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

ON OPTIMAL INTERCONNECTIONS

G. Robins

June 1992
CSD-920024

UNIVERSITY OF CALIFORNIA

Los Angeles

On Optimal Interconnections

A dissertation submitted in partial satisfaction

of the requirements for the degree

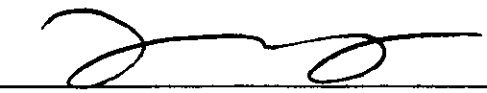
Doctor of Philosophy in Computer Science

by

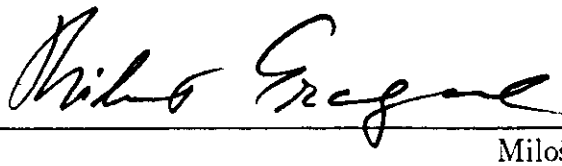
Gabriel Robins

1992

The dissertation of Gabriel Robins is approved.

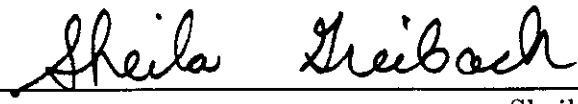


Jason Cong

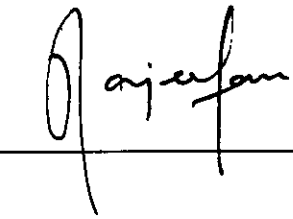


Miloš Ercegovac

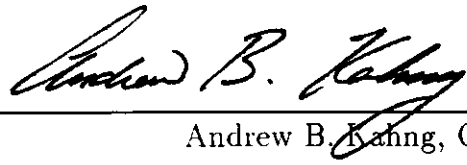
Basil Gordon



Sheila Greibach



Rajeev Jain



Andrew B. Kahng, Committee Chair

University of California, Los Angeles

1992

To my Family . . .

Table of Contents

List of Figures

List of Tables

ACKNOWLEDGMENTS

First and foremost, I would like to express my gratitude to my advisor, Professor Andrew Kahng, from whom I have learned many valuable skills, both research-related and otherwise. It is indeed very rare to encounter a person possessing all of Andrew's positive qualities. His accessibility to his students is only matched by his stamina: he is *always* ready and willing to discuss research, often into the early morning hours. Our professional working relationship has been so very productive and enjoyable that I expect it to continue for many decades to come.

Professor T. C. Hu has my gratitude for his wisdom and insight on countless occasions. His consistent support and his influence on his students extend far beyond the days of graduate training, and he is therefore an invaluable resource to his students (and grand-students) throughout their careers.

I thank Professor Jason Cong for many discussions and his contributions of the bounded-radius tree problem formulation in Chapter 4, and the geometric analysis of H-flipping in Chapter 3. Jason is also an impeccable example of the ideal researcher: he combines the subtle insight to ask the right questions, with the perseverance to relentlessly hunt for the solutions.

Professor Sheila Greibach has consistently supported me with feedback, encouragement, friendliness, and numerous opportunities to test my ideas on a competent audience in her regular theory seminar.

Professor Eli Gafni has provided me with much valuable advice, including a pointer to the "shallow-light" construction of Chapter 4; he has also set an excellent example on many occasions. I thank Professor Miloš Ercegovac for his continued support, and for brightening many days with his keen sense of humor.

I gratefully acknowledge numerous helpful comments from Professor Basil Gordon of the UCLA Mathematics Department, from Professor Rajeev Jain of the UCLA Electrical Engineering Department, and from Professor Sinai Robins of the University of Northern Colorado Mathematics Department. I thank Elizabeth A. Walkup for her insights and collaboration on the probe generation research.

Many thanks go to Verra Morgan of the UCLA Computer Science Department, who helped with the administrative hurdles at each step of the way: her pleasant character and warm helpful personality are assets to us all.

Finally, I appreciate the support of the IBM Corporation through a graduate fellowship, and the support of the National Science Foundation via a grant to my advisor from the MIPS directorate.

"There is timing in everything."

– Miyamoto Musashi, 1584-1645

A Book of Five Rings

VITA

1962	Born, New York, NY, USA.
1983	**B.S.** (Mathematics / Computer Science), UCLA.
1985	**M.S.** (Computer Science), Princeton University, Princeton, NJ.
1992	**Ph.D.** (Computer Science), UCLA, Los Angeles, CA.
1983–1985	**Teaching Assistant**, Princeton University.
1984	**Consultant**, Xerox Electro-Optical Systems.
1985–1989	**Researcher**, USC Information Sciences Institute.
1987–1990	**Teaching Assistant**, Computer Science Department, UCLA.
1989–1992	**Research Assistant**, Computer Science Department, UCLA.

PUBLICATIONS AND PRESENTATIONS

Cong, J., Kahng, A. B., and Robins, G., **Performance-Driven Global Routing for Cell Based IC's**, *UCLA CSD TR # CSD-900052*, December 1990.

——, **On Clock Routing For General Cell Layouts**, *Proc. IEEE Intl. ASIC Conf.*, Rochester, September 1991, pp. P14:5.1-P14:5.4.

——, Sarrafzadeh, M., and Wong, C. K., **Performance-Driven Global Routing for Cell Based IC's**, *Proc. IEEE Intl. Conf. on Computer Design*, Cambridge, October 1991, pp. 170-173.

——, **Provably Good Algorithms for Performance-Driven Global Routing**, *Proc. IEEE Intl. Symp. on Circuits and Systems*, San Diego, May 1992, pp. 2240-2243.

——, **Provably Good Performance-Driven Global Routing**, *IEEE Trans. on CAD*, Vol. 11, No. 6, pp. 739-752.

Foster, L., and Robins, G., **Solution to Problem E2872**, *American Mathematical Monthly*, Vol. 89, No. 7, August-September, 1982, pp. 499-500.

Gafni, E., and Robins G., **A Potential-Based Proof for a Certain Pebbling Game and its Generalization**, *UCLA CSD TR # CSD-880062*, August 1988.

Hu, T. C., Kahng A. B., and Robins, G., **Optimal Robust Path Planning in General Environments**, *UCLA CSD TR # CSD-910082*, December 1991.

——, **Optimal Solution of the Discrete Plateau Problem**, *UCLA CSD TR # CSD-920006*, March 1992.

Kaczmarek, T., Bates, R., and Robins, G., **Recent Developments in NIKL**, American Association of Artificial Intelligence, *Proc. Fifth National Conf. on Artificial Intelligence*, Philadelphia, Pennsylvania, August 1986.

Kahng, A. B., Cong, J., and Robins, G., **High-Performance Clock Routing Based on Recursive Geometric Matching**, *UCLA CSD TR # CSD-900046*, December 1990.

——, **High-Performance Clock Routing Based on Recursive Geometric Matching**, *Proc. ACM/IEEE Design Automation Conf.*, San Francisco, June 1991, pp. 322-327.

Kahng, A. B., and Robins, G., **A New Family of Steiner Tree Heuristics with Good Performance**, *UCLA CSD TR # CSD-900014*, April 1990.

——, **On Performance Bounds for a Class of Rectilinear Steiner Tree Heuristics in Arbitrary Dimension**, *UCLA CSD TR # CSD-900015*, April 1990.

——, **On Structure and Randomness in Practical Optimization**, *UCLA Computer Science Department Annual*, 1990, pp. 25-40.

——, **A New Family of Steiner Tree Heuristics with Good Performance: The Iterated 1-Steiner Approach**, Distinguished Paper, *Proc. Intl. Conf. on Computer-Aided Design*, Santa Clara, November 1990, pp. 428-431.

——, **An Optimal Algorithm for Computing All Regular Linear Degeneracies of Pointsets in E^d**, *UCLA CSD TR # CSD-900045*, December 1990.

——, **Optimal Algorithms for Determining Regularity in Pointsets**, *Proc. Canadian Conf. on Computational Geometry*, Vancouver, August 1991, pp. 167-170.

——, **Optimal Algorithms for Extracting Spatial Regularity in Images**, *Pattern Recognition Letters*, Vol. 12, No. 12, December 1991, pp. 757-764.

——, **On Performance Bounds for a Class of Rectilinear Steiner Tree Heuristics in Arbitrary Dimension**, to appear in *IEEE Trans. on CAD.*

——, **A New Class of Iterative Steiner Tree Heuristics With Good Performance**, to appear in *IEEE Trans. on CAD.*

Kahng A. B., Robins, G., and Walkup, E., **On Connectivity Verification in Multi-Chip Module Substrates**, *UCLA CSD TR # CSD-910074*, October 1991.

——, **New Results and Algorithms for MCM Substrate Testing**, *Proc. IEEE Intl. Symp. on Circuits and Systems*, San Diego, May 1992, pp. 1113-1116.

Robins, G., **The NIKL Manual**, USC Information Sciences Institute, Marina Del Rey, April 1985.

——, Invited Talk, **ISI Grapher: a Portable Tool for Displaying Graphs Pictorially**, *Symboliikka '87*, Helsinki, Finland, August 1987.

——, **The ISI Grapher: a Portable Tool for Displaying Graphs Pictorially**, *Proc. Symboliikka '87*, Helsinki, Finland, August 1987.

——, **On Style, Expressibility, and Efficiency in Functional Programming Languages**, *UCLA Computer Science Department Quarterly*, Fall 1987, pp. 105-121; also UCLA CSD TR # CSD-880029, 1988.

——, **The ISI Grapher Manual**, *ISI Technical Report ISI/TM-88-197*, USC Information Sciences Institute, Marina Del Rey, California, February 1988.

——, **Applications of The ISI Grapher**, *Proc. Fourth Annual Artificial Intelligence and Advanced Computer Conf.*, Long Beach, California, May 1988.

——, **The ISI Grapher: a Portable Tool for Displaying Graphs Pictorially**, *Multicomputer Vision*, Levialdi, S., ed., Chapter 12, Academic Press, London, 1988, pp. 185-202.

——, **Signal Constellation Design Tool: A Case Study in User Interface Synthesis**, *UCLA CSD TR # CSD-880051*, 1988.

——, **Teaching Theoretical Computer Science at the Undergraduate Level: Experiences, Observations, and Proposals to Improve the Status Quo**, *UCLA CSD TR # CSD-880063*, August 1988.

——, **An Interactive Gate-Level Simulator of a Classical Von Neumann Architecture, as an Educational Aid for Introducing Novices to the Fundamentals of Computer Organization**, *UCLA CSD TR # CSD-880064*, August 1988.

——, **Signal Constellation Design Tool: A Case Study in User Interface Synthesis**, *Proc. Second Intl. Conf. on Computer-Assisted Learning*, Dallas, Texas, May 1989, pp. 452-467.

——, **An Interactive Gate-Level Simulator of a Classical Von Neumann Architecture, as an Educational Aid for Introducing Novices to the Fundamentals of Computer Organization**, *Poster Session, Third Intl. Conf. on Human-Computer Interaction*, Boston, Massachusetts, September 1989.

ABSTRACT OF THE DISSERTATION

On Optimal Interconnections

by

Gabriel Robins

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 1992

Professor Andrew B. Kahng, Chair

Many applications require algorithms for determining optimal interconnections. This dissertation centers on new geometric formulations and approximation algorithms for optimizing interconnection objectives which are of particular interest in the design of high-performance VLSI systems. These formulations include Steiner trees, pathlength-balanced trees, bounded-radius trees, and prescribed-width paths; we also address the closely related question of efficiently testing physical interconnections. For most cases, we have new, best-known results, and in all cases we have empirically demonstrated significant improvements over the best previous methods.

We give the best-performing rectilinear Steiner tree heuristic to date: the algorithm has worst-case performance ratio strictly less than 3/2 times optimal, settling a long-standing open problem. We also give a class of instances which are pathological for virtually all existing Steiner tree heuristics in the literature, thus disproving several conjectures and claimed performance bounds.

We propose a matching-based method for pathlength-balanced trees: the con-

struction yields near-zero average pathlength skew while maintaining small total tree cost. To address a separate objective, we also offer the first general formulation of performance-driven routing, allowing a smooth tradeoff of tree cost for tree radius. Our algorithm melds the two classical constructions of the minimum spanning tree and the shortest paths tree, and has worst-case performance bounded by a constant times optimal with respect to both tree cost and tree radius.

Motivated by recent circuit testing applications, we formulate connectivity testing as a problem of tree verification via k-probes. We present linear-time algorithms which compute a minimum probe set achieving complete coverage of both edge and node fault classes. Actual testing demands the efficient scheduling of probe operations: we show that this entails a special type of metric traveling salesman optimization, and we give provably good heuristics.

Finally, we address a fundamental problem in routing and path planning: determining a minimum-cost path of prescribed width which connects a given source-destination pair in an arbitrarily costed region. We give the first known polynomial-time algorithm for this problem, and extend our approach to solve a discrete version of the classical Plateau problem on minimal surfaces.

CHAPTER 1

Interconnection Problems

Consider the task of connecting together a set of sites. This common problem arises in many domains, such as when a county government must connect several towns together via a network of roads. Since there is not necessarily a unique way to achieve connectivity, we are naturally concerned about the "expense" of a solution. For the county government, this expense might correspond to the cost of paving the roads and thus be proportional to the sum of lengths of all roads constructed. A small instance of this problem is shown in Figure 1.1, where three towns are located on the vertices of an equilateral triangle. We may achieve connectivity at a cost of 2 units by using any two edges of the triangle; however, substantial savings are realized if we add a new site at the center of the triangle and connect each town directly to this new junction, resulting in a reduced cost of $\sqrt{3} \approx 1.732$ units. The general problem of determining which new junctions, or Steiner points, will allow minimization of total interconnection cost is known as the *minimum Steiner tree problem.*

Continuing with our scenario, observe that while the county treasurer might wish to minimize the total road length, another official might promote a completely different objective. For example, public policy considerations would dictate that no driving distance between two locations be excessively long (e.g., so that no person lives more than ten minutes from a hospital), and we therefore

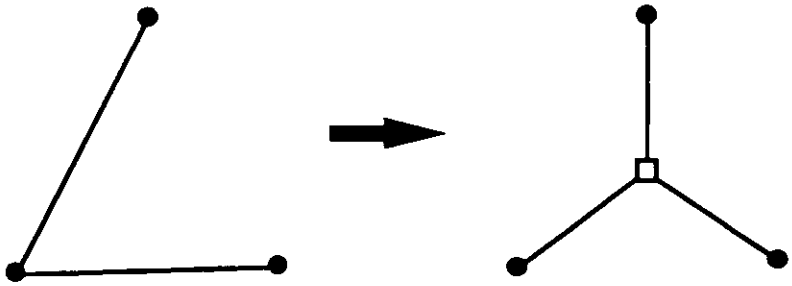


Figure 1.1: An example of three sites with two different routings: the minimum spanning tree (left), and the optimal Steiner tree (right).

wish to minimize the total length of the roads while at the same time maintaining reasonable driving distance between any two sites. We can capture these competing objectives with a *bounded-radius minimum spanning tree* formulation: Figure 1.2 contrasts a minimum spanning tree and a bounded-radius tree for the same point set.

Another connectivity criterion is motivated by issues of synchronization, particularly with respect to the distribution of time-critical information (e.g., news regarding the results of a horse race) to a set of sites. In this case, all sites, which correspond to leaves in a tree topology, must receive a given piece of information simultaneously. This gives rise to a *pathlength-balanced tree* problem, where we want a minimum cost tree over a given set of leaves such that all root-leaf paths have equal length. An example of such a tree is shown in Figure 1.3.

Given an interconnection tree, e.g., for a telephone communication network, one often wishes to test the tree topology. In the telephone network, we may verify parts of the topology by attempting calls between selected pairs of sites, with the assumption that a successful call between two nodes of the tree network checks all edges of the unique path between these nodes (see Figure 1.4). We may

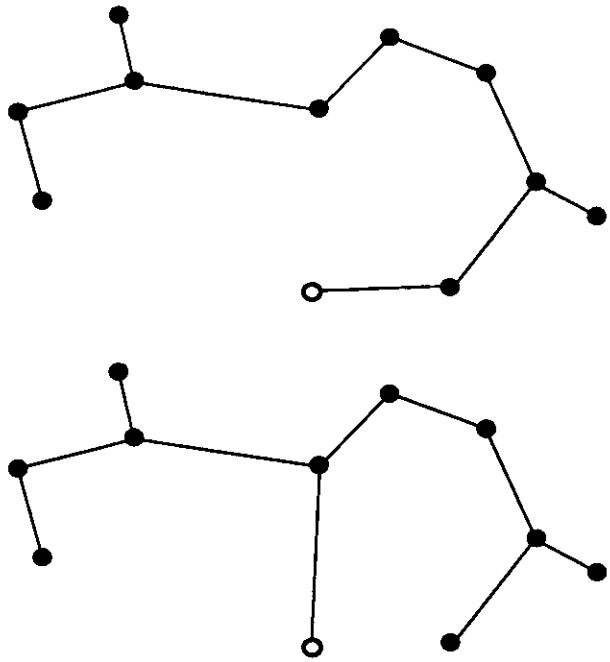


Figure 1.2: An example of contrasting a minimum spanning tree (top) and a bounded-radius tree (bottom).

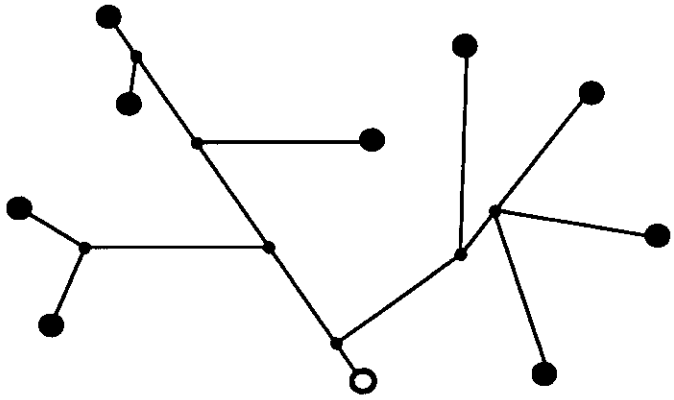


Figure 1.3: A pathlength-balanced tree: all pathlengths from the root (hollow dot) to the leaves are equal.

immediately ask for the minimum number of phone calls needed to completely test a given interconnection tree in this manner. Next, if only a single pair of agents must make these calls, we would naturally seek to minimize the total travel

time (or expense) needed to accomplish the testing.

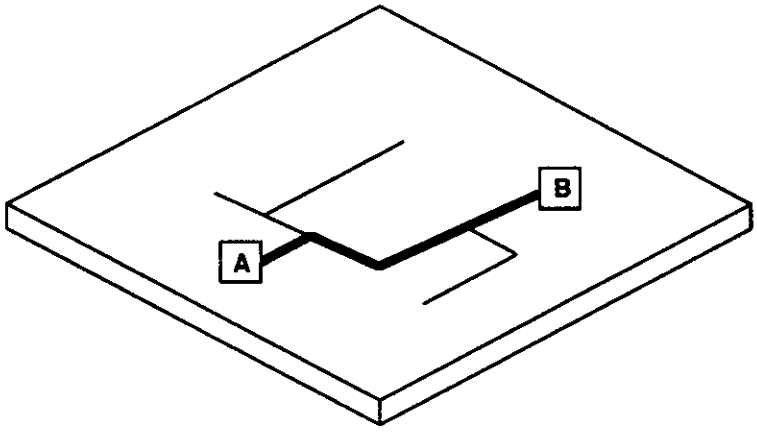


Figure 1.4: Testing a tree network: a phone call between A and B tests the entire A-B path.

Finally, yet another interesting problem of interconnection arises when we seek an optimal source-destination path which has prescribed minimum width. For example, if a four-lane road is to be paved through a national forest, we may wish to minimize the total number of trees felled (Figure 1.5); we may even wish to cut down small trees in favor of preserving large ones.

In this dissertation we study these "optimal interconnection" problems in detail, and provide efficient algorithms for the solution of each of these problems. Our chosen area of application and primary domain of discussion is computer-aided design (CAD) of very large scale integrated (VLSI) circuits; however, our algorithms may easily be applied to a number of other domains ranging from urban planning to the design of communication networks.

