 191–204.

[27] M. W. BERN AND M. DE CARVALHO, *A Greedy Heuristic for the Rectilinear Steiner Tree Problem*, Tech. Rep. UCB/CSD 87/306, Computer Science Division (EECS), UCB, 1986.

[28] K. D. BOESE, J. CONG, A. B. KAHNG, K. S. LEUNG, AND D. ZHOU, *On High-Speed VLSI Interconnects: Analysis and Design*, Proc. Asia-Pacific Conf. on Circuits and Systems, (1992), pp. 35–40.

[29] K. D. BOESE AND A. B. KAHNG, *Zero-Skew Clock Routing Trees with Minimum Wirelength*, in Proc. IEEE Intl. ASIC Conf., Rochester, NY, September 1992, pp. 17–21.

[30] K. D. BOESE, A. B. KAHNG, B. A. MCCOY, AND G. ROBINS, *Fidelity and Near-Optimality of Elmore-Based Routing Constructions*, in Proc. IEEE Intl. Conf. Computer Design, Cambridge, MA, October 1993, pp. 81–84.

[31] K. D. BOESE, A. B. KAHNG, B. A. MCCOY, AND G. ROBINS, *Towards Optimal Routing Trees*, in Proc. ACM/SIGDA Physical Design Workshop, Lake Arrowhead, CA, April 1993, pp. 44–51.

[32] K. D. Boese, A. B. Kahng, B. A. McCoy, and G. Robins, *Near-Optimal Critical Sink Routing Tree Constructions*, IEEE Trans. Computer-Aided Design (to appear), (1994).

[33] K. D. Boese, A. B. Kahng, B. A. McCoy, and G. Robins, *Rectilinear Steiner Trees with Minimum Elmore Delay*, in Proc. ACM/IEEE Design Automation Conf., San Diego, CA, June 1994, pp. 381–386.

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

[35] S. Boon, S. Butler, R. Byrne, B. Setering, M. Casalanda, and A. Scherf, *High Performance Clock Distribution For CMOS ASICS*, in Proc. IEEE Custom Integrated Circuits Conf., 1989, pp. 15.4.1–15.4.4.

[36] M. Borah, R. M. Owens, and M. J. Irwin, *An Edge-Based Heuristic for Rectilinear Steiner Trees*, Tech. Rep. CS-93-003, Department of Computer Science, Pennsylvania State University, 1993.

[37] J. Burkis, *Clock Tree Synthesis for High Performance ASICs*, in Proc. IEEE Intl. ASIC Conf., Rochester, NY, September 1991, pp. 9.8.1–9.8.4.

[38] H. Cai, *On Empty Rooms in Floorplan Graphs: Comments on a Deficiency in Two Papers*, IEEE Trans. Computer-Aided Design, 8 (1989), pp. 795–797.

[39] J. Canny, *The Complexity of Robot Motion Planning*, MIT Press, 1988.

[40] P. K. Chan, University of California, Santa Cruz, private communication, June 1993.

[41] P. K. Chan and K. Karplus, *Computing Signal Delay in General RC Networks by Tree/Link Partitioning*, IEEE Trans. Computer-Aided Design, 9 (1990), pp. 898–902.

[42] B. Chandra, G. Das, G. Narasimhan, and J. Soares, *New Sparseness Results on Graph Spanners*, in Proc. ACM Symp. Computational Geometry, June 1992, pp. 192–201.

[43] T. H. Chao and Y. C. Hsu, *Rectilinear Steiner Tree Construction by Local and Global Refinement*, IEEE Trans. Computer-Aided Design, 13 (1994), pp. 303–309.

[44] T. H. Chao, Y. C. Hsu, and J. M. Ho, *Zero Skew Clock Net Routing*, in Proc. ACM/IEEE Design Automation Conf., Anaheim, CA, June 1992, pp. 518–523.

[45] T. H. CHAO, Y. C. HSU, J. M. HO, K. D. BOESE, AND A. B. KAHNG, *Zero Skew Clock Routing With Minimum Wirelength*, IEEE Trans. Circuits and Systems, 39 (1992), pp. 799–814.

[46] B. CHAZELLE, *Tight Bounds on the Stabbing Number of Spanning Trees in Euclidean Space*, Tech. Rep. CS-TR-155-88, Department of Computer Science, Princeton University, 1988.

[47] B. CHAZELLE AND H. EDELSBRUNNER, *An Optimal Algorithm for Intersecting Line Segments*, J. ACM, 39 (1992), pp. 177–180.

[48] D. CHEN AND C. SECHEN, *Mickey: A Macro Cell Global Router*, in Proc. European Design and Test Conf., Amsterdam, The Netherlands, February 1991, pp. 248–252.

[49] D. S. CHEN AND M. SARRAFZADEH, *A Wire-Length Minimization Algorithm for Single-Layer Layouts*, in Proc. IEEE Intl. Conf. Computer-Aided Design, 1992, pp. 390–393.

[50] C. K. CHENG AND N. C. CHOU, *On General Zero-Skew Clock Net Construction*, IEEE Trans. VLSI Systems (to appear), (1995).

[51] C. CHIANG, M. SARRAFZADEH, AND C. K. WONG, *Global Routing Based on Steiner Min-Max Trees*, IEEE Trans. Computer-Aided Design, 9 (1990), pp. 1318–25.

[52] C. CHIANG, M. SARRAFZADEH, AND C. K. WONG, *An Algorithm for Exact Rectilinear Steiner Trees for Switchbox With Obstacles*, in Proc. IEEE Intl. Symp. Circuits and Systems, 1992, pp. 9–12.

[53] C. CHIANG, M. SARRAFZADEH, AND C. K. WONG, *A Weighted-Steiner-Tree-Based Global Router.* unpublished manuscript, 1992.