The rest of this Chapter is organized as follows. In Section 1.1, we give a brief overview of our canonical application domain, VLSI circuit routing. Section 1.2 develops the various problem formulations and main results that are treated in

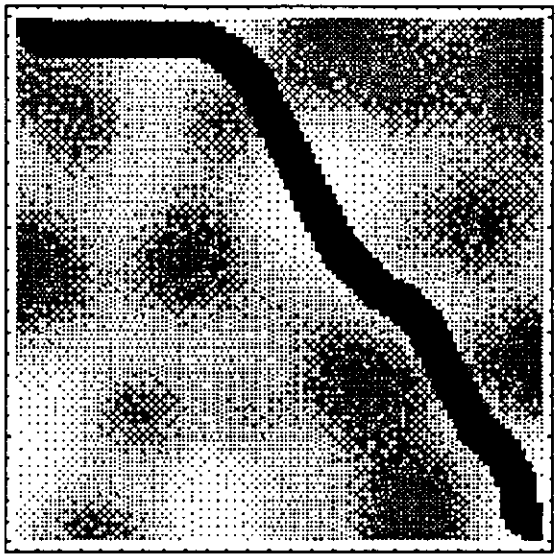


Figure 1.5: An example of an arbitrarily costed region and an optimal prescribed-width path between two opposite corners.

this dissertation, using the framework of VLSI CAD to ground the discussion. Subsequent chapters formulate and analyze each of these problems in detail, and then present algorithms to optimize the relevant objectives.

1.1 VLSI Design and Integrated Circuit Routing

The VLSI design process maps a behavior/function onto silicon [MC80]. The end product of this process is a fabricated chip which may contain many millions of gates within a very small area (e.g., a square centimeter) [PL88]. The primary application areas of this dissertation are embedded within the physical layout phase of VLSI design. Physical layout consists of (i) a **placement** task, where we map functional modules onto specific areas of the chip, and (ii) a subsequent **routing** task, where we connect specified sets of pins together using the free areas between modules to run wires. For example, the hollow dot in Figure 1.6

may be the output of a logic gate inside its containing module M_1, and we may want this logic value to be propagated to several gate inputs in other modules, indicated by black dots in Figure 1.6.

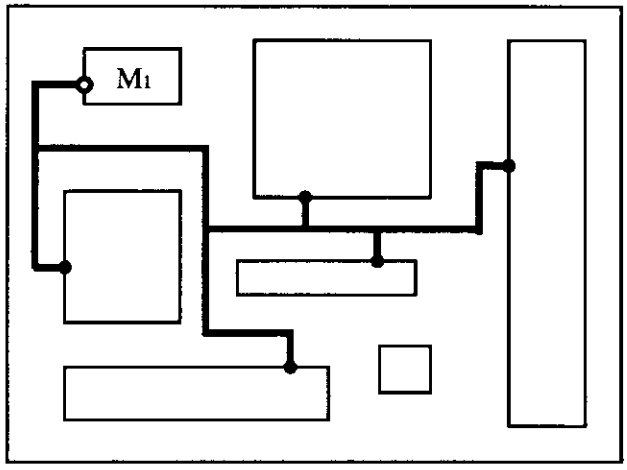


Figure 1.6: An example placement of several modules and a routing which interconnects a set of pins.

At the output of the routing stage, all components have already been arranged in the plane, so we need only ensure electrical connectivity of the prescribed nets. *Global routing* refers to the task of assigning each net to a subset of the routing area based on congestion or topological information, while *detailed routing* entails producing for a single net the actual geometries which realize the required connections [PL88] [Len90]. A high-level goal of this dissertation is to propose efficient algorithms that achieve connectivity topologies possessing various desirable properties.

Advanced VLSI process technologies admit multiple routing layers, where to reduce crosstalk and facilitate algorithm design, a preferred-direction routing methodology is used. With this methodology, physical *wiring layers* are divided into pairs, and within each pair horizontal wire segments are preferentially routed

on one layer, and vertical wiring segments are routed on the other layer. A connection between two wire segments from different layers is called a *via.*

The rectilinear wiring technology implies that the underlying interconnection metric is the *Manhattan* (or L_1) norm[1], where the *distance* between a pair of pins with coordinates (x_1, y_1) and (x_2, y_2) is defined to be $|x_1 - x_2| + |y_1 - y_2|$. A *segment* is an uninterrupted straight horizontal or vertical run of wire; a connection between two pins will consist of one or more wire segments.

Our treatment of interconnection problems in the context of VLSI will use the following terminology. We say that a *pin* or a *terminal* is a given location on a chip. A signal *net* N is a set of pins, with one pin $s \in N$ a designated *source* and the remaining pins *sinks.* A *routing* is a set of wires that connects together, i.e., spans, the pins of a net so that a signal generated at the source will be propagated to all of the sinks.

Sometimes it is convenient to embed the interconnection problem in an underlying *graph* $G = (V, E)$, consisting of a set V of nodes and a set of edges $E \subseteq V \times V$, within which the routing must be constructed. A *subgraph* of G is a graph $G' = (V', E')$ such that $V' \subseteq V$ and $E' \subseteq E$, and $E' \subseteq V' \times V'$. A *path* between two nodes $x, y \in V$ 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 nodes. A graph is a *tree* if it is connected, but the removal of any one of its edges will disconnect it. Since $|N| - 1$ edges suffice to span a net containing $|N|$ pins, any routing will in practice be a tree.

[1]More generally, in the L_p norm the distance function is given by $\Delta = \sqrt[p]{(\Delta x)^p + (\Delta y)^p}$; thus, $p = 1$, $p = 2$ and $p = \infty$ respectively define the Manhattan, Euclidean and Chebyshev norms.

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 the weights of its edges. A *shortest path* in G between two nodes $x, y \in V$, 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 reader is referred to, e.g., [Eve79] for a more detailed treatment of these basic graph-theoretic concepts.

1.2 Overview of the Dissertation

In this section, we summarize the results of each chapter in the dissertation.

1.2.1 Steiner Trees

VLSI design rules dictate a minimum separation between wires, and therefore the area that a net occupies on a chip is proportional to the wirelength required to route it. Moreover, wirelength contributes to signal delay and system power requirements due to the resistance and capacitance of the interconnect. Thus, we seek to minimize the tree cost required to achieve connectivity. This is precisely the minimum Steiner tree problem mentioned above, which for a given point set P asks for a set S of *Steiner points* such that the minimum spanning tree (MST) over $P \cup S$ has minimum cost (Figure 1.7).

The Steiner tree problem is a basic formulation in combinatorial optimization and network design; it is also fundamental to global routing and wirelength estimation for VLSI layout design, where we are interested in minimum-cost Steiner trees connecting the pins of signal nets. It is known that the minimum Steiner

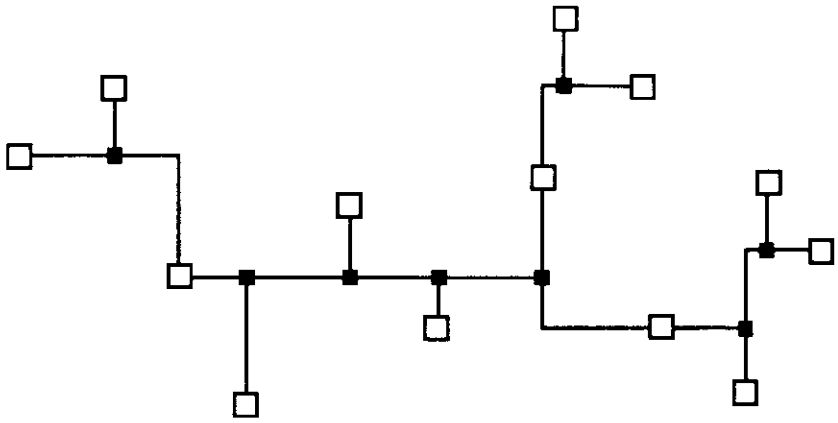


Figure 1.7: An example of a rectilinear Steiner routing of a net. Pins are shown as hollow dots while Steiner points are depicted as dark squares.

tree problem is NP-complete [GJ77], and numerous heuristics have been proposed, as surveyed in [Ric89] and [Win87]. Virtually all previous heuristics for computing rectilinear Steiner trees begin with a minimum spanning tree topology and rearrange edges to induce Steiner points. In Chapter 2, we show that surprisingly, the worst-case performance ratio of such methods is no better than that of the MST itself (i.e., $\frac{3}{2}$) [KR91b] [KR92b].

We therefore propose a new, direct approach which makes a significant departure from such spanning tree based strategies: we iteratively find optimal Steiner points to be added into the routing. Performance results show a very significant reduction in the tree cost over the best previous heuristics. In addition, our method escapes pathological instances for existing approaches [KR90] [KR91a] [KR92a]. We show that the performance ratio of our method can never be as bad as $\frac{3}{2}$, and is in fact bounded by $\frac{4}{3}$ on the entire class of "difficult" instances (which we characterize precisely) where the $\frac{MST}{MRST}$ cost ratio is exactly $\frac{3}{2}$. Using recent results of [BR92] [Zel92] it can be shown that a method very similar to ours has performance ratio bounded by $\frac{11}{8}$, thus settling the longstanding open

question of whether there exists a polynomial-time Steiner tree heuristic with performance ratio smaller than $\frac{3}{2}$ [Hwa76]. Sophisticated computational geometry techniques allow efficient and practical implementation, and we describe a number of variants and extensions.

1.2.2 Pathlength-Balanced Trees

In a high-performance VLSI design, circuit speed is limited in part by the *clock skew*, which is the difference between the longest and shortest arrival times of the clock signal at the synchronizing components. Thus, clock skew minimization has become a very important problem in the design of leading-edge VLSI systems, and has been studied by a number of researchers in recent years [BWM86] [DFW84] [FK82] [Fis90] [JSK90] [RS89] [Tsa91] [WF83].

In Chapter 3, we study the problem of high-performance clock routing, with the goal of developing a clock routing methodology that minimizes skew as well as tree cost [CKR91a] [CKR91b] [KCR90] [KCR91]. We use a linear approximation to signal delay in VLSI interconnections, so that the delay along a source-sink path is proportional to the cost of the path. Thus, the minimum clock skew objective leads to a pathlength-balanced tree formulation. Our clock routing solution is based on the construction of a binary tree by iterated geometric matching. The pathlength-balanced tree cost is on average within a constant factor of the cost of an optimal Steiner tree, and in the worst case is bounded by $O(\sqrt{n})$ for n pins arbitrarily distributed in the unit square.

Our bottom-up construction readily extends to different design styles, and also admits optimizations compatible with more sophisticated delay models, such as Elmore delay [Elm48]. We have tested our algorithms on numerous random ex-

amples and placements of industry benchmark circuits; results are very promising and suggest that our clock routing scheme yields near-zero average pathlength skew, while maintaining tree cost competitive with previously known methods. Recent work by other researchers [BK92a] has established that our bottom-up matching-derived topologies are more amenable than other solutions to improved embeddings which further minimize tree cost.

1.2.3 Bounded-Radius Trees

In the design of VLSI systems, a basic observation is that increased signal delay results in decreased system performance. In particular, signal delay forces a lower bound on the clock period, and the clock period inversely determines the system clock speed. To maximize the performance of the design, we clearly must consider the resulting circuit performance while we construct a routing.

To a first-order approximation, the delay in a circuit is proportional to the maximum source-sink pathlength. Thus, we would like to avoid long paths in the routing. On the other hand, it is undesirable if this can be achieved only at the expense of having to greatly increase the tree cost: more metal mass in the interconnect implies higher capacitance and resistance parameters, which in turn yield increased delay. Minimization of tree cost and minimization of the longest path are conflicting goals.

Motivated by the problem of performance-driven global routing in VLSI, Chapter 4 proposes an algorithm [CKR90] [CKR91c] [CKR92a] [CKR92b] which in some sense melds the two classical constructions of the minimum spanning tree and the shortest paths tree. The approach is based on a new bounded-radius minimum routing tree formulation, and we give an algorithm which simultaneously

minimizes both tree cost and the longest source-sink path, such that both are bounded by small constant factors away from optimal. This method is applicable in arbitrary weighted graphs, and hinges on the following result: for any given value of a non-negative parameter ϵ, we can construct a routing tree with longest interconnection pathlength at most $(1+\epsilon)\cdot R$, and with tree cost at most $(1+\frac{2}{\epsilon})$ times the minimum spanning tree cost, where R is the minimum possible pathlength between the source and the furthest sink.

For Steiner global routing in arbitrary weighted graphs, we achieve longest pathlength at most $(1+\epsilon)\cdot R$, with wiring cost within a factor $2\cdot(1+\frac{2}{\epsilon})$ of the optimal Steiner tree cost. We also show that geometry helps in routing: in the Manhattan plane, the tree cost for Steiner routing improves to $\frac{3}{2}\cdot(1+\frac{1}{\epsilon})$ times the optimal Steiner tree cost, while in the Euclidean plane, the total cost is further reduced to $\frac{2}{\sqrt{3}}\cdot(1+\frac{1}{\epsilon})$ times optimal. Furthermore, our method generalizes to the case where varying wirelength bounds are prescribed for different source-sink paths. Extensive simulations confirm the approach over a large set of examples reflecting several layout styles.

1.2.4 Efficient Connectivity Verification

Once a tree interconnection topology has been determined, it is often crucial that we verify the correctness of its construction. While this question is common in, e.g., the network reliability area, our motivating application stems from multi-chip module (MCM) technology, which has recently emerged as a successful way to increase circuit density without compromising production yields. MCMs eliminate individual integrated circuit packages by mounting a number of bare die on a wiring substrate. Since mounting the die onto a substrate containing faulty

wiring would result in a considerable waste and expense (because a faulty assembled unit must be discarded), it is necessary to first test the substrate wiring prior to mounting. This raises the question of how to efficiently test wiring faults on a "bare" substrate.

Chapter 5 addresses connectivity verification in MCM substrates. We formulate connectivity verification as a problem of tree testing using *k-probes*, and present a linear-time algorithm which computes a minimum set of probes achieving complete connectivity verification [KRW91a] [KRW91b] [KRW92]. Since actual testing also involves the scheduling of probe operations, we prove that probe scheduling is a special type of metric traveling salesman optimization; this affords effective heuristics which achieve a probe scheduling cost no worse than $\frac{3}{2}$ times optimal. Empirical results demonstrate significant reductions in testing costs over the best previous methods, and our method generalizes to alternate probe technologies.

1.2.5 Prescribed-Width Routing

Finally, we examine a problem which arises in applications ranging from printed circuit board (PCB) routing to robotics and autonomous-vehicle navigation: finding an optimal prescribed-width path connecting a given source and destination. Previous path planning formulations assume that the moving agent (and therefore the path) is of zero width; this assumption is usually not realistic, as is apparent when an army has to march over a geographical terrain and requires, say, a mile-wide path (Figure 1.5).

In our formulation, the region is costed arbitrarily (i.e., each point may have a different cost associated with it), and our objective is to minimize the total cost of

the path. For the marching army, areas containing mines or snipers will have very high cost, while clear level ground will be assigned low cost. The cost function can also capture, e.g., incomplete or uncertain knowledge of the environment, and the minimum-width formulation applies when we seek a certain degree of error-tolerance in the path. Previously no efficient algorithm was known for this problem; in Chapter 6 we show how to solve this problem optimally in polynomial time, based on a transformation to network flow [HKR91] [HKR92a]. We also extend this approach to higher dimensions, where it can be used to solve a discrete version of the classical Plateau problem of finding a minimum-area surface which spans a given closed curve [HKR92b] [HKR92c].

We conclude this chapter by briefly describing the research methodology which has guided our work. Most problems encountered in VLSI design automation, including the interconnection formulations which are at the heart of this dissertation, are computationally intractable [Len90]. Such NP-complete problems are not likely to be solved efficiently [GJ79], and we therefore turn to heuristic solutions (even problems solvable in polynomial time may require efficient heuristics because of the large scale of real-world instances). An underlying precept in our work is to prove that our proposed algorithms perform well: typically we might show that the average-case or worst-case quality of the algorithm output is no worse than a constant factor times optimal. Because the practicality and relevance of a solution depends on many issues beyond time and space complexity, we augment the analytical performance bounds with extensive empirical simulation of the proposed algorithms using standard industry benchmarks. This establishes the practicality of the method on real layouts which have non-random distributions, large problem sizes, etc. Ideally, we will prove good analytical performance

bounds and then confirm the bounds by observing reasonable empirical behavior. Indeed, this has been achieved throughout the body of our results.

CHAPTER 2

Steiner Trees

2.1 Introduction

The first interconnection objective that we address is that of minimizing the cost of the connecting topology. This is a fundamental problem in global routing and wirelength estimation for VLSI circuit layout, where we wish to find a Steiner tree connecting the pins of a signal net in the L_1 or Manhattan plane. The *minimum rectilinear Steiner tree* (MRST) problem is as follows:

The MRST Problem: Given a set P of n points in the L_1 plane, determine a set S of Steiner points such that the minimum spanning tree (MST) over $P \cup S$ has minimum cost.

Figure 2.1 shows an MST and an MRST for the same point set.

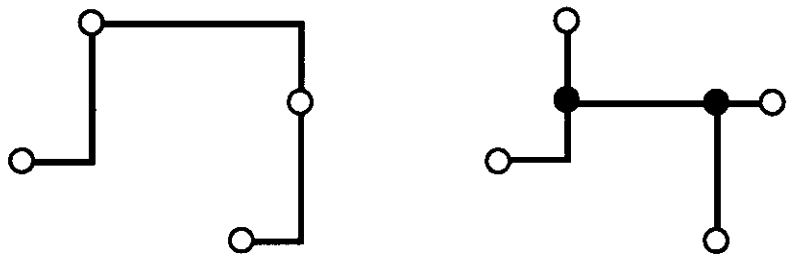


Figure 2.1: MST (left) and MRST (right) for the same 4-point set.

Several results have greatly influenced the progress of research on the MRST

problem. First, Hanan showed in 1966 that if one draws horizontal and vertical gridlines through each of the points in P, there is an MRST whose Steiner points S are all chosen from among the intersection points (the *Steiner candidate set*) in the resulting grid [Han66] (see Figure 2.2).[1]

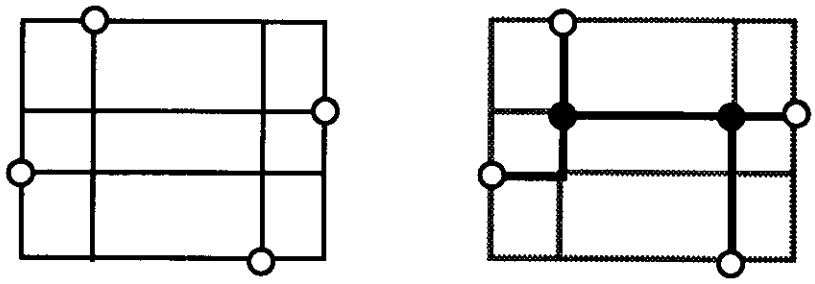


Figure 2.2: Hanan's theorem: an MRST may be found that uses only Steiner points (right) chosen from the intersection points of the horizontal and vertical lines drawn through all points of the original set (left).

Second, Garey and Johnson showed that despite this restriction on the solution space, the MRST problem is NP-complete [GJ77]. Thus, a number of heuristics have been proposed, as surveyed recently in [Ric89] [Win87]. As noted in Chapter 1, a basic goal in attacking intractable problems is to devise *provably good* heuristics, typically in the sense of having bounded worst-case error from optimal. Therefore, a third fundamental result is that of Hwang [Hwa76], who showed that the rectilinear minimum spanning tree over P is a fairly good approximation to the MRST, with worst-case performance ratio of $\frac{3}{2}$, so that if $cost(T)$ denotes the total cost of a tree T, $\frac{cost(MST)}{cost(MRST)} \leq \frac{3}{2}$. The result of Hwang implies that any MST-based strategy which improves upon an initial MST topology will also have performance ratio of at most $\frac{3}{2}$. Thus, a number of Steiner tree heuristics resemble classic MST construction methods.