[54] N. C. CHOU AND C. K. CHENG, *Wire Length and Delay Minimization in General Clock Net Routing*, in Proc. IEEE Intl. Conf. Computer-Aided Design, 1993, pp. 552–555.

[55] F. R. K. CHUNG AND R. L. GRAHAM, *On Steiner Trees for Bounded Point Sets*, Geometriae Dedicata, 11 (1981), pp. 353–361.

[56] J. CHUNG AND C. K. CHENG, *Skew Sensitivity Minimization of Buffered Clock Tree*, in Proc. IEEE Intl. Conf. Computer-Aided Design (to appear), November 1994.

[57] J. P. COHOON AND J. RANDALL, *Critical Net Routing*, in Proc. IEEE Intl. Conf. Computer Design, Cambridge, MA, October 1991, pp. 174–177.

[58] J. P. COHOON AND D. S. RICHARDS, *Optimal Two-Terminal $\alpha-\beta$ Wire Routing*, Integration: the VLSI Journal, 6 (1988), pp. 35–57.

[59] J. P. COHOON, D. S. RICHARDS, AND J. S. SALOWE, *An Optimal Steiner Tree Algorithm for a Net Whose Terminals Lie on the Perimeter of a Rectangle*, IEEE Trans. Computer-Aided Design, 9 (1990), pp. 398–407.

[60] J. CONG, A. B. KAHNG, AND G. ROBINS, *Matching-Based Methods for High-Performance Clock Routing*, IEEE Trans. Computer-Aided Design, 12 (1993), pp. 1157–1169.

[61] J. CONG, A. B. KAHNG, G. ROBINS, M. SARRAFZADEH, AND C. K. WONG, *Performance-Driven Global Routing for Cell Based IC's*, in Proc. IEEE Intl. Conf. Computer Design, Cambridge, MA, October 1991, pp. 170–173.

[62] J. CONG, A. B. KAHNG, G. ROBINS, M. SARRAFZADEH, AND C. K. WONG, *Provably Good Algorithms for Performance-Driven Global Routing*, in Proc. IEEE Intl. Symp. Circuits and Systems, San Diego, CA, May 1992, pp. 2240–2243.

[63] J. CONG, A. B. KAHNG, G. ROBINS, M. SARRAFZADEH, AND C. K. WONG, *Provably Good Performance-Driven Global Routing*, IEEE Trans. Computer-Aided Design, 11 (1992), pp. 739–752.

[64] J. CONG AND K. S. LEUNG, *Optimal Wiresizing Under the Distributed Elmore Delay Model*, in Proc. IEEE Intl. Conf. Computer-Aided Design, 1993, pp. 634 – 639.

[65] 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, June 1993, pp. 606–611.

[66] J. H. CONNELL, *Minimalist Mobile Robotics*, Academic Press, 1990.

[67] T. H. CORMEN, C. E. LEISERSON, AND R. RIVEST, *Introduction to Algorithms*, MIT Press, 1990.

[68] COURANT AND ROBBINS, *What is Mathematics? An Elementary Approach to Ideas and Methods*, Oxford University Press, London, England, 1941.

[69] J. T. CROFT, K. J. FALCONER, AND R. K. GUY, *Unsolved Problems in Geometry*, Springer-Verlag, New York, 1991.

[70] W. M. DAI, T. ASANO, AND E. S. KUH, *Routing Region Definition and Ordering Scheme for Building-Block Layout*, IEEE Trans. Computer-Aided Design, 4 (1985), pp. 189–197.

[71] R. R. L. DE MATOS, *A Rectilinear Arborescence Problem*, PhD thesis, University of Alabama, 1979.

[72] C. C. DESOUZA AND C. C. RIBIERO, *A Tight Worst Case Bound for the Performance Ratio of Heuristics for the Minimum Rectilinear Steiner Tree Problem*, OR Spektrum, 12 (1990), pp. 109–111.

[73] S. DHAR, M. A. FRANKLIN, AND D. F. WANN, *Reduction of Clock Delays in VLSI Structures*, in Proc. IEEE Intl. Conf. Computer Design, Port Chester, NY, October 1984, pp. 778–783.

[74] E. W. DIJKSTRA, *A Note on Two Problems in Connection With Graphs*, Numerische Mathematik, 1 (1959), pp. 269–271.

[75] R. P. DILWORTH, *A Decomposition Theorem for Partially Ordered Sets*, Ann. Math, 51 (1950), pp. 161–166.

[76] D. DOBBERPUHL, R. WITEK, R. ALLMON, R. ANGLIN, ET AL., *A 200-MHz 64-b Dual-Issue CMOS Microprocessor*, IEEE J. Solid State Circuits, 27 (1992), pp. 1555–1567.

[77] W. E. DONATH, R. J. NORMAN, B. K. AGRAWAL, S. E. BELLO, S. Y. HAN, J. M. KURTZBERG, P. LOWY, AND R. I. MCMILLAN, *Timing Driven Placement Using Complete Path Delays*, in Proc. ACM/IEEE Design Automation Conf., 1990, pp. 84–89.

[78] D. Z. DU AND F. K. HWANG, *A Proof of Gilbert-Pollak's Conjecture on the Steiner Ratio*, in Proc. IEEE Symp. Foundations of Computer Science, 1990.

[79] A. E. DUNLOP, V. D. AGRAWAL, D. DEUTSCH, M. F. JUKL, P. KOZAK, AND M. WIESEL, *Chip Layout Optimization Using Critical Path Weighting*, in Proc. ACM/IEEE Design Automation Conf., 1984, pp. 133–136.

[80] R. DUTTA AND M. MAREK-SADOWSKA, *Algorithm for Wire Sizing of Power and Ground Networks in VLSI Designs*, Journal of Circuits, Systems and Computers, 2 (1992), pp. 141–57.

[81] L. N. DWORSKY, *Modern Transmission Line Theory and Applications*, Wiley, 1979.

[82] M. EDAHIRO, *Minimum Skew and Minimum Path Length Routing in VLSI Layout Design*, NEC Research and Development, 32 (1991), pp. 569–575.

[83] M. EDAHIRO, *Equi-Spreading Tree in Manhattan Distance*. unpublished manuscript, 1992.

[84] M. EDAHIRO, *Clustering-Based Optimization Algorithm in Zero-Skew Routings*, in Proc. ACM/IEEE Design Automation Conf., 1993, pp. 612–616.

[85] H. EDELSBRUNNER, *Algorithms in Combinatorial Geometry*, Springer-Verlag, Berlin, 1987.

[86] H. EDELSBRUNNER, L. J. GUIBAS, J. HERSHBERGER, R. SEIDEL, M. SHARIR, J. SNOEYINK, AND E. WELZL, *Implicitly Representing Arrangements of Lines or Segments*, in Proc. ACM Symp. Computational Geometry, Urbana-Champaign, IL, June 1988, pp. 56–69.

[87] W. C. ELMORE, *The Transient Response of Damped Linear Networks with Particular Regard to Wide-Band Amplifiers*, J. Appl. Phys., 19 (1948), pp. 55–63.

[88] D. EPPSTEIN, G. ITALIANO, R. TAMASSIA, R. E. TARJAN, J. WESTBROOK, AND M. YUNG, *Maintenance of a Minimum Spanning Forest in a Dynamic Planar Graph*, in Proc. ACM/SIAM Symp. Discrete Algorithms, San Francisco, CA, January 1990, pp. 1–11.

[89] K. H. ERHARD AND F. M. JOHANNES, *Power/Ground Networks in VLSI: are General Graphs Better than Trees?*, Integration, The VLSI Journal, 14 (1992), pp. 91–109.

[90] K. H. ERHARD, F. M. JOHANNES, AND R. DACHAUER, *Topology Optimization Techniques for Power/Ground Networks in VLSI*, in Proc. European Design Automation Conf., Hamburg, Germany, September 1992, pp. 362–367.

[91] H. ESBENSEN AND P. MAZUMDER, *A Genetic Algorithm for the Steiner Problem in a Graph*, in Proc. European Design and Test Conf., Paris, France, February 1994, pp. 402–406.

[92] S. EVEN, *Graph Algorithms*, Computer Science Press, Inc., Potomac, MD, 1979.

[93] J. FISHBURN, *Clock Skew Optimization*, IEEE Trans. Computers, 39 (1990), pp. 945–951.

[94] A. L. FISHER AND H. T. KUNG, *Synchronizing Large Systolic Arrays*, in Proc. of SPIE, May 1982, pp. 44–52.

[95] U. FOESSMEIER, M. KAUFMANN, AND A. Z. ZELIKOVSKY, *Fast Approximation Algorithms for the Rectilinear Steiner Tree Problem*, Tech. Rep. WSI-93-14, Wilhelm Schickard-Institut fur Informatik, 1993.

[96] L. R. FORD AND D. R. FULKERSON, *Flows in Networks*, Princeton University Press, Princeton, NJ, 1961.

[97] L. R. FOULDS, *Maximum Savings in the Steiner Problem in Phylogeny*, J. Theoretical Biology, 107 (1984), pp. 471–474.

[98] G. N. FREDRICKSON, *Data Structures for On-Line Updating of Minimum Spanning Trees*, SIAM J. Comput., 14 (1985), pp. 781–798.

[99] E. G. FRIEDMAN, *A Partitionable Clock Distribution System for Sequential VLSI Circuits*, in Proc. IEEE Intl. Symp. Circuits and Systems, May 1986, pp. 743–746.

[100] E. G. FRIEDMAN, *Clock Distribution Design in VLSI Circuits - an Overview*, in Proc. IEEE Intl. Symp. Circuits and Systems, May 1993, pp. 1475–1478.

[101] E. G. FRIEDMAN AND J. H. MULLIGAN, *Clock Frequency and Latency in Synchronous Digital Systems*, IEEE Trans. Signal Processing, 39 (1991), pp. 930–934.

[102] E. G. FRIEDMAN AND J. H. MULLIGAN, *Pipelining of High Performance Synchronous Digital Systems*, Intl. Journal of Electronics, 70 (1991), pp. 917–935.

[103] S. C. GADRE, R. VAIDYANATHAN, AND S. Q. ZHENG, *A Potential-Driven Approach to Constructing Rectilinear Steiner Trees*, in Proc. Great Lakes Symp. VLSI, Kalamazoo, MI, March 1993, pp. 95–99.

[104] J. L. GANLEY AND J. P. COHOON, *Routing a Multi-Terminal Critical Net: Steiner Tree Construction in the Presence of Obstacles*, in Proc. IEEE Intl. Symp. Circuits and Systems, London, England, May 1994.

[105] J. L. GANLEY, M. J. GOLIN, AND J. S. SALOWE, *Minimum Spanning Trees for Multiply-Weighted Graphs.* unpublished manuscript, 1994.

[106] M. GAREY AND D. S. JOHNSON, *The Rectilinear Steiner Problem is NP-Complete*, SIAM J. Applied Math., 32 (1977), pp. 826–834.

[107] G. Georgakopoulos and C. H. Papadimitriou, *The 1-Steiner Tree Problem*, J. Algorithms, 8 (1987), pp. 122–130.

[108] L. Gerzberg, *Monolithic Power-Spectrum Centroid Detector*, PhD thesis, Stanford University, May 1979.

[109] E. N. Gilbert and H. O. Pollak, *Steiner Minimal Trees*, SIAM J. Applied Math., 16 (1968), pp. 1–29.

[110] A. V. Goldberg, E. Tardos, and R. E. Tarjan, *Network Flow Algorithms.* unpublished manuscript, March 1989.