[1] Snyder [Sny90] has recently generalized Hanan's theorem to higher dimensions.

Examples of this approach include two recent MRST heuristics due to Ho, Vijayan and Wong [HVW90b] and Hasan, Vijayan, and Wong [HVW90a]. The first gives a linear-time construction for the *optimal* rectilinear Steiner tree (RST) derivable from a given MST, i.e., lying within the union of the bounding boxes of the MST edges. The second heuristic also begins with an MST topology, and iteratively adds as many "locally independent" Steiner points as possible. Because the output of these heuristics will not have greater cost than the MST, they retain worst-case performance ratio of $\frac{3}{2}$ by the result of Hwang [Hwa76]. In practice, existing MRST heuristics exhibit very similar performance on random instances (n points chosen from a uniform distribution in the unit square), with the heuristic Steiner tree cost being on average 7% to 9% smaller than MST cost [Ric89] [Win87]. A fundamental open problem has been to find a heuristic method with performance ratio strictly less than $\frac{3}{2}$.

The worst-case bound given by Hwang and such average-case bounds as that given by Steele[2] provide compelling justification for MST-based MRST approximations. However, there are motivations for considering alternative approaches. In the first part of this chapter we give a class of examples which shows that the $\frac{3}{2}$ bound is tight for a wide range of MST-based methods [KCR91], i.e., the MST can be "unimprovable". Thus, it seems unlikely that an MST-based heuristic will have performance ratio strictly less than $\frac{3}{2}$.

[2] A more theoretical, retrospective justification for MST-based approaches is based on asymptotics of subadditive functionals [BHH59] [Ste88] in the L_p plane. Such functionals include the MST cost and the MRST cost. Steele [Ste88] 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., MRST versus MST, and the underlying L_p norm. The theory of subadditive functionals has many implications for VLSI CAD optimization. For example, several VLSI global routers (e.g., TimberWolfSC [Sec88]) use the semiperimeter of a signal net bounding box as a computationally efficient MRST approximation. The growth function above immediately implies that this estimate can be refined by using an $O(\sqrt{n})$ scaling factor, with negligible added CPU cost.

Furthermore, although the MST and MRST costs may have "similar" growth rates, an MST-derived solution is not entirely appropriate to VLSI routing applications. Both the optimal Steiner tree, as well as heuristic MST-based RSTs, will have a linear expected number of Steiner points [BD86][GP68]; these Steiner points in some sense correspond to vias. However, in certain board wiring technologies, or for performance and reliability considerations, having many Steiner points may not be desirable. Ideally, the relative incidence of Steiner points would be a prescribed routing parameter that is a function of technology, performance, or estimated layout congestion; unfortunately, this is not a natural concept when we use an MST-based method.

When we consider the extreme case where extra vias are very expensive, it is natural to ask the following: if we are allowed to introduce *exactly one* Steiner point into a net, where should it be placed? This is the motivation for the Iterated 1-Steiner heuristic, which repeatedly finds the best possible Steiner point and adds it to the point set until no further improvement is possible. The purpose of the second part of this chapter is to introduce the Iterated 1-Steiner method along with several variants. We show that this new approach has a number of practical advantages:

- the average performance of the method is significantly better than all previous MST-based methods, yielding an average improvement of 10 to 11 % over MST cost;
- we can limit the algorithm so that it introduces at most k Steiner points (e.g., in a layout regime where vias are expensive); and
- there are many useful extensions, including randomized, batched and par-

allel variants, as well as applications to alternate routing geometries.

In addition, our approach has a number of theoretical advantages:

- the method can be efficiently implemented by applying elegant computational geometry results;
- the performance ratio of the method is *never* as bad as $\frac{3}{2}$; and
- the performance ratio of our method is not greater than $\frac{4}{3}$ on the entire class of "difficult" instances for which $\frac{cost(MST)}{cost(MRST)} = \frac{3}{2}$, while other known methods have performance ratio arbitrarily close to $\frac{3}{2}$ on these instances.

2.2 Performance Bounds for MST-Based Steiner Tree Heuristics

Hwang's result [Hwa76], along with efficient methods for computing the MST of a planar point set, have motivated a number of MRST heuristics which start with an MST construction and then improve the solution by various methods (e.g., overlapping edges to induce Steiner points). Instances of this approach include the work of Hasan, Vijayan and Wong [HVW90a], Ho, Vijayan and Wong [HVW90b], Hwang [Hwa79a], Lee, Bose and Hwang [LBH76], and Lee and Sechen [LS90].

Other heuristics, such as those discussed by Bern [Ber88], Bern and Carvalho [BD86], Richards [Ric89] and Servit [Ser81], build a Steiner tree by emulating the classical MST constructions of Kruskal [Kru56] and Prim [Pri57]. As noted by Richards [Ric89] and in such surveys as those of Hwang [Hwa78] and Winter [Win87], these methods yield very similar results on random instances, i.e., the heuristic Steiner tree cost is on average 7-9 % less than the MST cost.

Since these Steiner tree constructions cannot have greater cost than the minimum spanning tree, the bound of $\frac{3}{2}$ proved by Hwang is a trivial upper bound on the worst-case performance ratio of these heuristics. However, the actual performance ratio for many MST-based methods has remained unknown. At times there has been hope that certain methods might be provably better than the simple MST approximation (e.g., [Hwa79b]), with the algorithms of Bern [Ber88] and Ho, Vijayan and Wong [HVW90b] being two more recent examples.

We first show that any Steiner tree heuristic in a very general class C of MST-based methods will have worst-case performance ratio arbitrarily close to $\frac{3}{2}$, i.e., the same bound as for the MST itself. We then show that many published heuristics [Ber88] [BD86] [HVW90a] [HVW90b] [Hwa79b] [Ric89] [Ser81] with previously unknown worst-case behavior fall into the class C, and thus we simultaneously resolve a number of error bounds. Our construction also points out a recent incorrect claim in [HVW90b] that the two heuristics of [HVW90b] yield optimal Steiner trees on a certain class of inputs. Furthermore, our examples establish a lower bound of $\frac{3}{2}$ on performance ratios for other heuristics which are not in the class C, e.g., [Hwa79b] [LBH76] [SLL80]. From these results, it seems doubtful that the popular MST-based approach will ever afford a worst-case ratio better than the $\frac{3}{2}$ bound attained by a simple MST. Finally, our examples generalize to d dimensions, where all of these heuristics have error bound of at least $\frac{2d-1}{d}$, improving the previously known lower bound of $\frac{2(d-1)}{d}$ on the performance ratio [Fou84] [GP68].

2.2.1 Counterexamples for Two Dimensions

We begin by discussing two common approaches to constructing a heuristic Steiner tree in the Manhattan plane. We exhibit pathological examples for these methods and then show that the same instances will force a $\frac{3}{2}$ performance ratio for an entire class C of Steiner tree constructions.

The first popular approach to the MRST problem starts with a rectilinear MST and computes a Steiner tree by "overlapping" edges of the MST as much as possible, as shown in Figure 2.3.

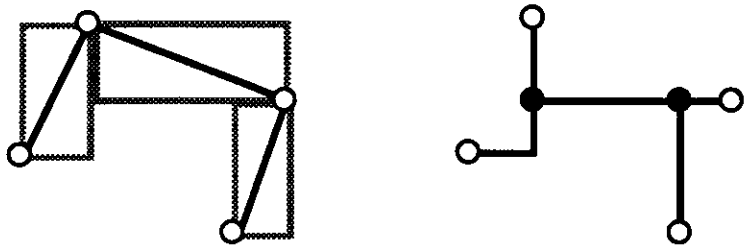


Figure 2.3: Optimal overlap of MST edges within their bounding boxes.

Clearly, the resulting RST cannot have cost greater than the MST cost. A general template for this **MST-Overlap** heuristic is given in Figure 2.4.

MST-Overlap: edge overlap within bounding boxes
Input: A fixed rectilinear MST **Output:** A rectilinear Steiner tree
Determine the least-cost Steiner tree which lies completely within the union of bounding boxes of the MST edges

Figure 2.4: A general template for the MST-Overlap heuristic.

A number of authors have explored this idea, including Hwang [Hwa79a], Lee, Bose and Hwang [LBH76] and Lee and Sechen [LS90]. Ho, Vijayan and

Wong [HVW90b] recently gave a linear-time algorithm for computing the *optimal* RST derivable in this fashion; their method is thus strictly better than those of [Hwa79a] [LBH76] [LS90]. Several researchers conjectured that the worst-case performance ratio of the new method in [HVW90b] was less than $\frac{3}{2}$ and, in fact, equal to $\frac{4}{3}$. However, the example of Figure 2.5 forces a sharp performance bound of *exactly* $\frac{3}{2}$.

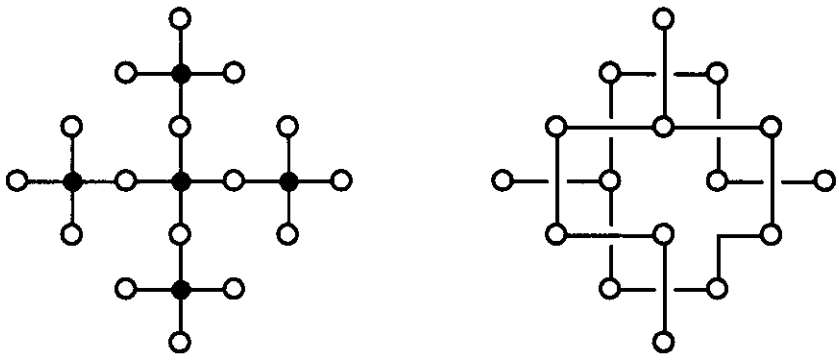


Figure 2.5: An example where the strict equality $\frac{cost(MST-Overlap)}{cost(MRST)} = \frac{3}{2}$ holds. On the left is the MRST ($cost = 20$); any Steiner tree derived from the MST on the right will have $cost = 30$.

Note that the authors of [HVW90b] define a *separable* MST to be one whose edge bounding boxes do not intersect except at their borders, and their linear-time algorithm actually finds optimal overlaps for separable initial MSTs (the MST of Figure 2.5 is not separable). However, even when we insist that the starting MST be separable, we can still force a performance ratio arbitrarily close to $\frac{3}{2}$, as illustrated in Figure 2.6. Figure 2.6(a) shows a separable MST on a point set where the strict equality $\frac{cost(MST)}{cost(MRST)} = \frac{3}{2}$ holds; Figure 2.6(b) shows a perturbation of the point set such that the MST is unique; and Figure 2.6(c) shows the optimal Steiner tree topology for both cases.

The example of Figure 2.6(a) points out a misstatement in reference [HVW90b]

(p. 192): "Both the algorithms produce the optimum Steiner trees for each member of the class of point sets whose optimal RST has a cost which is $\frac{2}{3}$ that of the cost of the MST." This quoted sentence refers to the so-called S-MST and L-MST algorithms, which rely on the separable-MST (SMST) construction on p. 187 of [HVW90b]. It is straightforward to verify that on the point set shown in Figure 2.6(a), the tie-breaking rules of the S-MST construction in [HVW90b] will force the initial separable MST to be exactly that shown in the figure. Edge-overlapping improves this only marginally to the solution shown in Figure 2.6(d), implying a performance ratio arbitrarily close to $\frac{3}{2}$ even though the optimal RST indeed has a cost exactly $\frac{2}{3}$ that of the MST.

Figure 2.6 also shows that a "folklore" heuristic and its variants, described in [Ric89] and ascribed to Clark Thompson by Bern [Ber88] [BD86], has worst-case performance ratio arbitrarily close to $\frac{3}{2}$. We refer to this second generic type of construction as the *Kruskal-Steiner* heuristic, since it is an analog of Kruskal's MST construction [Kru56]. This is shown in Figure 2.7.

Variants in the literature differ mostly in their definitions of the "closest pair" of components, but the example of Figure 2.6(b) is immune to these distinctions. When any variant of Kruskal-Steiner is executed on the point set of Figure 2.6(b), it will start at the leftmost points and alternate between the middle, top, and bottom rows, adding a single horizontal segment to each in turn. Therefore, the Steiner tree will consist entirely of straight horizontal line segments except at the left end, and its cost will be arbitrarily close to $\frac{3}{2}$ times optimal. Note that the ϵ perturbations in Figure 2.6(b) force the alternation between rows and make the heuristic construction completely deterministic.

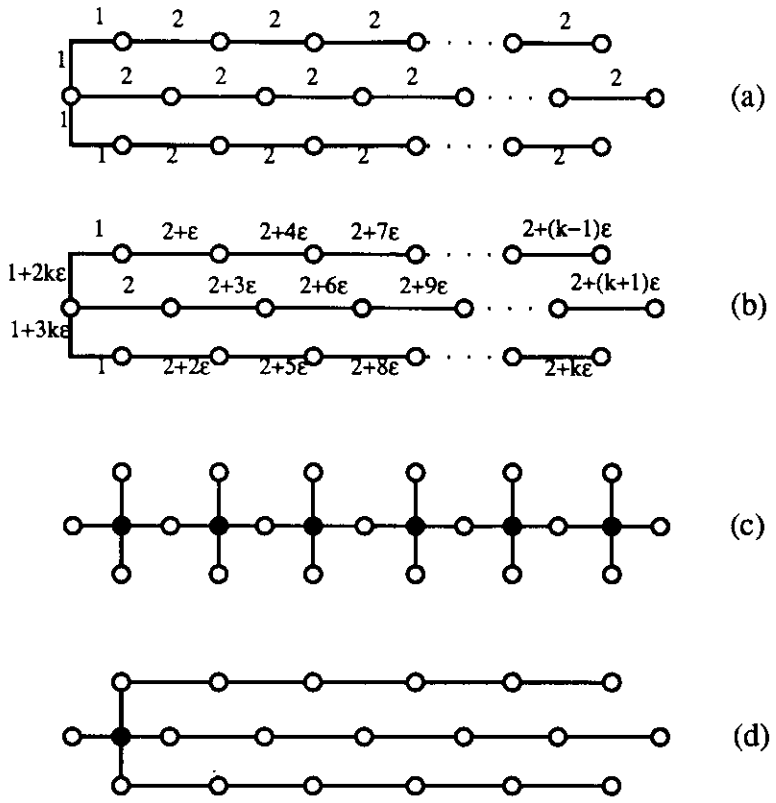


Figure 2.6: An example of a separable MST where $\frac{cost(MST-Overlap)}{cost(MRST)}$ is arbitrarily close to $\frac{3}{2}$. For n points, any Steiner tree derivable from the separable MSTs of (a) or (b) will have cost $2(n-2)$, while the MRST (c) has cost $\frac{4}{3}(n-1)$, yielding a performance ratio arbitrarily close to $\frac{3}{2}$ for large enough n. In (d), we show the best possible RST that can be produced by any MST-Overlap or Kruskal-Steiner heuristic.

Kruskal-Steiner: a Kruskal-like Steiner tree construction
Input: n isolated components (points of P) **Output:** A rectilinear Steiner tree over P
Until one component remains, connect the *closest pair* of components **Output** the single remaining component

Figure 2.7: The Kruskal-Steiner tree construction.

The MST-Overlap and Kruskal-Steiner heuristics form part of a very general class C of *greedy* Steiner tree methods. We now define the class C and show that the example of Figure 2.6 is pathological for the entire class C.

Recall that a Steiner tree is a minimum spanning tree over the union of an input point set P and an added set of Steiner points S. We define an *edge* as any wire connecting two points in $P \cup S$. The following terminology is used to denote progressively more general connection types: (i) a *point-point* connection is an edge between two points of P; (ii) a *point-edge* connection is a wire between a point of P and an edge, inducing up to one new Steiner point; and (iii) an *edge-edge* connection is a wire between two edges, which may induce up to two new Steiner points.

We say that a *greedy* algorithm is one that constructs a solution by iteratively selecting the best among all available alternatives [PS82]. The class C is defined as shown in Figure 2.8, with all algorithms in C being greedy with respect to Manhattan edge length.

Heuristic $H \in C$: greedy Steiner tree construction
Input: n isolated components (points of P) **Output:** A 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-route** any edge within its bounding box
  Optionally at any time, **Eliminate** any edge overlap
Output the single remaining component

Figure 2.8: The class C of greedy Steiner tree heuristics.

Theorem 2.1 *Any heuristic in the class C will have performance ratio arbitrarily close to $\frac{3}{2}$.*

Proof: The MST of the point set depicted in Figure 2.6(b) is clearly unique since all interpoint distances of length < 3 are unique. Even if general connection types are allowed, all connections in the MST will be simple 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. Note that no improvement is possible by edge re-routing within these degenerate edge bounding boxes. Therefore, no heuristic in C can do better than the result depicted in Figure 2.6(d). Since the effect of the optional re-routing of the two non-degenerate connections becomes negligible as the point set grows large, the performance ratio is arbitrarily close to $\frac{3}{2}$. □

We now list a number of published heuristics with previously unknown performance ratio, all of which are shown by Theorem 2.1 to have error bound arbitrarily close to $\frac{3}{2}$. We do not reproduce the various descriptions of each algorithm that we mention here, since it is easy to see from the high-level classification that these algorithms are indeed in C. Algorithms which follow a greedy Kruskal-type construction satisfy the verbatim definition of the class C: these include the methods of Hwang [Hwa79a] and Lee and Sechen [LS90], in addition to methods described in Bern [Ber88] [BD86], Richards [Ric89] and Servit [Ser81]. It is also easy to see that algorithms which start with an initial MST and then overlap rectilinear edges within their bounding boxes, such as those of Hasan, Vijayan and Wong [HVW90a] and Ho, Vijayan and Wong [HVW90b] are members of C, since using only point-point connections will build an MST, and the optional re-routing

is then used to induce edge overlaps. Interestingly, exponential-time methods can also fall into the class C, e.g., the suboptimal branch-and-bound method of Yang and Wing [YW72]. Theorem 2.1 implies that all of these methods have the same worst-case error bound as the simple MST.

Finally, the counterexample of Figure 2.6 also establishes new 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 [Hwa79b] and Lee, Bose and Hwang [LBH76], and the Delaunay triangulation-based method of Smith, Lee and Liebman [SLL80]. This is easy to verify using the point set in Figure 2.6(b): as with the heuristics in C, these latter methods are severely constrained by the nature of the unique minimum spanning tree.

Recent work by De Souza and Ribiero [SR90] constructs an instance similar to that of Figure 2.6 and also discusses the worst-case performance of rectilinear Steiner tree heuristics. However, the work of [SR90] is limited to two dimensions, while Section 2.2.2 below extends our construction to yield new bounds in higher dimensions. More importantly, the work of De Souza and Ribiero is concerned solely with several specific algorithms and thus does not establish a general result such as Theorem 2.1.

2.2.2 Counterexamples for Higher Dimensions

Most rectilinear Steiner tree heuristics, including the MST-Overlap and Kruskal-Steiner variants, extend to higher dimensions and are of special interest for emerging multi-layer packaging and three-dimensional process technologies. However, the examples of Figure 2.5 and Figure 2.6 also generalize to d dimensions and provide new lower bounds on the performance ratio of heuristics in C. In partic-

ular, the example of Figure 2.6 generalizes to $n = (2d-1)k+1$ points for any given positive integer k: 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 derivable from this MST is $2(n-d)$, as illustrated in Figure 2.9 for $d = 3$. Thus, in d dimensions the performance ratio of a heuristic in class C will be arbitrarily close to $\frac{2d-1}{d}$. This value improves the lower bound for the worst-case MST/MRST ratio in higher dimensions from the previously known value of $\frac{2(d-1)}{d}$ [Fou84] [GP68].