[111] D. Goldfarb and M. D. Grigoriadis, *A Computational Comparison of the Dinic and Network Simplex Methods for Maximum Flow*, Annals of Operation Research, 13 (1988), pp. 83–123.

[112] R. E. Gomory, T. C. Hu, and J. M. Yohe, *R-Separating Sets*, Can. J. Math., XXVI (1974), pp. 1418–1429.

[113] J. Griffith, G. Robins, J. S. Salowe, and T. Zhang, *Narrowing the Gap: Near-Optimal Steiner Trees in Polynomial Time*, IEEE Trans. Computer-Aided Design (to appear), (1994).

[114] L. J. Guibas and J. Stolfi, *On Computing all North-East Nearest Neighbors in the L1 Metric*, Information Processing Letters, 17 (1983), pp. 219–223.

[115] D. Gusfield and D. Naor, *Efficient Algorithms for Generalized Cut Trees*, in Proc. ACM/SIAM Symp. Discrete Algorithms, 1990, pp. 422–433.

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

[117] A. C. Harter, *Three-Dimensional Integrated Circuit Layout*, Cambridge University Press, New York, 1991.

[118] N. Hasan, G. Vijayan, and C. K. Wong, *A Neighborhood Improvement Algorithm for Rectilinear Steiner Trees*, in Proc. IEEE Intl. Symp. Circuits and Systems, New Orleans, LA, 1990.

[119] P. S. Hauge, R. Nair, and E. J. Yoffa, *Circuit Placement for Predictable Performance*, in Proc. IEEE Intl. Conf. Computer-Aided Design, Santa Clara, CA, November 1987, pp. 88–91.

[120] D. W. HIGHTOWER, *A Solution to the Line-Routing Problem on the Continuous Plane*, in Proc. Design Automation Workshop, 1969, pp. 1–24.

[121] J. HO, D. T. LEE, C. H. CHANG, AND C. K. WONG, *Bounded-Diameter Minimum Spanning Trees and Related Problems*, in Proc. ACM Symp. Computational Geometry, 1989, pp. 276–282.

[122] J. M. HO, M. T. KO, T. H. MA, AND T. Y. SUNG, *Algorithms for Rectilinear Optimal Multicast Tree Problem*, in Proc. Intl. Symposium on Algorithms and Computation, June 1992, pp. 106–15.

[123] J. M. HO, D. T. LEE, C. H. CHANG, AND C. K. WONG, *Minimum Diameter Spanning Trees and Related Problems*, SIAM J. Comput., 20 (1991), pp. 987–997.

[124] J. M. HO, G. VIJAYAN, AND C. K. WONG, *New Algorithms for the Rectilinear Steiner Tree Problem*, IEEE Trans. Computer-Aided Design, 9 (1990), pp. 185–193.

[125] T. D. HODES, B. A. MCCOY, AND G. ROBINS, *Dynamically-Wiresized Elmore-Based Routing Constructions*, in Proc. IEEE Intl. Symp. Circuits and Systems, London, England, May 1994, pp. 463–466 (Vol. I).

[126] X. HONG, T. XUE, E. S. KUH, C. K. CHENG, AND J. HUANG, *Performance-Driven Steiner Tree Algorithms for Global Routing*, in Proc. ACM/IEEE Design Automation Conf., June 1993, pp. 177–181.

[127] M. A. HOROWITZ, *Timing Models for MOS Circuits*, PhD thesis, Stanford University, January 1984.

[128] T. C. HU, *Integer Programming and Network Flows*, Addison-Wesley, Reading, MA, 1969.

[129] T. C. HU, A. B. KAHNG, AND G. ROBINS, *Optimal Solution of the Discrete Plateau Problem*, Tech. Rep. CSD-920006, Computer Science Department, UCLA, January 1992.

[130] T. C. HU, A. B. KAHNG, AND G. ROBINS, *Solution of the Discrete Plateau Problem*, Proc. of the National Academy of Sciences, 89 (1992), pp. 9235–9236.

[131] T. C. HU, A. B. KAHNG, AND G. ROBINS, *Optimal Robust Path Planning in General Environments*, IEEE Trans. Robotics and Automation, 9 (1993), pp. 775–784.

[132] T. C. Hu and T. Shing, *The α-β Routing*, in VLSI Circuit Layout: Theory and Design, New York, 1985, IEEE Press, pp. 139–143.

[133] J. H. Huang, A. B. Kahng, and C. W. Tsao, *On the Bounded-Skew Clock and Steiner Routing Problems*, Tech. Rep. 940026, Computer Science Department, UCLA, 1994.

[134] J. Hunt and S. Szymanski, *A Fast Algorithm for Computing Longest Common Subsequence*, Comm. of ACM, 20 (1977), pp. 350–353.

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

[136] F. K. Hwang, *An O(n log n) Algorithm for Rectilinear Minimal Spanning Trees*, J. ACM, 26 (1979), pp. 177–182.

[137] F. K. Hwang, *An O(n log n) Algorithm for Suboptimal Rectilinear Steiner Trees*, IEEE Trans. Circuits and Systems, 26 (1979), pp. 75–77.

[138] F. K. Hwang, D. S. Richards, and P. Winter, *The Steiner Tree Problem*, North-Holland, 1992.

[139] A. O. Ivanov and A. A. Tuzhilin, *Minimal Networks: The Steiner Problem and Its Generalizations*, CRC Press, Boca Raton, Florida, 1994.

[140] M. A. B. Jackson and E. S. Kuh, *Performance-Driven Placement of Cell-Based IC's*, in Proc. ACM/IEEE Design Automation Conf., 1989, pp. 370–375.