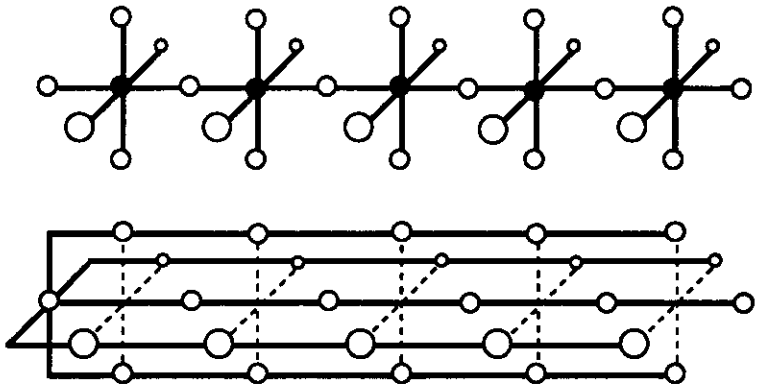


Figure 2.9: For $d = 3$, the MRST (top) has cost $\frac{6}{5}(n-1)$, while any MRST derivable from the MST (bottom) has cost $2(n-3)$, yielding performance ratio arbitrarily close to $\frac{5}{3}$ for n large.

2.3 The Iterated 1-Steiner Approach

Given that conventional MST-based methods have performance ratio no better than the simple MST approximation, we now present a very different and effective Steiner tree heuristic. For a point set P, a 1-*Steiner point* is any point x such that $cost(MST(P \cup \{x\}))$ is minimized, with $cost(MST(P \cup \{x\})) < cost(MST(P))$. A 1-*Steiner tree* is the minimum spanning tree over $P \cup \{x\}$.

Our approach is to iteratively find 1-Steiner points and include them into S. The cost of the MST over $P \cup S$ will decrease with each added point, and we terminate the construction if there is no x such that $cost(MST(P \cup S \cup \{x\})) < cost(MST(P \cup S))$. The *Iterated 1-Steiner* algorithm is thus stated as shown in Figure 2.11, and Figure 2.10 illustrates the execution of Iterated 1-Steiner on a 4-point example.

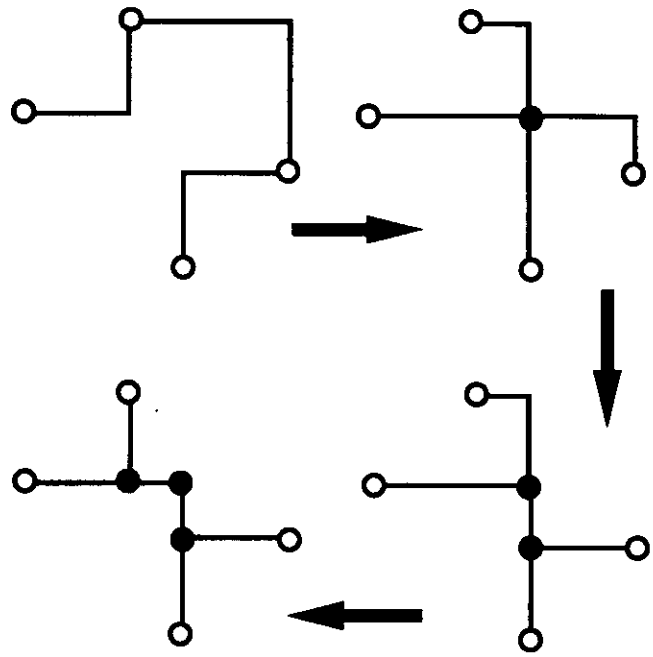


Figure 2.10: Execution of Iterated 1-Steiner on a 4-point example.

By the result of Hanan, we can find a 1-Steiner point by constructing a new MST on $n+1$ points for each element in the Steiner candidate set, then picking the candidate which results in the shortest MST. Each MST computation can be performed in $O(n \log n)$ time [PS85], yielding an $O(n^3 \log n)$ time bound. Note that this is the time required to find just one 1-Steiner point, and that the Steiner tree may contain up to $n-2$ Steiner points [GP68].

Iterated 1-Steiner: Steiner tree construction
Input: A set P of n points **Output:** A rectilinear Steiner tree over P
$S = \emptyset$ **While** $\|S\| < n$ and $\exists$ 1-Steiner point x **Do** $S = S \cup \{x\}$ **Output** MST($P \cup S$)

Figure 2.11: The Iterated 1-Steiner algorithm.

As it turns out, a new 1-Steiner point may be added in $O(n^2)$ time, as described below. A linear number of Steiner points can thus be found efficiently with a total of $O(n^3)$ effort, and finding heuristic solutions with $\leq k$ Steiner points requires $O(kn^2)$ time.

There are numerous extensions to the Iterated 1-Steiner approach, including randomized variants and a very useful amortization of the 1-Steiner point computation which adds an entire set of "independent" Steiner points in a single iteration. Before we discuss these and other variants of this iterative construction, we review the $O(n^2)$ method for finding a 1-Steiner point and analyze the performance ratio of the Iterated 1-Steiner approach.

2.3.1 Finding 1-Steiner Points Efficiently

Georgakopoulos and Papadimitriou [GP87] give an $O(n^2)$ method for computing a 1-Steiner tree for n points in the Euclidean plane. We use a direct adaptation of their method for the Manhattan norm. The idea is summarized as follows:

- A point p cannot have two neighbors in the MST which lie in the same octant of the plane with respect to p. Thus we can fix eight "orientations" at 45-degree intervals, each of which induces a Voronoi-like partition (the

oriented Dirichlet cells) of the plane.

- These eight plane partitions can be computed and overlaid into a "coarsest common partition" within $O(n^2)$ time. The $O(n^2)$ regions of the coarsest common partition are *isodendral*: introducing any point within a given region will result in a constant MST topology.

- The minimum spanning tree on the n points is constructed, and we perform preprocessing in $O(n^2)$ time such that whenever a new point is added to the point set, updating the MST to include the new point requires *constant* time.

- We then iterate through the $O(n^2)$ regions of the overlaid partitions and determine, in constant time per region, the optimal Steiner point in each region. Each such point will induce an MST on $n + 1$ points that can be computed in constant time using the information obtained from the preprocessing. Comparing the costs of these trees and selecting the smallest one will give the minimum-cost MST on $n + 1$ points. The total time for all phases is $O(n^2)$.

There are at most n iterations, each requiring $O(n^2)$ computation, and therefore the time complexity of the Iterated 1-Steiner method is thus $O(n^3)$. Empirical results given below show that Iterated 1-Steiner significantly outperforms all existing heuristics (see Tables 2.1 and 2.2). The actual number of iterations our algorithm performs for random point sets is less than $\frac{n}{2}$ on average.[3]

[3]There are examples where as many as $n-1$ iterations are performed. Thus, our method can generate more Steiner points than would exist in the optimal MRST, although we can easily enforce the $n-2$ bound by removing degree-2 and degree-1 Steiner points without increasing the tree cost.

In surveying the vast Steiner tree literature, it seems that the closest conceptual relative of the Iterated 1-Steiner heuristic is a method of Smith and Liebman [SL79][SLL80] which involves a highly ad hoc examination of a linear-size subset of the candidate Steiner set. Our method seems preferable for several reasons: (i) *performance*: the method in [SL79] gives less than 8% average improvement over MST cost for random point sets and thus seems to fall in with the other methods in the literature, while our method gives up to 11% average improvement[4]; (ii) *efficiency*: [SL79] gives an $O(n^4)$ method, while the Iterated 1-Steiner algorithm is $O(n^3)$; (iii) *simplicity*: the algorithm in [SL79] requires several pages to describe while our method is simply described.

2.3.2 Performance Ratio of Iterated 1-Steiner

Several results can be proved which bound the error of the Iterated 1-Steiner method. A main result is that the output of Iterated 1-Steiner can never be as bad as $\frac{3}{2}$ times optimal. We prove our bound as follows. First, we completely characterize the class of instances having $\frac{cost(MST)}{cost(MRST)} = \frac{3}{2}$, which is a result of independent interest. We then show that the Iterated 1-Steiner algorithm will always find a 1-Steiner point on such instances, whereas previous methods may fail to find any improvement over the MST. Finally, we show that on this class of "difficult" instances, the Iterated 1-Steiner method actually has performance bound $\leq \frac{4}{3}$, significantly better than previous methods.

Lemma 2.2 *Any point set P with $|P| \leq 3$ has $\frac{cost(MST)}{cost(MRST)} \leq \frac{4}{3}$.*

[4]Recently, the method of [CH90] has been reported to also yield up to 11% average improvement over MST cost.

Proof: For $|P| = 2$, $\frac{cost(MST)}{cost(MRST)} = 1$. For $|P| = 3$ we have $cost(MRST) = \frac{R}{2}$, where R is the perimeter of the bounding box of P. On the other hand, we observe that by the pigeonhole principle $cost(MST) \leq \frac{2}{3}R$. It follows that $\frac{cost(MST)}{cost(MRST)} \leq \frac{\frac{2}{3}R}{\frac{R}{2}} = \frac{4}{3}$. □

Definition: A *plus* is a Steiner tree over four points having coordinates of the form $\{(x-r, y), (x+r, y), (x, y-r), (x, y+r)\}$; a plus has exactly one Steiner point at (x, y), the *midpoint* of the plus.

Lemma 2.3 *A plus is the only configuration of four points that achieves a ratio* $\frac{cost(MST)}{cost(MRST)}$ *of* exactly $\frac{3}{2}$ *using exactly one Steiner point.*

Proof: If a 4-point configuration has exactly one Steiner point in its MRST, its topology is the unique one depicted in Figure 5a of [Hwa76] (i.e., that of a plus), and thus the point set must have coordinates of form $P = \{(x-h_1, y), (x+h_2, y), (x, y-v_1), (x, y+v_2)\}$. Again, let R be the perimeter of the bounding box of P. The MRST for P has cost exactly equal to $\frac{R}{2}$, while the MST has cost at most $R - \frac{1}{4}R$ since we can obtain a spanning tree by deleting the largest of the four edges which make up the bounding box. This implies that $\frac{cost(MST)}{cost(MRST)} \leq \frac{3}{2}$ with equality holding only when the largest edge around the bounding box is not greater than $\frac{1}{4}R$, i.e., when all four edges around the bounding box are of 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 assume without loss of generality that $h \leq v$. We now have

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

with equality holding when $h = v$, implying that the configuration is indeed a plus. □

Definition: A *union of pluses* is a Steiner tree with k Steiner points over a point set P of size $|P| = 3k + 1$, where each Steiner point has degree 4 and all four edges incident to any Steiner point are of equal length.

Theorem 2.4 *Any point set having* $\frac{cost(MST)}{cost(MRST)} = \frac{3}{2}$ *has an MRST which is a union of pluses.*

Proof: Following the proof of the result in Hwang [Hwa76], note that for any point set P there is an optimal Steiner tree composed of connected components, each of which has all of its Steiner points forming a chain. Without loss of generality, all of the Steiner points on such a chain are collinear, with the possible exception of the Steiner point at the end of the chain. Using the same upper bound for MST cost and the exact expression for MRST cost as in [Hwa76], we can equate expressions for $\frac{2}{3} \cdot cost(MST)$ and $cost(MRST)$ for the points of any chain:

$$R \cdot (\frac{1}{2} + \frac{2}{3} \cdot \Theta) = R \cdot (\frac{1}{2} + \Theta) \tag{2.1}$$

where R is the length of the bounding box of the points in the chain, and Θ is defined so that $R \cdot \Theta$ is equal to the sum of the distances from all (except the last) of the original points to their adjacent Steiner points in the chain. Equation 2.1 implies that $\Theta = 0$ and thus 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. We already know from Lemma 2.3 that any chain which has only one Steiner point and which exactly achieves the $\frac{3}{2}$ ratio must be a plus. It follows that any optimal Steiner tree which exactly achieves the $\frac{3}{2}$ ratio must be decomposable into a union of pluses. □

Theorem 2.4 completely characterizes the point sets for which $\frac{cost(MST)}{cost(MRST)}$ is *exactly* equal to $\frac{3}{2}$. Using this, we show the following:

Theorem 2.5 *The performance ratio of Iterated* 1*-Steiner is always* $< \frac{3}{2}$.

Proof: If a point set has $\frac{cost(MST)}{cost(MRST)} < \frac{3}{2}$ then even if Iterated 1-Steiner does not find *any* Steiner points, its performance ratio will be less than $\frac{3}{2}$. From Theorem 2.4, we know that any point set for which $\frac{cost(MST)}{cost(MRST)} = \frac{3}{2}$ will have an MRST that is a union of pluses; in this case Iterated 1-Steiner will certainly select and add the midpoint of some plus at the first iteration, hence the overall performance ratio will be strictly less than $\frac{3}{2}$. To see this, note that a spanning tree with cost $\frac{3}{2} \cdot cost(MRST)$ is found by simply replacing every plus in the MRST by an arbitrary tree on the four endpoints of the plus, as shown in Figure 2.12.

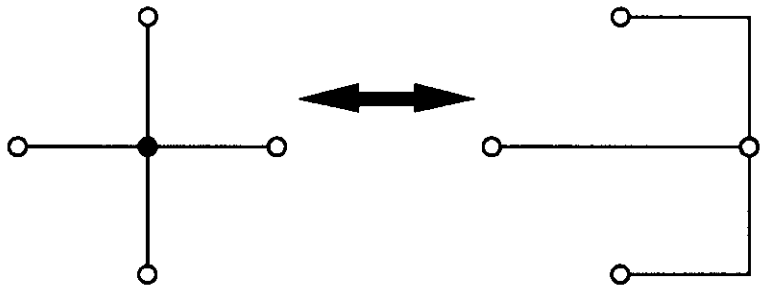


Figure 2.12: Locally replacing each plus (left) with an MST (right).

Adding the midpoint of the plus as a Steiner point will reduce the cost of connecting these four endpoints, and the midpoint is indeed one of the candidates considered during the first iteration of the Iterated 1-Steiner algorithm. Even if there are other 1-Steiner candidates within the convex hull of the four points of the plus, the midpoint trivially gives the greatest possible savings since it achieves a cost improvement of exactly one-third.

Note that existing MST-based methods will have performance ratio arbitrarily close to $\frac{3}{2}$ on unions of pluses, as shown by the example of Figure 2.6. In contrast, we may show the following good performance bound for the Iterated 1-Steiner method:

Theorem 2.6 *The performance ratio of Iterated 1-Steiner on instances whose MRST's are unions of pluses is always* $\leq \frac{4}{3}$.

Proof: When Iterated 1-Steiner selects a midpoint of a plus, at most three midpoints of other pluses may be excluded from future selection. By the greedy selection rule of Iterated 1-Steiner, the three midpoints that are possibly excluded cannot belong to pluses larger than the one selected. Thus if Iterated 1-Steiner selects a plus that is not in the optimal MRST, the 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, as shown in Figure 2.13.

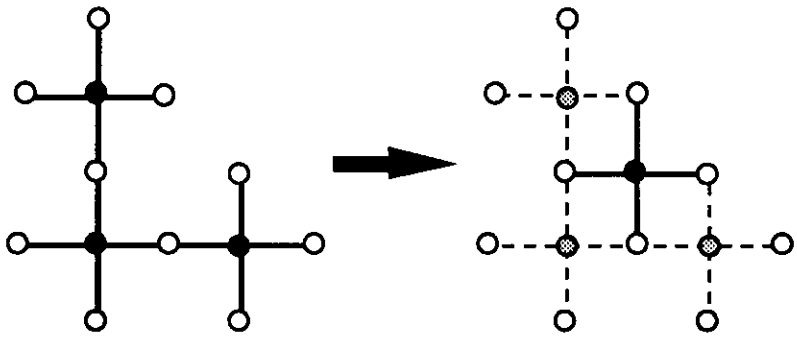


Figure 2.13: Each selected 1-Steiner point may exclude at most three potential 1-Steiner points from future selection; thus at least $\frac{1}{3}$ of the maximum possible savings is achieved.

Each plus represents a savings of $\frac{1}{3}$ of the MST cost over the endpoints of 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 Iterated 1-Steiner is no greater than $\frac{\frac{8}{9} \cdot cost(MST)}{\frac{2}{3} \cdot cost(MST)} = \frac{4}{3}$. □

We note that this bound can probably be tightened by more exhaustive case analysis. Since most nets of practical size have less than six terminals, we now briefly discuss performance bounds for small nets.

Theorem 2.7 *The Iterated* 1*-Steiner heuristic is optimal for* ≤ 4 *points.*

Proof: For three points, there can be at most one Steiner point, and since Iterated 1-Steiner examines all candidates, it is optimal. For a set of four points, the MRST can have zero, one or two Steiner points, and our method is trivially optimal when this number is less than two. When the MRST has two Steiner points, it must have one of the two topologies shown in Figure 2.14 [Hwa76]. A simple case analysis shows that our heuristic always selects both Steiner points, with order of selection irrelevant. □

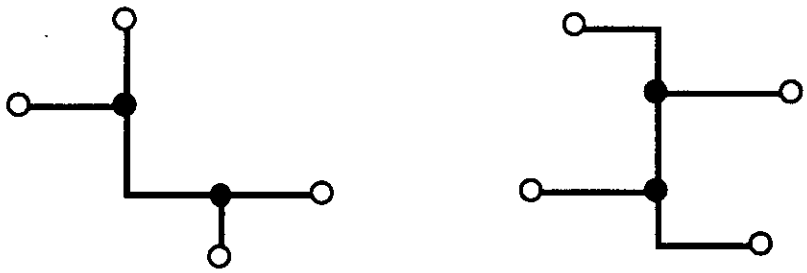


Figure 2.14: The two possible Steiner tree topologies on 4 points.

In contrast, MST-based methods are generally not optimal even for 4-point nets, as shown by the example in Figure 2.15. We have found a 9-point example where Iterated 1-Steiner performs as badly as $\frac{13}{11}$ times optimal (Figure 2.16).

After considerable effort we have not found any instance for which Iterated 1-Steiner has performance ratio worse than $\frac{13}{11}$.

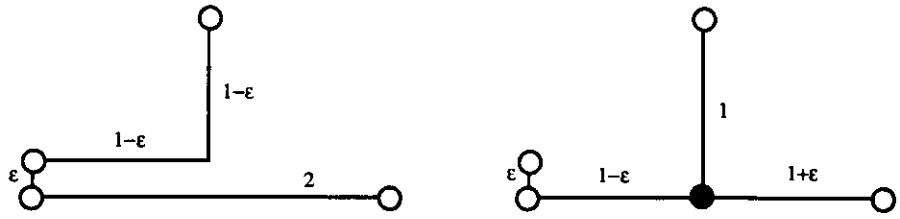


Figure 2.15: A 4-point example where MST-improvement algorithms perform arbitrarily close to $\frac{4}{3}$ times optimal (left); in contrast, Iterated 1-Steiner performs optimally on all point sets of size 4 or less (right).

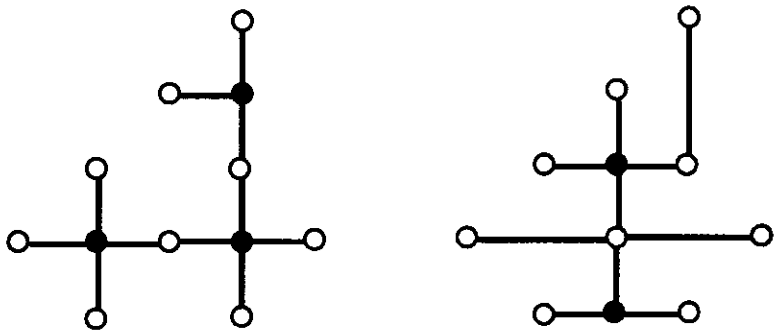


Figure 2.16: A 9-point example where the Iterated 1-Steiner performance ratio is $\frac{13}{11}$; the optimal MRST (left) has cost 11, while the (possible) heuristic output (right) has cost 13.

It is encouraging that while 5- or 6-point examples exist which *force* a performance ratio of $\frac{3}{2}$ for other MRST heuristics in the literature, the worst-case performance ratio of Iterated 1-Steiner for a 5-point example seems to be only $\frac{7}{6}$ (Figure 2.17). In [KR90], we conjectured that the Iterated 1-Steiner method has a performance ratio uniformly bounded away from $\frac{3}{2}$, i.e., there exists a positive constant c such that Iterated 1-Steiner will never have performance ratio greater than $\frac{3}{2} - c$ on any instance. Indeed, using recent results [BR92] [Zel92]

it can be shown that a method very similar to ours (essentially the Batched 1-Steiner described below) has performance ratio bounded by $\frac{11}{8}$, thus settling the longstanding open question of whether there exists a polynomial-time rectilinear Steiner tree heuristic with performance ratio smaller than $\frac{3}{2}$ [Hwa76].

Figure 2.17: A 5-point example where the Iterated 1-Steiner performance ratio is $\frac{7}{6}$. The optimal MRST (left) has cost 6, while the (possible) heuristic output (right) has cost 7.

2.4 Iterated 1-Steiner Variants

In this section we describe several variants of the Iterated 1-Steiner heuristic.

2.4.1 A Random Variant

An important variant of Iterated 1-Steiner is motivated by observing that it may not be necessary to find the *best* candidate Steiner point at each iteration. In particular, the quality of the final tree may be acceptable even if each step simply chooses a *random* improving point. For both this method and the original heuristic, we may simplify the output by removing Steiner points that become degree-1 or degree-2 points in subsequent MSTs; by the triangle inequality the latter can be removed without increasing the MST cost, and the former can trivially be removed. The advantage of this refinement is that performance is not affected, while the final layout is guaranteed by topological constraints to have at most $n-2$ Steiner points. The *Iterated Random* 1-*Steiner* heuristic is given

in Figure 2.18.

Iterated Random 1-Steiner: Steiner tree construction
Input: A set P of n points **Output:** A rectilinear Steiner tree over P
$S = \emptyset$ **While** $\exists$ 1-Steiner points **Do** $S = S \cup \{$ a random improving Steiner point $\}$ **Remove** points of S with degree ≤ 2 in $MST(P \cup S)$ **Output** MST$(P \cup S)$

Figure 2.18: The Iterated Random 1-Steiner algorithm.

Iterated Random 1-Steiner lends itself well to a simple, compact implementation. Empirical performance is on average worse than that of Iterated 1-Steiner, but remains slightly better than MST-derived solutions for typical instances. Iterated Random 1-Steiner will clearly terminate because the MST cost decreases monotonically, but the cost can take on any one of an exponential number of distinct values for certain instances (intuition suggests that there is a polynomial expected upper bound on the number of iterations). A variant which requires that a point cannot return to the layout after it has been deleted will have a trivial $O(n^2)$ bound on the number of iterations, and there exists a family of instances for which Iterated Random 1-Steiner actually produces a quadratic number of Steiner points.

2.4.2 A Batched Variant

Perhaps the most promising variant amortizes computational expense as follows. We use the approach of [GP87] to compute within each isodendral region an optimal 1-Steiner point and its associated MST cost savings; however, instead of

selecting only a Steiner candidate which has highest cost savings, we select a maximal "independent" set of Steiner points, similar to the approach of [HVW90a] (Figure 2.19).

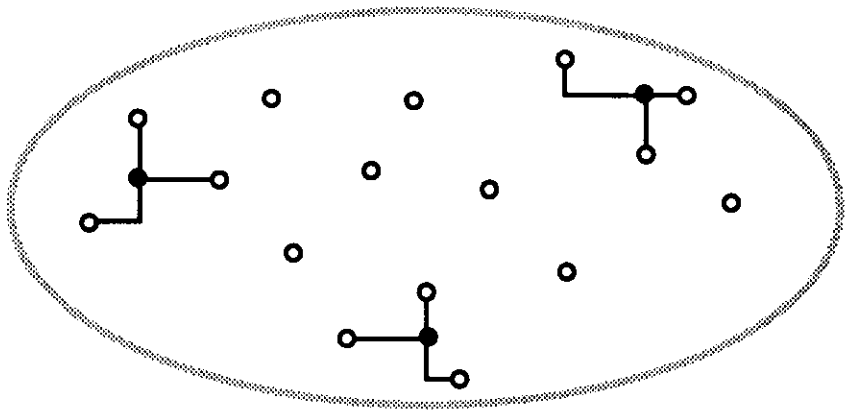


Figure 2.19: The Batched 1-Steiner heuristic: selecting an independent set of of Steiner points in one round.

The criterion for independence is that no candidate Steiner point is allowed to interfere with (i.e., reduce) the potential MST cost savings of any other candidate in the proposed set of Steiner points to be added. In particular, for a set of points P, candidate Steiner points x and y are independent (and thus may be added in the same round) only if

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

where $\Delta MST(P, S) = max(0, cost(MST(P)) - cost(MST(P \cup S)))$, and where we also insist that $\Delta MST(P, \{x\}) > 0$ for x to be a candidate Steiner point. A round of this method is formally described as follows:

- Compute the MST over P in $O(n \log n)$ time using a Voronoi diagram-based method [PS85]. Also construct the weighted undirected graph $G = (P, E)$

where $E = \{(x,y) \mid (x,y)$ is an edge in the Delaunay triangulation over $P\}$ and the cost of each edge in E is the rectilinear distance between its two endpoints.

- 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 [GP87]. This requires a total of $O(n^2)$ time.

- Preprocess the $O(n^2)$ isodendral regions, now treated as a planar subdivision, so that future planar subdivision searches (i.e., determining the planar region in which a given point lies) may be performed in $O(\log n)$ time [PS85]. This preprocessing requires $O(n^2 \log n)$ time, using $O(n^2 \log n)$ space.

- For each candidate Steiner point x, compute the cost savings $\Delta MST(P, \{x\})$ associated with x. We determine the isodendral region to which x belongs in $O(\log n)$ time via planar subdivision search, and let X be the set of potential MST neighbors of x. For each subset $Y \subseteq X$ we 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 [EIR90]. 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. By Hanan's theorem there are at most n^2 candidate Steiner points, and therefore the time for this entire phase is $O(n^2 \log n)$.

- Next, 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 of independent candidate Steiner points to be added during this round, by successively 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 point set 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 of deletion of a single point can be determined in time $O(\log n)$ [EIR90], bringing the total time for this entire step to $O(n^2 \log n)$.

Once a maximal set of independent Steiner points has been determined, it is inserted into P. We iterate this process with $P := P \cup S$ until we reach a round which fails to add a Steiner point to P. Clearly, the total time required for each round is $O(n^2 \log n)$. The *Batched* 1-*Steiner* algorithm is summarized in Figure 2.20.

Batched 1-Steiner: Steiner tree construction
Input: A set P of n points **Output:** A rectilinear Steiner tree over P

While $\exists$ a set $S = \{x \mid \Delta MST(P, \{x\}) > 0\} \neq \emptyset$ **Do**
  For $x \in \{S$ in order of non-increasing $\Delta MST\}$ **Do**
  If $\Delta MST(P - S, \{x\}) \leq \Delta MST(P, \{x\})$ **Then** $P = P \cup \{x\}$
Output MST(P)

Figure 2.20: The Batched 1-Steiner algorithm.

Empirical data indicates that the number of rounds required grows considerably more slowly than the number of Steiner points produced. For example, experimental results on point sets of size 40 show an average number of about 17 Steiner points produced (with a maximum of 22), while the average number of rounds for Batched 1-Steiner is only 2.05 (with a maximum of 4). We conjecture that the number of rounds grows only sub-linearly with the number of points.

2.5 Experimental Results

We implemented the Iterated 1-Steiner, the Iterated Random 1-Steiner, and Batched 1-Steiner heuristics, along with several existing methods, using ANSI C in both the Sun-4 and Apple Macintosh environments. An example of the output of Iterated 1-Steiner is shown in Figure 2.21.

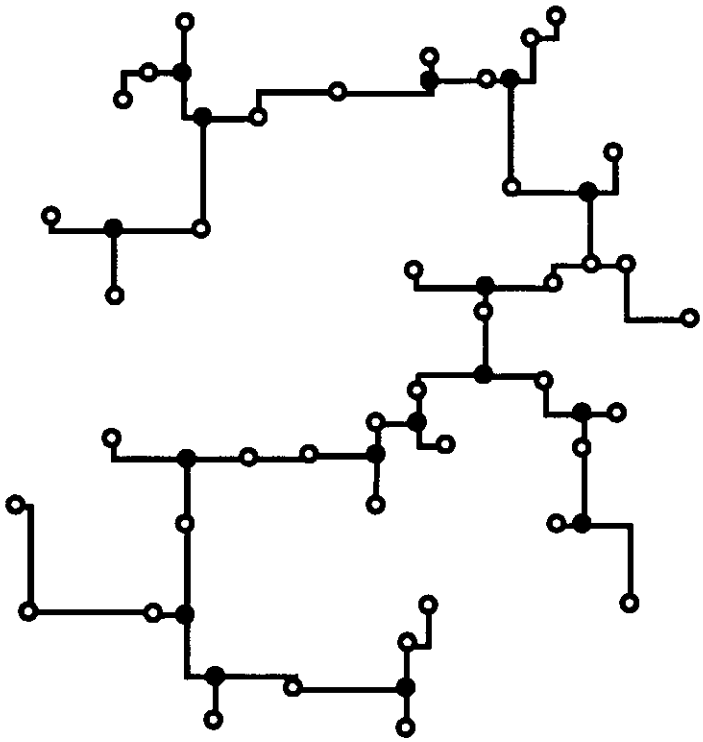


Figure 2.21: An example of the output of Iterated 1-Steiner on a random point set (hollow dots). The Steiner points produced by our algorithm are denoted by dark solid dots.

We have compared the Iterated 1-Steiner and Iterated Random 1-Steiner heuristics with the standard Corner and Prim methods described below. For typical values of n, 5000 n-point instances were solved using all methods. The instances were generated randomly from a uniform distribution in a 1000×1000 grid; such instances are statistically indistinguishable from the pin locations of

actual cell-based layouts, and are the standard testbed for Steiner tree heuristics [Ric89]. The results are summarized in Tables 2.1 through 2.3, and are depicted graphically in Figure 2.22.

	Corner			Prim		
$\|P\|$	Min Perf.	Ave Perf.	Max Perf.	Min Perf.	Ave Perf.	Max Perf.
3	0.00	6.60	24.93	0.00	4.10	24.93
4	0.00	7.85	29.11	0.00	5.71	29.11
5	0.00	8.09	25.07	0.00	6.20	25.07
6	0.00	8.16	27.00	0.00	6.34	22.56
7	0.00	8.12	23.15	0.00	6.45	23.24
8	0.00	8.20	22.57	0.00	6.52	19.97
9	0.37	8.27	20.94	0.00	6.57	19.16
10	0.39	8.20	19.29	0.00	6.45	17.34
12	1.02	8.20	19.27	0.00	6.44	18.40
14	1.93	8.25	18.04	0.12	6.46	16.09
16	2.46	8.24	16.65	0.41	6.48	14.34
18	3.07	8.15	16.12	0.72	6.60	13.24
20	3.26	8.23	14.21	0.84	6.31	11.89
25	3.10	8.37	13.70	1.81	6.47	13.01
30	3.51	8.44	12.70	2.19	6.67	11.20
35	4.98	8.35	13.18	2.85	6.53	11.32
40	4.35	8.50	12.78	2.89	6.68	11.40

Table 2.1: Steiner tree heuristic statistics. These performance figures denote average percent improvement over MST cost.

2.5.1 Incremental Calculations

These tables also show that even when restricted to a k-point or k-round solution, both the Iterated and Batched 1-Steiner algorithms still perform well, with a large portion of the cost savings (as a percent improvement over the MST cost) occurring in the first several iterations/rounds. Because of this, it seems reason-

	Iterated 1-Steiner					
$\|P\|$	Min Perf.	Ave Perf.	Max Perf.	Min # SPs	Ave # SPs	Max # SPs
3	0.00	6.94	24.93	0	0.66	1
4	0.00	8.73	29.11	0	1.09	3
5	0.00	9.41	25.07	0	1.59	4
6	0.00	9.74	27.00	0	2.03	5
7	0.00	9.99	23.80	0	2.52	5
8	0.31	10.10	25.46	1	2.96	6
9	0.80	10.19	22.57	1	3.42	7
10	0.39	10.15	22.62	1	3.83	7
12	1.50	10.26	19.93	2	4.73	8
14	2.45	10.34	21.77	2	5.60	9
16	2.95	10.43	19.90	3	6.61	10
18	3.61	10.35	17.19	4	7.42	11
20	5.40	10.52	16.25	5	8.35	13
25	4.07	10.64	15.62	6	10.59	15
30	5.00	10.90	15.60	9	12.80	17
35	6.46	10.74	15.32	10	15.04	20
40	5.78	10.93	14.76	12	17.37	22

Table 2.2: Steiner tree heuristic statistics (continued). These performance figures denote average percent improvement over MST cost. Also given are statistics regarding the number of Steiner points produced.

able for a layout system to use our method for "k-Steiner point routing"; this will be accomplished in $O(kn^2)$ time and as noted above, the parameter k can reflect via costs, routing congestion, performance, and other manufacturability or reliability attributes. Similar arguments can be made for a k-round implementation of the Batched 1-Steiner variant, which will take $O(kn^2 \log n)$ time. For Batched 1-Steiner, the advantages of incremental calculation are dramatic: on 40-point instances, over 95% of the total improvement occurs in the first round, and over 99% of the improvement occurs in the first two rounds. Results detailing the

	Iterated Random 1-Steiner					
$\|P\|$	Min Perf.	Ave Perf.	Max Perf.	Min # SPs	Ave # SPs	Max # SPs
3	0.00	6.94	24.93	0	0.66	1
4	0.00	8.70	29.11	0	1.20	3
5	0.00	9.15	25.07	0	1.72	4
6	0.00	9.35	27.00	0	2.23	4
7	0.00	9.42	23.80	0	2.76	5
8	0.00	9.36	25.46	1	3.22	6
9	0.00	9.38	22.14	1	3.73	7
10	3.30	9.24	19.99	1	4.18	7
12	1.26	9.16	19.93	1	5.11	9
14	0.00	8.93	21.73	1	5.98	11
16	0.00	8.85	19.35	1	6.81	11
18	0.09	8.78	15.99	1	7.71	12
20	0.04	8.61	15.98	1	8.50	14
25	0.42	8.21	14.39	1	10.34	17
30	0.16	8.44	15.00	1	12.50	19
35	0.44	7.75	14.82	1	13.56	24
40	0.22	7.90	13.64	1	16.00	26

Table 2.3: Steiner tree heuristic statistics (continued). These performance figures denote average percent improvement over MST cost. Also given are statistics regarding the number of Steiner points produced.

nature of the incremental improvements are given in Tables 2.4 and 2.5 below.

2.5.2 On Meta-Heuristics

For a number of combinatorial problems, the following concept of a *meta-heuristic* is natural. Given an instance of a problem and m different heuristics (algorithms) H_1, H_2, ...,H_m, the meta-heuristic Meta($H_1, H_2, \ldots, H_m$) will output the best among the m outputs of heuristics H_1, H_2, ...,H_m. Intuitively, several methods can trade off in their "areas of expertise", so while the meta-heuristic has the same

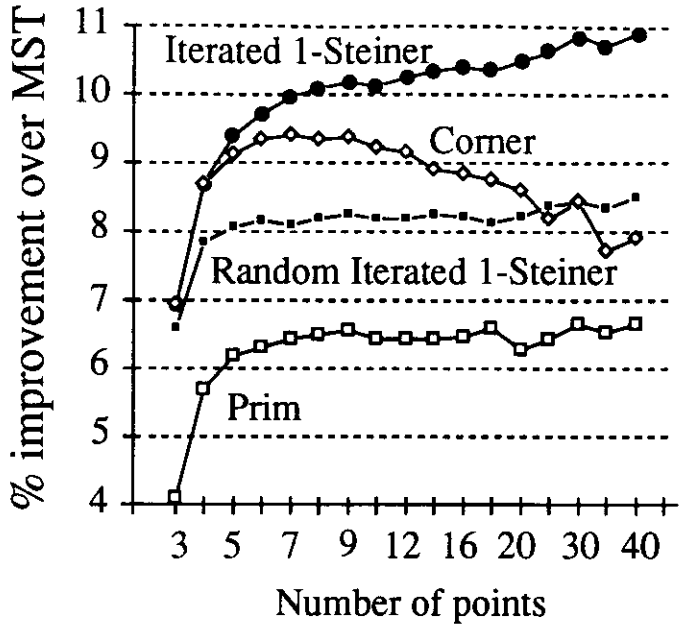


Figure 2.22: Performance comparison of heuristics; the horizontal axis represents the number of points per set, while the vertical axis represents average percent improvement over MST cost.

asymptotic time complexity as its slowest component heuristic, its performance is often significantly better than that of any individual component heuristic.

To illustrate this phenomenon, we give computational results from implementations of *Corner* (from "corner-flipping"; this method gives results similar to the method of [HVW90b]) and *Prim*, a simple analog of Prim's MST heuristic construction that is similar to MRST heuristics analyzed in [Ric89]. Table 2.6 shows that Meta(Corner, Prim) gives an average performance of about half a percent better than Corner alone, although the average performance of Prim is about two percent worse than that of Corner. In contrast, Meta(Prim, Corner, Iterated 1-Steiner) gives essentially the same performance as Iterated 1-Steiner alone, implying that Iterated 1-Steiner strictly dominates the other methods (Table 2.7). This is a very important aspect: it suggests that the Iterated 1-Steiner

	# Rounds			Improvement per round			
$\|P\|$	Min	Ave	Max	1	2	3	4
3	0	0.67	1	7.66	0.00	0.00	0.00
4	0	0.96	2	8.83	0.18	0.00	0.00
5	0	1.07	4	9.30	0.30	0.00	0.00
6	0	1.15	4	9.48	0.38	0.01	0.00
7	0	1.19	5	9.61	0.40	0.01	0.00
8	0	1.24	4	9.75	0.45	0.01	0.00
9	1	1.28	5	9.75	0.47	0.02	0.00
10	1	1.33	6	9.82	0.49	0.02	0.00
12	1	1.40	4	9.79	0.48	0.02	0.00
14	1	1.48	5	9.87	0.53	0.02	0.00
16	1	1.56	4	9.90	0.54	0.02	0.00
18	1	1.61	4	9.85	0.54	0.03	0.00
20	1	1.65	4	9.81	0.58	0.03	0.00
25	1	1.77	5	9.97	0.52	0.03	0.00
30	1	1.93	4	10.14	0.69	0.03	0.01
35	1	2.00	4	10.09	0.58	0.02	0.00
40	1	2.05	4	9.80	0.55	0.04	0.01

Table 2.4: Average improvement percent per round for Batched 1-Steiner.

method will universally give "reasonably good" solutions. For completeness, Table 2.8 gives the empirical performance of Meta(Prim, Corner, Iterated 1-Steiner, Iterated Random 1-Steiner).

The meta-heuristic is a general algorithmic phenomenon that applies to numerous other problems and subareas of computer science. There is very little evidence in the literature to indicate that this phenomenon, especially for heuristics, has received the attention it deserves. Particularly in light of advances in parallel computation and hardware implementation of algorithms, such composite methods should become a highly fertile avenue of research in practical optimization.

	Improvement per Steiner point									
$\|P\|$	1	2	3	4	5	6	7	8	9	10
3	7.66	0.00								
4	8.39	0.62	0.00							
5	7.78	1.72	0.10	0.00						
6	7.06	2.37	0.42	0.03	0.00					
7	6.34	2.73	0.82	0.11	0.01	0.00				
8	5.90	2.83	1.15	0.30	0.04	0.00				
9	5.38	2.87	1.36	0.50	0.12	0.01	0.00			
10	5.01	2.83	1.51	0.69	0.24	0.04	0.00			
12	4.33	2.71	1.65	0.91	0.46	0.20	0.04	0.00		
14	3.87	2.56	1.70	1.07	0.63	0.37	0.17	0.05	0.01	0.00
16	3.44	2.39	1.71	1.17	0.77	0.50	0.31	0.15	0.04	0.01
18	3.16	2.23	1.65	1.20	0.83	0.56	0.39	0.26	0.11	0.03
20	2.88	2.09	1.59	1.22	0.89	0.62	0.43	0.25	0.21	0.12
25	2.45	1.84	1.46	1.18	0.94	0.74	0.57	0.41	0.30	0.24
30	2.17	1.68	1.36	1.12	0.93	0.78	0.62	0.51	0.39	0.31
35	1.82	1.44	1.23	1.07	0.92	0.79	0.67	0.54	0.45	0.38
40	1.74	1.12	0.94	0.81	0.71	0.62	0.56	0.48	0.43	0.37

Table 2.5: Average improvement per point for Batched 1-Steiner.