[141] M. A. B. Jackson and E. S. Kuh, *Estimating and Optimizing RC Interconnect Delay During Physical Design*, in Proc. IEEE Intl. Symp. Circuits and Systems, New Orleans, LA, 1990, pp. 869–871.

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

[143] M. A. B. Jackson, A. Srinivasan, and E. S. Kuh, *Clock Routing for High-Performance IC's*, in Proc. ACM/IEEE Design Automation Conf., 1990, pp. 573–579.

[144] A. B. Kahng, J. Cong, and G. Robins, *High-Performance Clock Routing Based on Recursive Geometric Matching*, in Proc. ACM/IEEE Design Automation Conf., June 1991, pp. 322–327.

[145] A. B. KAHNG AND S. MUDDU, *Delay Estimation of Trees using Two-pole Method and Optimal Equivalent Circuits*, Tech. Rep. CSD-TR-930035, Computer Science Department, UCLA, October 1993.

[146] A. B. KAHNG AND S. MUDDU, *A General Methodology for Response and Delay Computations in VLSI Interconnects*, Tech. Rep. 940015, Computer Science Department, UCLA, 1994.

[147] A. B. KAHNG AND S. MUDDU, *Optimal Equivalent Circuits for Interconnect Delay Calculations Using Moments*, in Proc. European Design Automation Conf., Grenoble, September 1994, pp. 164–169.

[148] A. B. KAHNG AND S. MUDDU, *Two-pole Analysis of VLSI Interconnect Trees.* unpublished manuscript, March 1994.

[149] A. B. KAHNG AND S. MUDDU, *Two-pole Analysis of Interconnection Trees*, in Proc. IEEE Multi-Chip Module Conf. (to appear), Santa Cruz, CA, February 1995.

[150] A. B. KAHNG AND G. ROBINS, *A New Family of Steiner Tree Heuristics With Good Performance: The Iterated 1-Steiner Approach*, in Proc. IEEE Intl. Conf. Computer-Aided Design, Santa Clara, CA, November 1990, pp. 428–431.

[151] 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. 893–902.

[152] A. B. KAHNG AND G. ROBINS, *On Performance Bounds for a Class of Rectilinear Steiner Tree Heuristics in Arbitrary Dimension*, IEEE Trans. Computer-Aided Design, 11 (1992), pp. 1462–1465.

[153] A. B. KAHNG AND C. W. TSAO, *Planar-DME: Improved Planar Zero-Skew Clock Routing With Minimum Pathlength Delay*, Tech. Rep. CSD-TR-940006, Computer Science Department, UCLA, February 1994.

[154] A. B. KAHNG AND C. W. TSAO, *Planar-DME: Improved Planar Zero-Skew Clock Routing With Minimum Pathlength Delay*, in Proc. European Design Automation Conf., Grenoble, September 1994, pp. 440–445.

[155] W. KHAN, X. HE, L. BANGARU, AND N. SHERWANI, *Combat: Zero Skew Minimal Delay Planar Clock Routing for High Performance Systems*, Tech. Rep. 93-08, Western Michigan Univ. Computer Science Department, April 1993.

[156] S. KHULLER, B. RAGHAVACHARI, AND N. YOUNG, *Balancing Minimum Spanning and Shortest Path Trees*, in Proc. ACM/SIAM Symp. Discrete Algorithms, January 1993, pp. 243–250.

[157] S. KIM, R. M. OWENS, AND M. J. IRWIN, *Experiments with a Performance Driven Module Generator*, in Proc. ACM/IEEE Design Automation Conf., June 1992, pp. 687–690.

[158] S. KIMURA, N. KUBO, T. CHIBA, AND I. NISHIOKA, *An Automatic Routing Scheme for General Cell LSI*, IEEE Trans. Computer-Aided Design, 2 (1983), pp. 285–292.

[159] L. KOU, G. MARKOWSKY, AND L. BERMAN, *A Fast Algorithm for Steiner Trees*, Acta Informatica, 15 (1981), pp. 141–145.

[160] M. KRUSKAL, *On the Shortest Spanning Subtree of a Graph, and the Traveling Salesman Problem*, Proc. Amer. Math Soc., 7 (1956), pp. 48–50.

[161] J. C. LATOMBE, *Robot Motion Planning*, Kluwer Academic Publishers, Boston, MA, 1991.

[162] J. W. LAVINUS AND J. P. COHOON, *Routing a Multi-Terminal Critical Net: Steiner Tree Construction in the Presence of Obstacles*, Tech. Rep. CS-93-19, Department of Computer Science, University of Virginia, April 1993.

[163] E. L. LAWLER, *Combinatorial Optimization: Networks and Matroids*, Holt Rinehart and Winston, New York, 1976.

[164] J. H. LEE, N. K. BOSE, AND F. K. HWANG, *Use of Steiner's Problem in Sub-Optimal Routing in Rectilinear Metric*, IEEE Trans. Circuits and Systems, 23 (1976), pp. 470–476.

[165] K. W. LEE AND C. SECHEN, *A New Global Router for Row-Based Layout*, in Proc. IEEE Intl. Conf. Computer-Aided Design, Santa Clara, CA, November 1990, pp. 180–183.

[166] K. W. LEE AND C. SECHEN, *A Global Router for Sea-of-Gates Circuits*, in Proc. European Design and Test Conf., Amsterdam, The Netherlands, February 1991, pp. 242–247.

[167] K.W. Lee and Y.Z. Liao, ArcSys Corporation, personal communication, April 1994.

[168] T. LENGAUER, *Combinatorial Algorithms for Integrated Circuit Layout*, John Wiley & Sons Ltd, West Sussex, England, 1990.

[169] H. P. LENHOF, J. S. SALOWE, AND D. E. WREGE, *New Methods to Mix Shortest-Path and Minimum Spanning Trees.* unpublished manuscript, 1993.