$\|P\|$	Corner Ave %	Prim Ave %	Meta Ave %
5	8.022	6.162	8.580
10	8.155	6.455	8.584
15	8.352	6.548	8.613
20	8.240	6.392	8.424

Table 2.6: Meta(Corner,Prim) outperforms its component heuristics. Figures represent average percent improvement over MST cost.

2.6 Remarks and Extensions

In this chapter, we began by showing that conventional (i.e., MST-based) approaches to Steiner tree approximation fail in the sense that their worst-case

$\|P\|$	Corner Ave %	Prim Ave %	1-Steiner Ave %	Meta Ave %
10	8.18	6.54	10.23	10.26
12	8.16	6.30	10.25	10.28
15	8.19	6.54	10.33	10.35
17	8.16	6.43	10.38	10.39
18	8.25	6.48	10.51	10.52
22	8.29	6.49	10.45	10.46
25	8.38	6.53	10.65	10.66

Table 2.7: Iterated 1-Steiner dominates both Corner and Prim. Figures represent average percent improvement over MST cost.

performance is no better than that of the MST itself. We have therefore departed from MST-based methods and proposed a new approach to the minimum rectilinear Steiner tree problem. Our method yields results that reduce tree cost by a very significant amount over the best previous methods. Furthermore, it is the first heuristic which has been shown to have performance ratio less than $\frac{3}{2}$; in fact, the performance ratio of Iterated 1-Steiner is $\leq \frac{4}{3}$ on the entire class of instances where the ratio $\frac{cost(MST)}{cost(MRST)}$ is exactly equal to $\frac{3}{2}$. The algorithm has practical asymptotic complexity due to an efficient implementation which uses methods from computational geometry, and which parallelizes readily. Randomized and batched variants of the algorithm have also proved successful.

Our approach extends easily to three-dimensional global routing formulations. In higher dimensions, MST-improvement methods become more complicated due to additional edge orientations, and thus the benefits of the constructive Iterated 1-Steiner strategy are even more apparent. Empirical results for three-dimensional problem instances seem quite favorable. The Iterated 1-Steiner approach also succeeds in the presence of non-orthogonal wiring directions [SW92].

	Meta-Heuristic					
$\|P\|$	Min Perf.	Ave Perf.	Max Perf.	Min # SPs	Ave # SPs	Max # SPs
3	0.00	6.94	24.93	0	0.66	1
4	0.00	8.73	29.11	0	1.09	3
5	0.00	9.45	25.07	0	1.60	4
6	0.00	9.80	27.00	0	2.07	5
7	0.00	10.07	23.80	0	2.57	5
8	0.31	10.19	25.46	1	3.02	6
9	0.80	10.28	22.57	1	3.49	7
10	0.39	10.25	22.62	1	3.94	7
12	1.50	10.35	19.93	2	4.86	9
14	2.45	10.42	21.77	3	5.76	10
16	3.01	10.51	19.90	3	6.78	11
18	4.09	10.43	17.19	4	7.63	11
20	5.40	10.60	16.25	5	8.58	14
25	4.07	10.67	15.62	6	10.83	16
30	5.00	10.90	15.60	10	13.11	18
35	6.46	10.78	15.32	12	15.31	22
40	5.78	10.97	14.76	12	17.78	26

Table 2.8: The empirical performance of Meta(Prim, Corner, Iterated 1-Steiner, Iterated Random 1-Steiner). These performance figures denote average percent improvement over MST cost.

There exists an infinite family of higher-dimensional point sets (Figure 2.9) for which our Iterated 1-Steiner scheme performs *optimally* yet all other MST-based heuristics can perform as badly as $\frac{2d-1}{d}$ times optimal in d dimensions, which is no better than the MST cost for the same point sets. Furthermore, Theorem 2.6 can be generalized to arbitrary dimension d, where the performance of Iterated 1-Steiner would be no worse than $\frac{4d^2-5d+2}{d(2d-1)}$, e.g., for $d = 3$ we obtain worst-case bound of $\frac{23}{15}$ for "difficult" point sets which have an MRST that is a union of "pluses".

We have conjectured [KR90] [KR92a] [KR92b] that $\frac{2d-1}{d}$ is not only a lower bound, but also a general upper bound for the worst-case performance ratio in d dimensions of any MRST heuristic in C, i.e.,

$$\frac{cost(MST)}{cost(MRST)} \leq \frac{2d-1}{d}$$

We also believe that $\frac{2d-1}{d}$ is the higher-dimensional analogue of Hwang's values of $\frac{3}{2}$ in two dimensions. Two additional avenues for future research are: (i) the concept of a meta-heuristic introduced above, which may be effective in addressing other optimizations, and (ii) deriving even tighter bounds for the performance ratio of the Iterated 1-Steiner method.

CHAPTER 3

Pathlength-Balanced Trees

3.1 Introduction

A second interconnection objective arises due to synchronization considerations, i.e., the interconnection topology should allow signals to arrive at their destinations simultaneously. Our motivating application is clock tree synthesis for large, high-performance VLSI designs. In a synchronous VLSI design, limitations on circuit speed are determined by two factors: (i) the delay on the longest path through combinational logic, and (ii) the clock skew among the synchronizing components, where skew is defined to be the maximum difference between arrival times of the clocking signal at any pair of destinations. With advances in VLSI fabrication technology, the switching speed of combinational logic has increased dramatically. Thus, the clock skew induced by non-symmetric clock distribution has become a more significant limitation on circuit performance.

Clock skew minimization has been studied by a number of researchers in recent years. For example, H-tree constructions have been used extensively for clock routing in regular systolic arrays [BWM86] [DFW84] [FK82] [WF83]. Although the H-tree structure can significantly reduce clock skew, it is applicable only when all of the synchronizing components are identical in size and are placed in a symmetric array. Ramanathan and Shin [RS89] proposed a clock distribution

scheme for building-block design where all blocks are organized in a hierarchical structure. They assume that all clock entry points are known at each level of the hierarchy and, moreover, that the number of blocks at each level is small since an exhaustive search algorithm is used to enumerate all possible routes.

Burkis [Bur91] and Boon et al. [BBB89] have proposed hierarchical clock tree synthesis approaches involving geometric clustering and buffer optimization at each level. More powerful clock tree resynthesis or reassignment methods were used by Fishburn [Fis90] and Edahiro [Eda90] to minimize the clock period while avoiding hazards or race conditions; Fishburn employed a mathematical programming formulation, while Edahiro used a clustering-based heuristic augmented by techniques from computational geometry. All of these methods are highly limited in their application, either to small problem sizes by their algorithmic complexity and reliance on strong hierarchical clustering [Eda90] [Fis90] [BBB89] [Bur91] [RS89], or by their inability to even construct an actual clock routing topology. In contrast, we are interested in clock tree synthesis for "flat" problem instances with many synchronizing elements, as will arise in large standard-cell, sea-of-gates, and multi-chip module designs.

For designs with many small cells, the H-tree approach cannot be used since synchronizing components may be of different sizes and may be in arbitrary locations in the layout. Thus, Jackson, Srinivasan and Kuh [JSK90] proposed the "method of means and medians" (MMM) algorithm, which recursively partitions the set of clock terminals into two equal-cardinality subsets, and then connects the center of mass of the entire set to the centers of mass of the two subsets. Although it was shown that the maximum difference in pathlength from the root to different synchronizing components is bounded by $O(\frac{1}{\sqrt{n}})$ in the average case

one may easily construct small examples for which the pathlengths between clock source and clock terminals in their solution may vary by as much as half the chip diameter [KCR91]. In some sense, this reflects an inherent weakness of the top-down approach, which is that it can commit to an unfortunate topology very early in the construction.

In this chapter, we consider the problem of high-performance clock routing via pathlength-balanced tree constructions, with the goal of minimizing skew while incurring little added wiring expense. We first cast the problem in a geometric setting, reflecting cell-based design methodologies such as standard-cell or gate array. We then extend our method to general cell or building-block layouts, where the wiring is restricted to specific channels so that an underlying distance graph is induced. We present a basic algorithm and several variants, which minimize skew by constructing a tree that is balanced with respect to root-leaf pathlengths. The approach is based on geometric matching: we start with a set of trees, each containing a single terminal of the clock signal net, and at each level of the clock tree topology we combine the trees into bigger trees using the edges of an optimal geometric matching. The end result is a binary tree whose leaves are the terminals in the clock signal net and whose root is the clock entry point. Our method is particularly suitable for designs which employ a single large buffer to drive the clock tree, and we note that there are a number of reasons for such a design choice, as discussed in [BWM86].

Our algorithm always yields perfect pathlength-balanced trees for inputs of four or less terminals. Extensive experimental results indicate that even for large clock signal nets, the pathlength skew of the tree constructed by our algorithm remains essentially zero. This performance is obtained without undue sacrifice of

tree cost: we prove that on average our tree cost is within a constant factor of the optimal Steiner tree cost. Furthermore, our worst-case tree cost is bounded by $O(\sqrt{n})$ for n terminals in the unit square, which is the same bound as for the worst-case optimal Steiner tree cost.

Since the work in [JSK90] addresses minimum-skew clock routing for cell-based designs, we used an implementation of the MMM algorithm, similar to that of [JSK90] for comparison purposes. For uniformly distributed nets of up to 1024 terminals in the unit square, our method produces trees with near-zero pathlength skew both in the average case and worst case, with tree cost significantly lower than that produced by MMM. In addition, routing results for layouts of the MCNC Primary1 and Primary2 benchmarks are significantly better than those reported in [JSK90]; we obtain perfectly balanced root-leaf pathlengths in the tree using several percent less tree cost than MMM. Actual clock skews for our benchmark routings, as determined by using the circuit simulation package HSPICE, are very reasonable.

Extension of our method to general cell design is accomplished by generalizing the notion of matching to weighted graphs. Thus, our algorithm produces a routing tree that is embedded in the channel intersection graph [DAK85] of an arbitrary building-block layout. The trees produced by our method attain almost zero pathlength skew with only modest tree cost penalty. Experimental results show that the pathlength skew of our routing tree is less than 2% of that of a heuristic Steiner tree. This is achieved on average with less than 50% increase in tree cost over the Steiner tree. Simulation results using HSPICE confirm that the actual clock skew of our tree is also considerably less than that of the Steiner tree.

The remainder of this chapter is organized as follows. Section 3.2 defines a number of basic concepts and gives a precise formulation of the pathlength skew minimization problem. In Section 3.3 we present the clock routing algorithm in detail for cell-based designs; Section 3.4 extends the algorithm to general cell layouts. Experimental results of our algorithm and comparisons with previous methods are presented in Section 3.5, and Section 3.6 concludes with possible extensions of the method.

3.2 Preliminaries

A synchronous VLSI circuit consists of two types of elements, synchronizing elements (such as registers) and combinational logic gates (such as NAND gates and NOR gates). The synchronizing elements are connected to one or more system-wide clock signals. Every closed path in a synchronous circuit contains at least one synchronizing element (Figure 3.1). The speed of a synchronous circuit is chiefly determined by the clock signal period(s). It is well known [Bak90] [JSK90] that the clock period C_P of each clock signal net must satisfy the inequality:

$$C_P \geq t_d + t_{skew} + t_{su} + t_{ds}$$

where t_d is the delay on the longest path through combinational logic, t_{skew} is the clock skew, t_{su} is the set-up time of the synchronizing elements, and t_{ds} is the propagation delay within the synchronizing elements.

The term t_d itself can be further decomposed into $t_d = t_{d\_interconnect} + t_{d\_gates}$ where $t_{d\_interconnect}$ is the delay associated with the interconnect of the longest path through combinational logic, and t_{d_gates} is the delay through the combinational logic gates on this path. As VLSI feature sizes become smaller, the

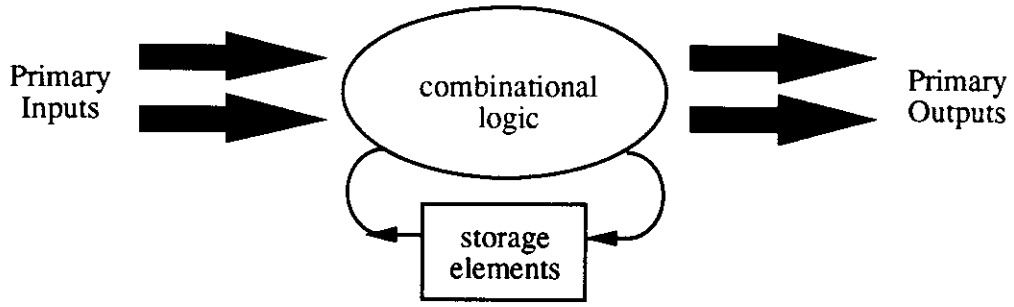


Figure 3.1: A typical combinational circuit.

terms t_{su}, t_{ds}, and t_{d_gates} all decrease significantly. Therefore, as noted above, $t_{d\_interconnect}$ and t_{skew} become increasingly dominant in determining circuit performance. It was noted in [Bak90] that t_{skew} may account for 10% or more of the system cycle time. The objective of this chapter is to minimize t_{skew}, while we address the problem of minimizing $t_{d\_interconnect}$ in Chapter 4.

Given a routing solution for a clock signal net, the clock skew is defined to be the maximum difference among the delays from the *clock entry point* (CEP) to synchronizing elements in the net. The notion of clock skew is well defined only in the context of a method for evaluating signal delays. The delay from the source to any terminal depends on the wirelength in the source-terminal path, the RC constants of the wire segments in the routing, and the underlying connection topology of the clock tree.[1] Using equations such as those of Rubinstein et al. [RPH83], one can achieve tight upper and lower bounds on delay in a distributed RC tree model of the clock net. However, in practice it is appropriate to apply one of two simpler RC delay approximations, either the linear model or the Elmore

[1]The global routing phase of layout will typically consider the clock and power/ground nets for preferential assignment to (dedicated) routing layers. We assume that the interconnect delay parameters are the same on all metal routing layers, and we ignore via resistances. Thus wirelength becomes a valid measure of the RC parameters of interconnections.

model, both of which are easier to compute and optimize during clock tree design. The following treatment of these models is due to [BK92a] [BK92b].

3.2.1 Linear Delay

The linear delay t_{LD} along a path in a routing tree is proportional to the length of the path and is independent of the rest of the connection topology. Thus, the delay between two terminals u and w in a net is given by:

$$t_{LD}(u,w) = \sum_{e \in path(u,w)} |e|$$

While less accurate than the distributed RC tree delay formulas of Rubinstein et al [RPH83], the linear delay model has been effectively used in clock tree synthesis [KCR91] [RS89]. In general, use of the linear approximation is reasonable with older application-specific IC (ASIC) technologies, which have larger mask geometries and slower packages. Tsay [Tsa91] notes that the linear delay model is also appropriate for emerging optical and wave interconnect technologies. In addition, we observe that linear delay applies to hybrid multi-chip module packages, e.g., with thick-film substrate interconnects that have relatively large geometries [Sha91].

3.2.2 Elmore Delay

With smaller device dimensions and higher ASIC system speeds, a distributed RC tree model for signal delay in clock nets is often required to derive accurate timing information. Typically, we use the first-order moment of the impulse response, also known as the Elmore delay [CK90] [Tsa91]. The Elmore delay model is developed as follows. Let ρ and κ respectively denote the resistance

and capacitance per unit length of interconnect, so that the resistance ρ_e and capacitance κ_e of an edge e are given by $\rho \cdot |e|$ and $\kappa \cdot |e|$, respectively. For each terminal t in the tree T, there is a loading capacitance κ_t which is the input capacitance of the functional unit driven by t.

We let T_v denote the subtree of T rooted at v, and let κ_v denote the node capacitance of v.[2] The *tree capacitance* of T_v is denoted by $\hat{\kappa}_v$ and equals the sum of capacitances in T_v. According to [Elm48] [RPH83] [Sak83], the Elmore delay $t_{ED}(s,t)$ can be calculated by the following formula (see [Tsa91] for a discussion of underlying circuit models):

$$t_{ED}(s,t) = \sum_{e_v \in path(s,t)} \rho_{e_v}(\frac{\kappa_{e_v}}{2} + \hat{\kappa}_v)$$

Elmore delay is additive: if v is a vertex on the u-w path, then $t_{ED}(u,w) = t_{ED}(u,v) + t_{ED}(v,w)$.

3.2.3 Problem Formulation

The discussion in this chapter will assume the linear delay model, so that clock skew is given by the maximum difference in wirelength from the CEP to synchronizing elements in the clock signal net; this allows geometric formulations compatible with the cell-based design regime. However, it should be noted that our algorithm may be easily extended to incorporate the Elmore delay model simply by a more judicious selection of "balance points", as will be discussed below. We represent a clock routing solution by a rooted (Steiner) tree whose root is the CEP and whose leaves are synchronizing elements in the clock signal net. Recall that the cost of an edge in the tree is the Manhattan distance between the

[2] We assume that $\kappa_v = 0$ for all internal nodes.

two endpoints of the edge, and that the tree cost is the sum of all edge costs in the tree.

Definition: The *pathlength skew* of a tree is the maximum difference of the pathlengths in the tree from the root to any two leaves.

A tree is called a *perfect pathlength-balanced tree* if its pathlength skew (i.e., clock skew under the linear delay model) is zero. It is not difficult to construct a perfect pathlength-balanced tree if we are allowed to use an arbitrary amount of wire (right side of Figure 3.2). However, such a construction can lead to a routing tree with very high cost, implying a large RC constant that may distort the clock signal due to longer signal rise and fall times. On the other hand, blindly minimizing tree cost could result in a tree with very large pathlength skew (left side of Figure 3.2). Thus, we wish to construct a tree whose pathlength skew is as small as possible, without incurring a large tree cost penalty (Figure 3.3).

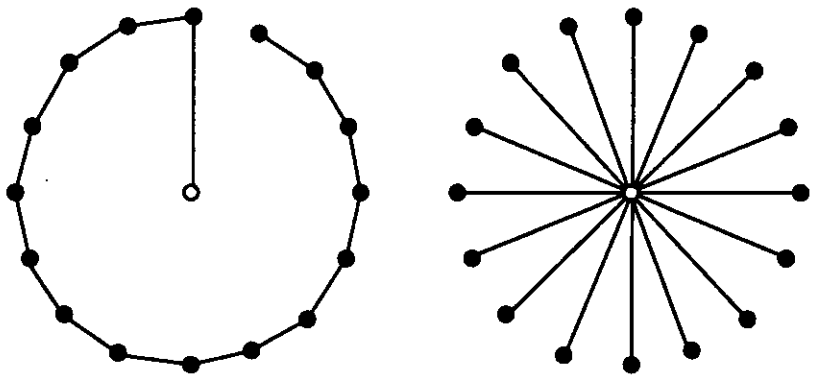


Figure 3.2: Neither the minimum spanning tree (left), nor the shortest paths tree (right) are necessarily good clock trees.

With this in mind, we formulate the clock routing problem as follows:

The Pathlength-Balanced Tree (PBT) Problem: Given a net N and a real parameter ψ, find a minimum-cost tree with pathlength skew $\leq \psi$.

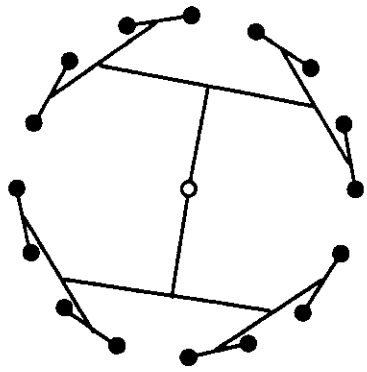


Figure 3.3: A pathlength-balanced tree, having zero skew and reasonably low tree cost.