[170] C. LEVCOPOULOS AND A. LINGAS, *There are Planar Graphs Almost as Good as the Complete Graphs and as Short as Minimum Spanning Trees*, in Proc. Intl. Symposium on Optimal Algorithms, June 1989, pp. 9–13.

[171] F. D. LEWIS, W. C. PONG, AND N. VANCLEAVE, *Local Improvement in Steiner Trees*, in Proc. Great Lakes Symp. VLSI, Kalamazoo, MI, March 1993, pp. 105–106.

[172] Y. M. LI AND M. A. JABRI, *A Zero-Skew Clock Routing Scheme for VLSI Circuits*, in Proc. IEEE Intl. Conf. Computer-Aided Design, 1992, pp. 458–463.

[173] I. LIN AND D. H. C. DU, *Performance-Driven Constructive Placement*, in Proc. ACM/IEEE Design Automation Conf., 1990, pp. 103–106.

[174] I. LIN, J. A. LUDWIG, AND K. ENG, *Analyzing Cycle Stealing on Synchronous Circuits with Level-Sensitive Latches*, in Proc. ACM/IEEE Design Automation Conf., June 1992, pp. 393–398.

[175] S. LIN AND C. K. WONG, *Process-Variation-Tolerant Clock Skew Minimization*, in Proc. IEEE Intl. Conf. Computer-Aided Design (to appear), November 1994.

[176] T. M. LIN AND C. A. MEAD, *Signal Delay in General RC-networks*, IEEE Trans. Computer-Aided Design, CAD-3 (1984), pp. 331–349.

[177] A. L. LOEB, *Space Structures: Their Harmony and Counterpoint*, Birkhauser, New York, 1991.

[178] M. MAREK-SADOWSKA AND S. P. LIN, *Timing Driven Placement*, in Proc. IEEE Intl. Conf. Computer-Aided Design, Santa Clara, CA, November 1989, pp. 94–97.

[179] D. MARTIN AND N. C. RUMIN, *Delay Prediction From Resistance-Capacitance Models of General MOS Circuits*, IEEE Trans. Computer-Aided Design, 12 (1993), pp. 997–1003.

[180] S. P. MCCORMICK, *Modeling and Simulation of VLSI Interconnections with Moments*, PhD thesis, MIT, June 1989.

[181] B. A. McCoy and G. Robins, *Non-Tree Routing*, in Proc. European Design and Test Conf., Paris, France, February 1994, pp. 430–434.

[182] C. A. Mead and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley, Reading, MA, 1980.

[183] M. Minoux, *Efficient Greedy Heuristics for Steiner Tree Problems Using Reoptimization and Supermodularity*, INFOR, 28 (1990), pp. 221–233.

[184] S. Mirayala, J. Hashmi, and N. Sherwani, *Switchbox Steiner Tree Problem in Presence of Obstacles*, in Proc. IEEE Intl. Conf. Computer-Aided Design, Santa Clara, CA, November 1991, pp. 536–539.

[185] J. S. B. Mitchell, *An Algorithmic Approach to Some Problems in Terrain Navigation, in Geometric Reasoning, D. Kapur and J. L. Mundy, editors*, MIT Press, 1988.

[186] J. S. B. Mitchell, *On Maximum Flows in Polyhedral Domains*, Journal of Computer and System Sciences, 40 (1990), pp. 88–123.

[187] S. Muddu, University of California, Los Angeles, private communication, August 1993.

[188] P. R. O'Brien and T. L. Savarino, *Modeling the Driving-Point Characteristic of Resistive Interconnect for Accurate Delay Estimation*, in Proc. IEEE Intl. Conf. Computer-Aided Design, November 1989, pp. 512–515.

[189] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization*, Prentice-Hall, 1982.

[190] C. H. Papadimitriou and U. V. Vazirani, *On Two Geometric Problems Relating to the Traveling Salesman Problem*, J. Algorithms, 5 (1984), pp. 231–246.

[191] D. Peleg and A. Schaffer, *Graph Spanners*, J. Graph Theory, 13 (1989), pp. 99–116.

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

[193] B. T. Preas, *Placement and Routing Algorithms for Hierarchical Integrated Circuit Layout*, PhD thesis, Department of Electrical Engineering, Stanford University, 1979.

[194] B. T. PREAS AND M. J. LORENZETTI, *Physical Design Automation of VLSI Systems*, Benjamin/Cummings, Menlo Park, CA, 1988.

[195] F. P. PREPARATA AND M. I. SHAMOS, *Computational Geometry: An Introduction*, Springer-Verlag, New York, 1985.

[196] A. PRIM, *Shortest Connecting Networks and Some Generalizations*, Bell Syst. Tech J., 36 (1957), pp. 1389–1401.

[197] S. PULLELA, N. MENEZES, J. OMAR, AND L. T. PILLAGE, *Skew and Delay Optimization for Reliable Buffered Clock Trees*, in Proc. IEEE Intl. Conf. Computer-Aided Design, Santa Clara, CA, November 1993, pp. 556–559.

[198] S. PULLELA, N. MENEZES, AND L. T. PILLAGE, *Reliable Non-Zero Skew Clock Trees Using Wire Width Optimization*, in Proc. ACM/IEEE Design Automation Conf., San Diego, CA, 1993, pp. 165–170.

[199] Y. V. RAJPUT, *Modelling Distributed RC Lines for the Transient Analysis of Complex Networks*, Intl. Journal of Electronics, 36 (1974), pp. 709–717.

[200] P. RAMANATHAN AND K. G. SHIN, *A Clock Distribution Scheme for Non-Symmetric VLSI Circuits*, in Proc. IEEE Intl. Conf. Computer-Aided Design, Santa Clara, CA, November 1989, pp. 398–401.

[201] S. K. RAO, P. SADAYAPPAN, F. K. HWANG, AND P. W. SHOR, *The Rectilinear Steiner Arborescence Problem*, Algorithmica, (1992), pp. 277–288.

[202] D. S. RICHARDS, *Fast Heuristic Algorithms for Rectilinear Steiner Trees*, Algorithmica, 4 (1989), pp. 191–207.

[203] G. ROBINS, *On Optimal Interconnections*, PhD thesis, Department of Computer Science, UCLA, CSD-TR-920024, 1992.

[204] G. ROBINS AND J. S. SALOWE, *On the Maximum Degree of Minimum Spanning Trees*, in Proc. ACM Symp. Computational Geometry, Stony Brook, NY, June 1994, pp. 250–258.

[205] J. RUBINSTEIN, P. PENFIELD, AND M. A. HOROWITZ, *Signal Delay in RC Tree Networks*, IEEE Trans. Computer-Aided Design, 2 (1983), pp. 202–211.

[206] T. SAKURAI, *Approximation of Wiring Delay in MOSFET LSI*, IEEE J. Solid State Circuits, 18 (1983), pp. 418–426.

[207] J. S. SALOWE, D. S. RICHARDS, AND D. WREGE, *Mixed Spanning Trees: A Technique for Performance-Driven Routing*, in Proc. Great Lakes Symp. VLSI, Kalamazoo, MI, March 1993, pp. 62–66.

[208] J. S. SALOWE AND D. M. WARME, *An Exact Rectilinear Steiner Tree Algorithm*, in Proc. IEEE Intl. Conf. Computer Design, Cambridge, MA, October 1993, pp. 472–475.

[209] S. SAPETNEKAR, *RC Interconnect Optimization Under the Elmore Delay Model*, in Proc. ACM/IEEE Design Automation Conf., San Diego, CA, June 1994, pp. 387–391.

[210] M. SARRAFZADEH AND C. K. WONG, *Hierarchical Steiner Tree Construction in Uniform Orientations*, IEEE Trans. Computer-Aided Design, 11 (1992), pp. 1095–1103.

[211] J. T. SCHWARTZ, M. SHARIR, AND H. HOPCROFT, *Planning, Geometry and Complexity of Robot Motion*, Ablex Publishing Corp., 1987.

[212] C. SECHEN, *VLSI Placement and Global Routing Using Simulated Annealing*, Kluwer Academic Publishers, Boston, MA, 1988.

[213] R. SEDGEWICK, *Algorithms, 2nd ed.*, Addison-Wesley, Reading, MA, 1988.

[214] M. SEKI, K. INOUE, K. KATO, K. TSURUSAKI, S.FUKUSAWA, H. SASAKI, AND M. AIZAWA, *A Specified Delay Accomplishing Clock Router Using Multiple Layers*, in Proc. IEEE Intl. Conf. Computer-Aided Design (to appear), November 1994.

[215] M. SERVIT, *Heuristic Algorithms for Rectilinear Steiner Trees*, Digital Process., 7 (1981), pp. 21–31.

[216] N. SHERWANI, *Algorithms for VLSI Physical Design Automation*, Kluwer Academic Publishers, Boston, MA, 1993.

[217] N. SHERWANI AND B. WU, *Effective Buffer Insertion of Clock Tree for High Speed VLSI Circuits*, Microelectronics Journal, 23 (1992), pp. 291–300.

[218] G. SHUTE, *Worse-Case Length Ratios for Various Heuristics for Rectilinear and Euclidean Steiner Minimal Trees*, Tech. Rep. 90-10, University of Minnesota, Duluth, 1990.

[219] G. M. SHUTE, L. L. DENEEN, AND C. D. THOMBORSON, *An O(n log n) Plane-Sweep Algorithm for L_1 and L_∞ Delaunay Triangulations*, Algorithmica, 6 (1991), pp. 207–221.

[220] J. M. SMITH, D. T. LEE, AND J. S. LIEBMAN, *An O(N log N) Heuristic Algorithm for the Rectilinear Steiner Minimal Tree Problem*, Engineering Optimization, 4 (1980), pp. 179–192.

[221] J. M. SMITH AND J. S. LIEBMAN, *Steiner Trees, Steiner Circuits and the Interference Problem in Building Design*, Engineering Optimization, 4 (1979), pp. 15–36.

[222] T. L. SNYDER, *A Simple and Faster Algorithm for the Rectilinear Steiner Problem in General Dimension*, in Proc. ACM Symp. Computational Geometry, 1990.

[223] T. L. SNYDER, *On the Exact Location of Steiner Points in General Dimension*, SIAM J. Comput., 21 (1992), pp. 163–180.

[224] T. L. SNYDER AND J. M. STEELE, *Worst-Case Greedy Matchings in the Unit d-Cube*, Networks – an International Journal, 20 (1990), pp. 779–800.

[225] J. SOUKUP, *Circuit Layout*, Proc. IEEE, 69 (1981), pp. 1281–1304.

[226] A. Srinivasan, private communication, October 1991.

[227] A. SRINIVASAN, K. CHAUDHARY, AND E. S. KUH, *RITUAL: A Performance Driven Placement Algorithm for Small-Cell ICs*, in Proc. IEEE Intl. Conf. Computer-Aided Design, Santa Clara, CA, November 1991, pp. 48–51.

[228] M. SRIRAM AND S. M. KANG, *Performance Driven MCM Routing Using a Second Order RLC Tree Delay Model*, in IEEE Intl. Conf. on Wafer Scale Integration, San Francisco, CA, USA, January 1993, pp. 262–267.

[229] J. M. STEELE, *Growth Rates of Euclidean Minimal Spanning Trees With Power Weighted Edges*, Annals of Probability, 16 (1988), pp. 1767–1787.

[230] V. S. SUNDERAM, *PVM: A Framework for Parallel Distributed Computing*, Concurrency: Practice and Experience, 2 (1990), pp. 315–339.