The following result is immediate:

Theorem 3.1 *The PBT problem is NP-hard.*

Proof: Set $\psi = \infty$ so that the PBT problem "simplifies" to the minimum Steiner tree problem, which is known to be NP-complete even in the Manhattan plane [GJ77]. □

Our objective is to give an efficient heuristic for the PBT problem. In particular, for cell-based design methodologies, we wish to construct a clock tree with pathlength skew as small as possible, and cost as close as possible to that of the optimal Steiner tree. Note that a zero skew tree can be trivially achieved by routing $n = |N|$ separate wires of constant length from the clock source to all of the target terminals (right side of Figure 3.2), but this can result in tree cost arbitrarily higher than the Steiner tree cost. Ideally, we would like to obtain a routing solution in the $L \times L$ grid with cost $O(L \cdot \sqrt{n})$ since the optimal Steiner tree also has cost $O(L \cdot \sqrt{n})$ in the average case [Ste88] (recall the discussion of subadditive functionals in Chapter 2). In what follows, we will assume that the circuit layout is in the $L \times L$ grid.

3.3 A Pathlength-Balanced Tree Algorithm for Cell-Based Design

In developing our pathlength-balanced tree algorithm for cell-based layouts, we first introduce the notion of a geometric matching:

Definition: Given a net of $2n$ terminals, a *geometric matching* consists of n line segments between the terminals, with no two of the n segments sharing an endpoint.

Each line segment in the matching defines a *matching edge*. The cost of a geometric matching is the sum of the costs of its matching edges. A geometric matching over a net is optimal if its cost is minimum among all possible matchings. An example of an optimal geometric matching over four terminals is shown in Figure 3.4.

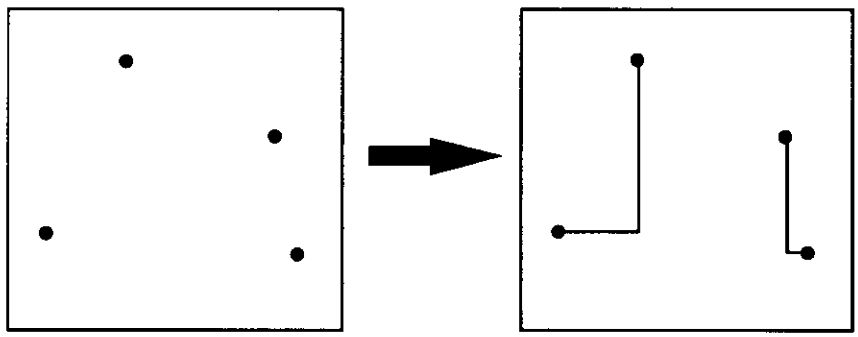


Figure 3.4: An optimal geometric matching over four terminals.

To construct a tree by iterative matching, we begin with a forest of n isolated terminals (for convenience, assume that n is a power of 2), each of which is considered to be a tree with CEP equal to the location of the terminal itself. The optimal geometric matching on these n CEPs yields $\frac{n}{2}$ segments, each of which

defines a subtree with two nodes. The optimal CEP for each subtree of two nodes is the midpoint of the corresponding segment, i.e., so that the clock signal will have zero skew between the segment endpoints.

In general, the matching operation will pair up the CEPs (roots) of all trees in the current forest. At each level, we choose the root of each new merged tree to be the *balance point* which minimizes pathlength skew to the leaves of its two subtrees. The balance point is the point p along the "straight" line connecting the roots of the two subtrees, such that the maximum difference in pathlengths from p to any two leaves in the combined tree is minimized. Computing the balance point requires constant time if we know the minimum and maximum root-leaf pathlengths in each of the two subtrees, and these values can be maintained incrementally using constant time per each node added to the tree.

At each level of the recursion, we only match half as many nodes as in the previous level. Thus, after $\lceil \log n \rceil$ matching iterations, we obtain the complete tree topology. In practice, we actually compute optimal maximum-cardinality matchings, i.e., if there are $2m+1$ nodes, we find the optimal m-segment matching and match $m+1$ CEPs at the next level. We call this algorithm CLOCK1 (see Figure 3.5 for an example of its execution), and use T_{CLOCK1} to denote its output tree. Figure 3.6 gives a formal description of CLOCK1.

The following two results show that the cost of T_{CLOCK1} is indeed reasonable: (i) $cost(T_{CLOCK1})$ grows at the same asymptotic rate as the worst-case optimal Steiner tree cost, and (ii) $cost(T_{CLOCK1})$ is on average within a constant factor of the optimal Steiner tree cost.

Theorem 3.2 *For a net of n terminals arbitrarily distributed in the $L \times L$ square,* $cost(T_{CLOCK1}) = O(L \cdot \sqrt{n})$.

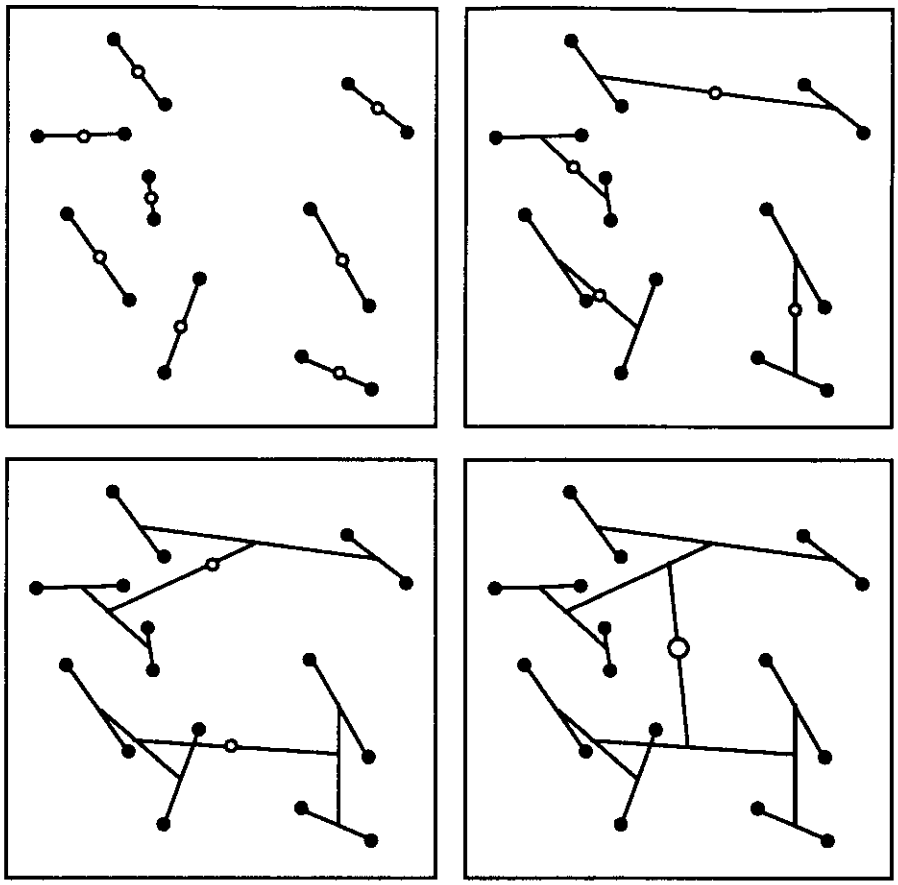


Figure 3.5: Example of algorithm CLOCK1 running on a set of 16 terminals. Solid dots denote terminals, and hollow dots represent the balance points of the corresponding edges. At each level a geometric matching is computed on the balance points of the previous level. Note that although edges are depicted as straight lines, they are actually routed rectilinearly.

Proof: For n terminals in the $L \times L$ grid, the worst-case cost of an optimal matching is $O(L \cdot \sqrt{n})$ [SRP83]. 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

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

□

CLOCK1: A Pathlength-Balanced Tree Algorithm for Cell-Based Designs

Input: A net N

Output: A pathlength-balanced tree T_{CLOCK1} with root CEP

$T = \emptyset$
$P = N$
While $|P| > 1$
  M = the edges of the 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
      containing p_1 and p_2, such that p minimizes skew
      of the tree $T_1 \cup T_2 \cup \{(p, p_1), (p, p_2)\}$ rooted at p
    $P' = P' \cup \{p\}$
    $T = T \cup \{(p, p_1), (p, p_2)\}$
  $P = P'$ plus a possible unmatched node if $|P|$ is odd
CEP = root of T = single remaining point in P
Output $T_{CLOCK1} = T$

Figure 3.6: The matching-based pathlength-balanced tree algorithm for cell-based design.

This is of the same order as the maximum possible cost of an optimal Steiner tree on n arbitrarily located terminals [Ste88]. Note that Theorem 3.2 does not directly relate the tree cost of our construction to the cost of the optimal Steiner tree; this is addressed by the following result.

Theorem 3.3 *For nets 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 minimum Steiner tree cost for n terminals randomly chosen from a

uniform distribution in the $L \times L$ Manhattan grid grows as $\beta \cdot L \cdot \sqrt{n}$ for some constant β [Ste88]. The theorem follows from the $O(L \cdot \sqrt{n})$ worst-case bound on the minimum-cost matching at any level of the construction [SRP83]. □

The balancing operation to determine the CEP of a merged tree is necessary because the root-leaf pathlength varies between subtrees at a given stage of the construction. In general, when we merge subtrees T_1 and T_2 into a higher-level subtree T, the optimal entry point of T will not necessarily be equidistant from the entry points of T_1 and T_2 (this can be observed in the example of Figure 3.5). Intuitively, balancing entails "sliding" the CEP along the "bar of the H". However, it might not always be possible to obtain perfectly balanced pathlengths in this manner (see Figure 3.7).

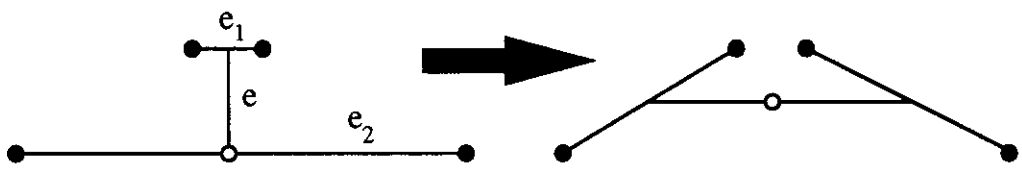


Figure 3.7: Example of flipping an H as to minimize pathlength skew: the tree on the left has *no* zero-skew balance point along the middle segment of the "H", while the tree on the right does.

We therefore use a further optimization, which we call *H-flipping*: for each edge e added to the layout which matches CEPs on edges e_1 and e_2, replace the "H" formed by the three edges e, e_1, and e_2 by the "H" over the same four terminals which (i) minimizes pathlength skew, and (ii) to break ties, minimizes tree cost. We now prove that for four terminals it is always possible find an "H" orientation which achieves zero pathlength skew, and we also bound the increase in tree cost caused by H-flipping for nets of size 4. As discussed below, extensive empirical tests confirm that even for very large nets, the H-flipping refinement

almost always yields perfect pathlength-balanced trees with essentially no cost increase.

If a net is of size two, CLOCK1 selects the midpoint of the segment connecting the two terminals as the balance point and this clearly yields a perfect pathlength-balanced tree. Now we show that for nets of size 4, CLOCK1 with the H-flipping refinement also yields perfect pathlength-balanced trees (a net of size three can be treated as a net of size 4 in which two terminals coincide).

Let a, b, c and d be the terminals of a net of size 4. Without loss of generality, assume that ab and cd are the edges in an optimal matching and $ab \geq cd$. (For convenience, we use ab to denote both the segment connecting terminals a and b and also the length of the segment ab.) Let m_1 and m_2 be the midpoints of ab and cd, respectively. CLOCK1 chooses m_1 to be the root of the subtree for a and b, and m_2 to be the root of the subtree for c and d. Then, CLOCK1 attempts to find a balance point p on segment m_1m_2 such that

$$\frac{ab}{2} + pm_1 = \frac{cd}{2} + pm_2 \tag{3.1}$$

It is easy to see that if $m_1m_2 \geq \frac{ab-cd}{2}$, we can always choose p satisfying Equation 3.1. In this case, the pathlengths from p to all four terminals are the same, so that we have a perfect pathlength-balanced tree. However, if $m_1m_2 < \frac{ab-cd}{2}$, we carry out H-flipping as described earlier, replacing ab and cd by ad and bc. We choose the midpoint n_1 on bc to be the root of the subtree for b and c, and the midpoint n_2 on ad to be the root of the subtree for a and d. We then choose p' on n_1n_2 such that

$$\frac{ad}{2} + p'n_1 = \frac{bc}{2} + p'n_2 \tag{3.2}$$

According to the following lemma, we are guaranteed to find p' on n_1n_2 satisfying Equation 3.2.

Lemma 3.4 *If* $m_1m_2 < \frac{ab-cd}{2}$ *then* $n_1n_2 \geq \frac{bc-ad}{2}$.

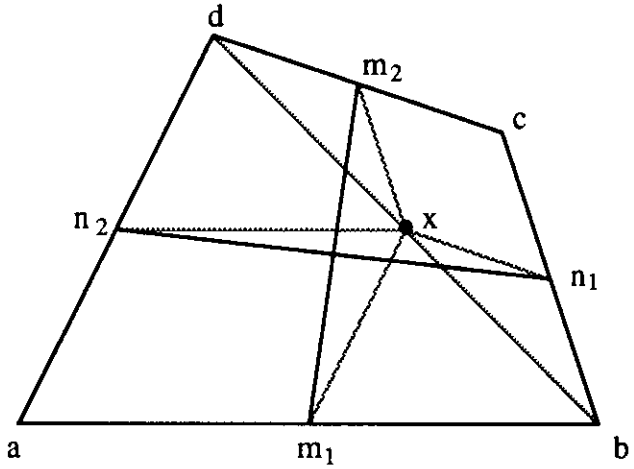


Figure 3.8: Using similar triangles to analyze H-flipping skew.

Proof: If we have both $m_1m_2 < \frac{ab-cd}{2}$ and $n_1n_2 < \frac{bc-ad}{2}$ then

$$m_1m_2 + n_1n_2 < \frac{ab-cd}{2} + \frac{bc-ad}{2}$$

therefore

$$\frac{ab+bc}{2} > m_1m_2 + n_1n_2 + \frac{cd+ad}{2} \tag{3.3}$$

Let x be the midpoint of bd. Using similar triangles and the triangle inequality (see Figure 3.8), we obtain

$$\frac{ab}{2} = xn_2 \leq n_1n_2 + xn_1 = n_1n_2 + \frac{cd}{2}$$

and

$$\frac{bc}{2} = xm_2 \leq m_1m_2 + xm_1 = m_1m_2 + \frac{ad}{2}$$

thus

$$\frac{ab+bc}{2} \leq m_1m_2 + n_1n_2 + \frac{cd+ad}{2}$$

which is a contradiction to Equation 3.3. Therefore if $m_1m_2 < \frac{ab-cd}{2}$ we must have $n_1n_2 \geq \frac{bc-ad}{2}$. □

Lemma 3.4 implies that we can always choose the balance point p' on n_1n_2 after H-flipping. Therefore, CLOCK1 always constructs a perfect pathlength-balanced tree for a net of up to 4 terminals. The following lemma shows that when we replace ab and cd by ad and bc during H-flipping, the cost increase is bounded by a constant factor.

Lemma 3.5 *If* $m_1m_2 < \frac{ab-cd}{2}$ *then* $bc + ad \leq 3(ab + cd)$.

Proof: Let x be the midpoint of bd. Again applying similar triangles and the triangle inequality (see Figure 3.9), we obtain

$$\frac{bc}{2} = xm_2 \leq xd + dm_2 = xd + \frac{cd}{2}$$

and

$$\frac{ad}{2} = xm_1 \leq xb + bm_1 = xb + \frac{ab}{2}$$

thus

$$\frac{bc+ad}{2} \leq bd + \frac{ab+cd}{2} \tag{3.4}$$

Let y be the intersection of bd and m_1m_2, so that

$$by \leq m_1y + m_1b = m_1y + \frac{ab}{2}$$

$$dy \leq m_2 y + m_2 d = m_2 y + \frac{cd}{2}$$

$$bd \leq m_1 m_2 + \frac{ab+cd}{2} < \frac{ab-cd}{2} + \frac{ab+cd}{2} = ab \tag{3.5}$$

Thus, from Equations 3.4 and 3.5 we have

$$\frac{bc+ad}{2} \leq ab + \frac{ab+cd}{2} \leq \frac{3(ab+cd)}{2}$$

or $bc + ad \leq 3(ab + cd)$. □

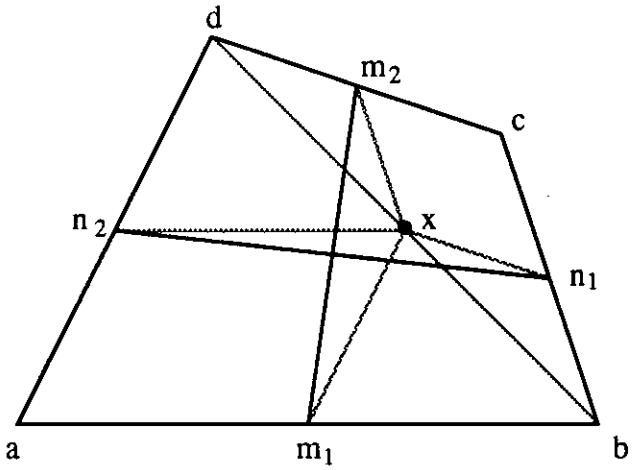


Figure 3.9: Using similar triangles to analyze H-flipping cost.

Together, the two previous lemmas imply the following:

Theorem 3.6 *It is always possible find an "H" orientation over four terminals which achieves zero pathlength skew, with at most a constant factor cost increase.*

Since our method is based on geometric matching, its time complexity depends on that of the matching subroutine. A well-known algorithm for general weighted matching requires time $O(n^3)$ [Gab76] [Law76]. By taking advantage of the planar

geometry, the algorithmic complexity can be reduced to $O(n^{2.5} \log n)$ [Vai88]. However, even this lower-complexity method may be inefficient for large problem instances.

In order to solve problems of practical interest, and since there is no clear relationship between the optimality of the matching and the magnitude of the pathlength skew of the resulting tree, we may choose to speed up the implementation by using efficient matching heuristics [BP83] [SR83] [Sup90]. Although most of these methods were designed for the Euclidean plane, they also perform well in the Manhattan metric, especially if their output is further improved by uncrossing pairs of intersecting edges in the heuristic matching (in any metric, this reduces the matching cost by virtue of the triangle inequality; to this end, note that k intersections of n line segments may be found efficiently in time $O(n \log n + k)$ [CE92]).

We shall later discuss the empirical behavior of CLOCK1 based on three matching methods which require time $O(n)$, $O(n \log n)$ and $O(n \log^2 n)$ respectively. Each of these three matching heuristics yields very good solutions. When performance is critical, an optimal geometric matching algorithm might give an improvement over our current implementations, but will require correspondingly greater computational resources.

The basic approach of CLOCK1 thus consists of $\lceil \log n \rceil$ applications of the matching algorithm. H-flipping requires constant time per node, and therefore does not add to the asymptotic time complexity. If the underlying matching algorithm runs within monotonically non-decreasing time $S(n) = \Omega(n)$, we may write $S(n) = n \cdot T(n)$ where $T(n) = \frac{S(n)}{n}$ is monotonically non-decreasing, and hence the total time required by CLOCK1 is

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

i.e., the time complexity of CLOCK1 is asymptotically equal to the time complexity of the underlying matching algorithm.

3.4 A Pathlength-Balanced Tree Algorithm for General Cell Design

The same ideas of bottom-up iterative matching which we developed in the preceding section may be generalized to pathlength-balanced tree construction where distances are specified as an arbitrary weighted graph. This corresponds to the practical application of clock routing in general cell designs, where a circuit is partitioned into a set of arbitrarily-sized rectangular cells (also referred to as blocks). Blocks may be of widely varying sizes, and are not necessarily placed in any regular arrangement. The routing is carried out in the channels between blocks, which may be represented using a channel intersection graph [DAK85]. For this design style, the approximation of routing cost by geometric distance, which we used for cell-based design in the previous section, does not apply since Manhattan distance is no longer a good approximation of the routing cost between two terminals.