[231] K. J. SUPOWIT, *New Techniques for Some Dynamic Closest-Point and Farthest-Point Problems*, in Proc. ACM/SIAM Symp. Discrete Algorithms, 1990, pp. 84–90.

[232] K. J. SUPOWIT AND E. M. REINGOLD, *Divide and Conquer Heuristics for Minimum Weighted Euclidean Matching*, SIAM J. Computing, 12 (1983), pp. 118–143.

[233] K. J. Supowit, E. M. Reingold, and D. A. Plaisted, *The Travelling Salesman Problem and Minimum Matching in the Unit Square*, SIAM J. Computing, 12 (1983), pp. 144–156.

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

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

[236] S. Teig, R. L. Smith, and J. Seaton, *Timing Driven Layout of Cell-Based ICs*, VLSI Systems Design, (1992), pp. 63–73.

[237] G. Tellez and M. Sarrafzadeh, *Clock Period Constrained Minimal Buffer Insertion in Clock Trees*, in Proc. IEEE Intl. Conf. Computer-Aided Design (to appear), November 1994.

[238] G. F. Tóth, *New Results in the Theory of Packing and Covering*, Convexity and its Applications, 1983.

[239] V. A. Trubin, *Subclass of the Steiner Problems on a Plane with Rectilinear Metric*, Cybernetics, 21 (1985), pp. 320–322.

[240] R. S. Tsay, *Exact Zero Skew*, in Proc. IEEE Intl. Conf. Computer-Aided Design, Santa Clara, CA, November 1991, pp. 336–339.

[241] R. S. Tsay, *Exact Zero Skew*, Tech. Rep. RC-16683, IBM T. J. Watson Research Center, Yorktown Heights, March 1991.

[242] R. S. Tsay and I. Lin, *Robin Hood: A System Timing Verifier for Multi-Phase Level-Sensitive Clock Designs*, in Proc. IEEE Intl. ASIC Conf., September 1992, pp. 516–519.

[243] P. Vaidya, *Geometry Helps in Matching*, in Proc. ACM Symp. the Theory of Computing, 1988, pp. 422–425.

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

[245] J. Vlach, J. A. Barby, A. Vannelli, T. Talkhan, and C. J. Shi, *Group Delay as an Estimate of Delay in Logic*, IEEE Trans. Computer-Aided Design, 10 (1991), pp. 949–953.

[246] D. F. Wann and M. A. Franklin, *Asynchronous and Clocked Control Structure for VLSI Based Interconnection Networks*, IEEE Trans. Computers, 21 (1983), pp. 284–293.

[247] E. WELZL, *Partition Trees for Triangle Counting and Other Range Searching Problems*, in Proc. ACM Symp. Computational Geometry, Urbana-Champaign, IL, June 1988, pp. 23–33.

[248] S. WOLFRAM, *Mathematica: A System for Doing Mathematics by Computer, Second Edition*, Addison-Wesley, Reading, MA, 1991.

[249] Y. F. WU, P. WIDMAYER, AND C. K. WONG, *A Faster Approximation Algorithm for the Steiner Problem in Graphs*, Acta Informatica, 23 (1986), pp. 223–229.

[250] Y. Y. YANG AND O. WING, *Suboptimal Algorithm for a Wire Routing Problem*, IEEE Trans. on Circuit Theory, 19 (1972), pp. 508–511.

[251] A. C. C. YAO, *On Constructing Minimum Spanning Trees in k-Dimensional Spaces and Related Problems*, SIAM J. Comput., 11 (1982), pp. 721–736.

[252] A. Z. Zelikovsky, Institute of Mathematics, Moldavian Academy of Sciences, private communication, April 1994.

[253] A. Z. ZELIKOVSKY, *An 11/8-approximation Algorithm for the Steiner Problem on Networks with Rectilinear Distance*, in Janos Bolyai Mathematica Societatis Conf.: Sets, Graphs, and Numbers, January 1991, pp. 733–745.

[254] A. Z. ZELIKOVSKY, *An 11/6 Approximation Algorithm for the Network Steiner Problem*, Algorithmica, 9 (1993), pp. 463–470.

[255] A. Z. ZELIKOVSKY, P. BERMAN, AND M. KARPINSKI, *Improved Approximation Bounds for the Rectilinear Steiner Tree Problem*, Tech. Rep. Report No. 85108-CS, Institut fur Informatik, Universitat Bonn, 1994.

[256] D. ZHOU, S. SU, F. TSUI, D. S. GAO, AND J. S. CONG, *Analysis of Tree of Transmission Lines*, Tech. Rep. CSD-TR-920010, Computer Science Department, UCLA, March 1992.

[257] D. ZHOU, F. TSUI, J. CONG, AND D. GAO, *A Distributive RCL-Model for MCM Layout*, in IEEE Multi-Chip Module Conf., March 1993, pp. 191–197.

[258] D. ZHOU, F. TSUI, AND D. S. GAO, *High Performance Multichip Interconnection Design*, in Proc. ACM/SIGDA Physical Design Workshop, Lake Arrowhead, CA, April 1993, pp. 32–43.

[259] Q. ZHU AND W. W. M. DAI, *Perfect-Balance Planar Clock Routing With Minimal Path-Length*, in Proc. IEEE Intl. Conf. Computer-Aided Design, 1992, pp. 473–476.

[260] Q. ZHU AND W. W. M. DAI, *Hierarchical Clock Routing for Multi-Chip Modules Based on Area Pad Interconnection.* unpublished manuscript, 1994.

[261] Q. ZHU, W. W. M. DAI, AND J. G. XI, *Optimal Sizing of High-Speed Clock Networks Based on Distributed RC and Lossy Transmission Line Models*, in Proc. IEEE Intl. Conf. Computer-Aided Design, 1993, pp. 628–633.