Our goal remains to construct a pathlength-balanced tree which has both pathlength skew and cost as small as possible, except that routing of tree edges is now restricted to lie within prescribed routing channels. Recall that given a graph G with a non-negative cost function on the edges, we let $minpath_G(x, y)$ denote the minimum cost path between nodes x and y, and use $dist_G(x, y)$ to denote the cost of $minpath_G(x, y)$. The notion of a matching may be extended to arbitrary weighted graphs as follows:

Definition: Given a graph $G = (V, E)$ with a non-negative cost function on the edges, a *generalized matching* M in G is a set of shortest paths connecting m mutually disjoint node pairs, i.e., $M = \{minpath_G(x_1, y_1), minpath_G(x_2, y_2), \ldots, minpath_G(x_m, y_m)\}$, where the x_i's and y_i's are all distinct.

A generalized matching on a set of nodes $N \subseteq V$ in G is *complete* if $m = \lfloor \frac{|N|}{2} \rfloor$. 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 $N \subseteq V$ is one with least cost. We can show the following properties of optimal complete generalized matchings:

Lemma 3.7 *Each edge of G belongs to at most one shortest path in an optimal complete generalized matching on $N \subseteq V$ in G.*

Proof: Let M be an optimal complete generalized matching on N. Suppose that 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 3.10.

It is not difficult to see that

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

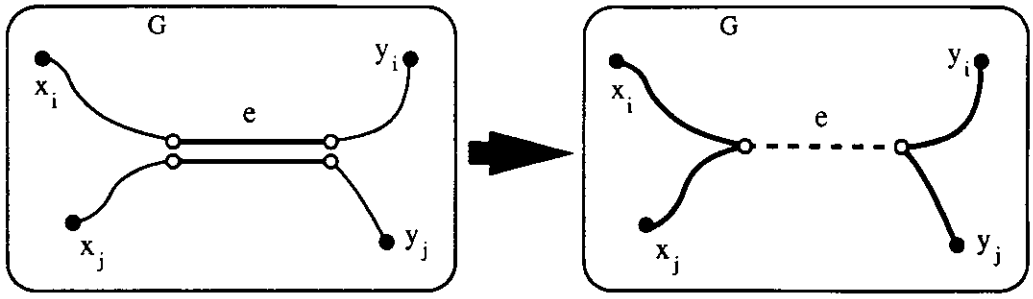


Figure 3.10: Each edge belongs to at most one shortest path in an optimal complete generalized matching.

Therefore, we would obtain a complete generalized matching on N with smaller cost by replacing $minpath_G(x_i, y_i)$ and $minpath_G(x_j, y_j)$ in M by $minpath_G(x_i, x_j)$ and $minpath_G(y_i, y_j)$, a contradiction. □

Henceforth, we will assume that there are b blocks in the design. G is the underlying channel intersection graph and we assume that the n terminals are embedded on edges of G.

Lemma 3.8 *The routing cost between any two terminals in G is no more than $2L$.*

Proof: Let x and y be two terminals 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'. Then, $cost(P_1) + cost(P_2) = 4L$. Since at least one of w or w' can be reached from *both* x and y with cost at most $2L$, the shortest path between x and y has cost no more than $2L$. □

It is clear from Lemma 3.8 that an optimal complete generalized matching on the terminals in G has cost no more than $2L \cdot \lfloor \frac{n}{2} \rfloor \leq n \cdot L$.

As in the previous section, our basic strategy is to construct a pathlength-balanced tree by computing a sequence of generalized matchings on the terminals. We begin with a forest of n isolated terminals in G (again for convenience, we assume that n is a power of 2), each of which is a degenerate tree with CEP being the terminal itself. The optimal complete generalized matching on these n terminals yields $\frac{n}{2}$ paths, each of which defines a subtree. The optimal CEP into each subtree is the midpoint of the corresponding path, so that the pathlength skew from the CEP to the two terminals is zero. At each level, we compute an optimal generalized matching on the set of CEPs (roots) of all subtrees in the current forest and merge each pair of subtrees into a larger subtree. As before, the root of the resulting tree is chosen to be the balance point on the path connecting the two subtrees such that the pathlength skew in the resulting tree is minimized (see Figure 3.11).

Note that at each iteration, we need only match half as many nodes as in the previous iteration. Thus, in $\lceil \log n \rceil$ matching iterations, we obtain a complete tree topology. If n is not a power of 2, then as noted in the discussion of CLOCK1, there will be an odd number $2m + 1$ of nodes to match at some level. For such cases, we compute an optimal maximum-cardinality generalized matching on $2m$ nodes, and then match $m + 1$ nodes at the next level. Figure 3.12 gives a formal description of our pathlength-balanced tree algorithm CLOCK2.

The worst-case cost of T_{CLOCK2} can be bounded as follows:

Theorem 3.9 *Given b blocks and n terminals in the $L \times L$ grid, cost$(T_{CLOCK2}) \leq 2nL$.*

Proof: By Lemma 3.8, the cost of a generalized matching on n terminals is

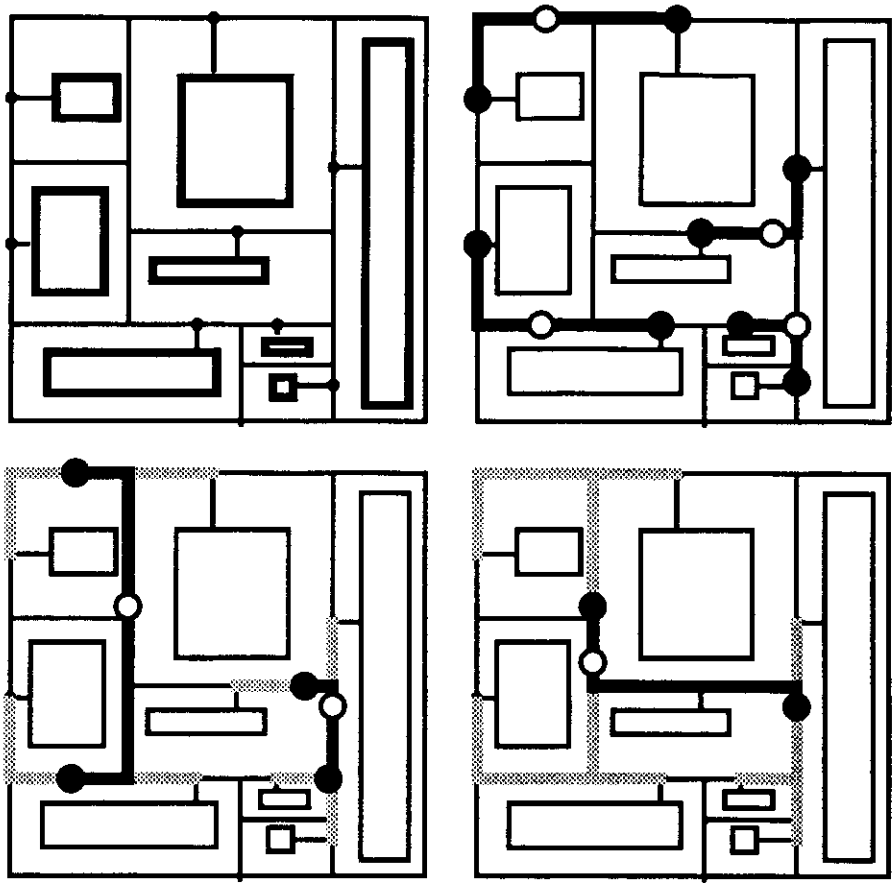


Figure 3.11: Example of the CLOCK2 algorithm running on a random module placement with an 8-terminal net. Solid dots are roots of subtrees in the previous level; hollow dots are roots (CEPs) of new subtrees computed at the current level. At each level an optimal complete generalized matching is computed on the solid points. For clarity, only the newly added wires are highlighted at each level.

bounded by nL. After each iteration, the number of nodes to be matched is reduced by half. Therefore, $cost(T_{CLOCK2}) \leq nL + \frac{nL}{2} + \frac{nL}{4} + \ldots \leq 2nL$. □

In order to compute an optimal generalized matching on a set of nodes N in G, we construct a weighted complete graph G' on N such that $weight(x, y) = dist_G(x, y)$ for each pair of nodes x and y in N. This can be accomplished by applying an $O(|E| \cdot |V| + |V|^2)$ implementation of Floyd's all-pairs shortest paths

CLOCK2: A Pathlength-Balanced Tree Algorithm for General Cell Design

Input: A set of terminals N embedded in a CIG G

Output: A pathlength-balanced tree T_{CLOCK2} with root CEP

$T = \emptyset$
$P = N$
While $|P| > 1$
  M = opt 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 an unmatched node if $|P|$ is odd
CEP = Root of T = single remaining point in P
Output $T_{CLOCK2} = T$

Figure 3.12: The matching-based pathlength-balanced tree algorithm CLOCK2 for general cell design.

algorithm [Sed88] to the graph $G = (V, E)$. Note that G is a planar graph and therefore $|E| = O(|V|)$. Since a channel intersection graph induced by b blocks has $|V| = O(b+n)$, and typically $b > n$, the overall time complexity for this step is $O(b^2)$. We may then apply an $O(n^3)$ algorithm for computing an optimal complete matching in general graphs [Law76]. However, this complexity will result in long runtimes for large problem instances. Therefore, in order to achieve an efficient implementation, we use the greedy matching heuristic [SS90a]. Such a heuristic matching may be improved by removing overlapping edges of shortest paths, as described in the proof of Lemma 3.7, so that no edge is used in more than one shortest path. The time complexity of each iteration of CLOCK2 is dominated

by the $O(b^2)$ all-pairs shortest paths computation, which we invoke $\lceil \log n \rceil$ times, so that the overall time complexity of CLOCK2 is $O(b^2 \cdot \log n)$. This complexity is reasonable, since the number of blocks is typically not large.

3.5 Experimental Results

Both CLOCK1 and CLOCK2 were implemented in ANSI C for the Sun-4, Macintosh and IBM 3090 environments. This section summarizes the empirical results.

3.5.1 Empirical Data for Cell-Based Design

We have implemented three basic variants of CLOCK1, corresponding to different matching subroutines. The first variant (SP) uses the linear-time space partitioning heuristic of [SR83] to compute an approximate matching; the second variant (GR) uses an $O(n \log^2 n)$ greedy matching heuristic [Sup90]; and the third variant (SFC) uses an $O(n \log n)$ spacefilling curve-based method [BP83]. We have further tested these three variants by running each both with and without two refinements: (i) removing all edge crossings in the heuristic matching, and (ii) performing "H-flipping" as necessary. Either of these optimizations can be independently used with any of the three basic variants, yielding a total of twelve distinct versions of the CLOCK1 algorithm. These variants are denoted and summarized as follows:

- **SP** - Use the space-partitioning matching heuristic of [SR83], which induces a matching through recursive bisection of the region (rather than bisection of the set of terminal locations).

- **GR** - Use a greedy matching heuristic, which always adds the shortest edge between unmatched terminals [Sup90].

- **SFC** - Use a space-filling curve to map the plane to a circle, then choose the better of the two embedded matchings (i.e., either all odd edges or all even edges in the induced Hamiltonian cycle through the terminal locations) [BP83].

- **SP+E, GR+E, SFC+E** - Same as SP, GR and SFC, respectively, except that the heuristic matching cost is further improved by edge-uncrossing.

- **SP+H, GR+H, SFC+H** - Same as SP, GR and SFC, respectively, except that pathlength skew is further reduced 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 that of Jackson, Srinivasan and Kuh [JSK90].

The algorithms were tested on a large number of random nets of up to 1024 terminals, generated from a uniform distribution in the 1000 x 1000 grid. Results for a sample run with 50 random nets of each cardinality are summarized here: Tables 3.1 and 3.2 compare the average tree costs and Tables 3.3 and 3.4 compare the average pathlength skews for all heuristics. The data in the tables are given in grid units.

The computational results indicate that both optimizations (edge-uncrossing and H-flipping) significantly improve both pathlength skew and tree cost. When

$\|N\|$	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 3.1: Tree cost averages, in grid units, for the various heuristics.

$\|N\|$	SP+H	GR+H	SFC+H	SP+E+H	GR+E+H	SFC+E+H	Meta
4	1125	1125	1125	1125	1125	1125	1125
8	2027	2028	1994	1971	1979	1980	1960
16	3502	3416	3428	3333	3322	3329	3268
32	5860	5628	5577	5329	5273	5304	5151
64	9226	8794	8748	8076	7982	8047	7844
128	13997	3315	13159	11871	11697	11914	11566
256	21307	19611	19713	17457	16955	17629	16919
512	31646	29175	28688	25188	24465	25483	24480
1024	46417	42110	41540	36276	34965	36814	34992

Table 3.2: Tree cost averages, in grid units, for the various heuristics (continued).

the refinements are combined, average pathlength skew essentially vanishes completely, and the tree cost of several of the variants is noticeably superior to that of MMM, which may yield trees with large pathlength skews even for small examples (Figure 3.13). The best variant appears to be GR+E+H, which is based on the greedy matching heuristic together with edge-uncrossing and H-flipping. This is noteworthy because the greedy method is asymptotically as good as the optimal

$\|N\|$	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 3.3: Pathlength skew averages, in grid units, for the various heuristics.

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

Table 3.4: Pathlength skew averages, in grid units, for the various heuristics (continued).

matching [SS90a]. Tables 3.5 and 3.6 highlight the contrast between GR+E+H and MMM, showing minimum, maximum and average values for both tree cost and pathlength skew. Figures 3.14 and 3.15 depicts these same comparisons graphically.

As noted in Chapter 2, any set of approximation heuristics induces a *meta-heuristic* which returns the best solution found by any heuristic in the set; we also

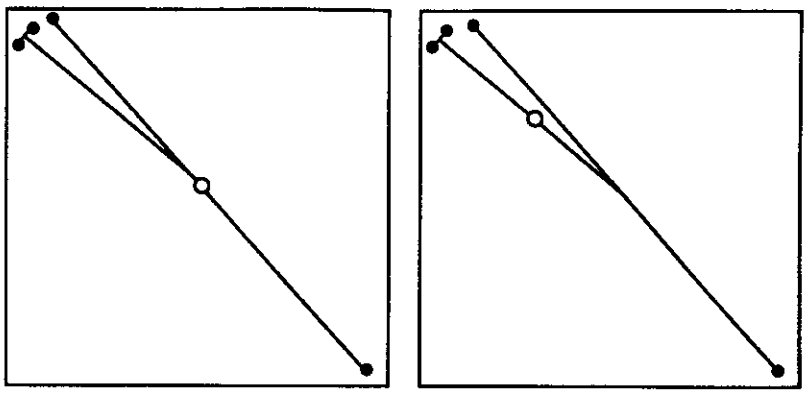


Figure 3.13: MMM may yield large skew (right), compared with the optimal tree zero skew tree (left).

	MMM			GR+E+H		
$\|N\|$	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 3.5: Minimum, average and maximum pathlength skew values, in grid units, for GR+E+H and MMM.

implemented this (denoted as "Meta"), which returns the minimum-skew result from all of the other variants. Interestingly, in our experience Meta *always* returns a perfect pathlength-balanced tree, i.e., for each problem instance, at least one of the heuristic variants will yield a zero pathlength skew solution. This is very useful, especially when the heuristics are of similar complexity. For example, we can solve the Primary1 benchmark using all twelve methods in a few minutes on

	MMM			GR+E+H		
$\|N\|$	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 3.6: Minimum, average and maximum tree costs, in grid units, for GR+E+H and MMM.

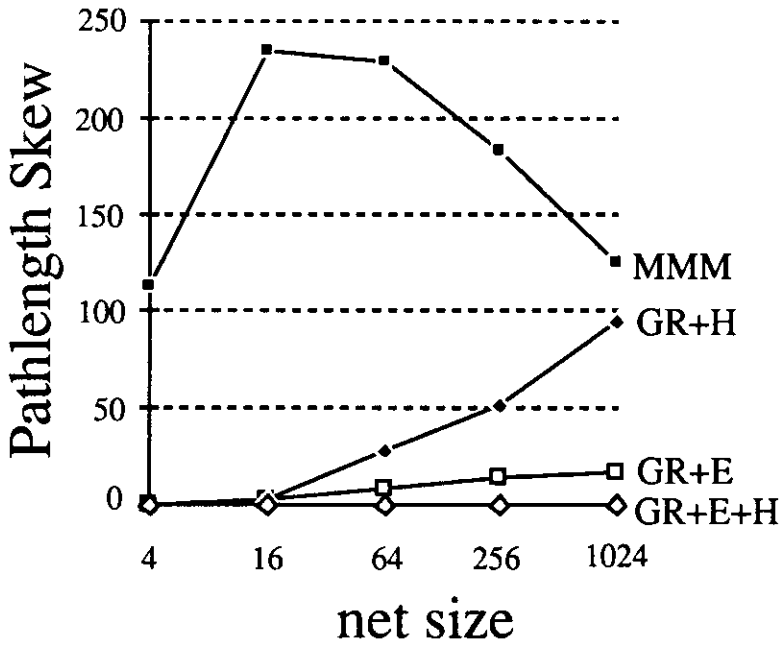


Figure 3.14: Overall pathlength skew comparisons between CLOCK1 (GR+E+H) and MMM.

a Sun-4/60 workstation.

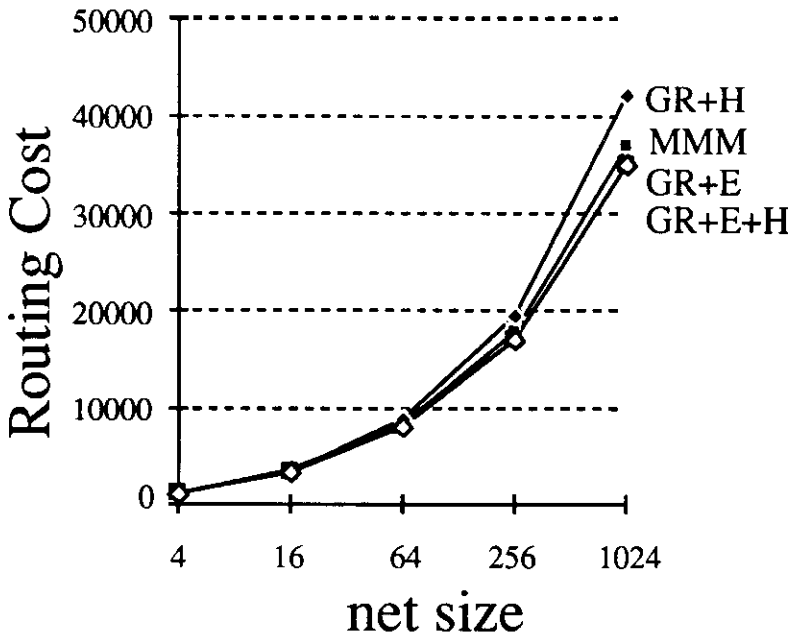


Figure 3.15: Overall tree cost comparisons between CLOCK1 (GR+E+H) and MMM.

Figures 3.16 and 3.17 illustrate the output of variant GR+E+H on the Primary1 and Primary2 benchmarks, using the same placement solutions as in [JSK90]; note that although edges are depicted as straight lines in these diagrams, they are actually routed rectilinearly. Table 3.7 compares the results of GR+E+H and the results of [JSK90] which were provided by the authors [Sri91]: GR+E+H completely eliminates pathlength skew while obtaining 5% - 7% reduction in tree cost. To confirm the correlation between the linear delay model and actual delay, we ran HSPICE simulations on the Primary1 and Primary2 clock trees (using MOSIS 2.0μ CMOS technology and 0.3pF gate loading capacitance): the actual skews of our clock trees for Primary1 and Primary2 were 181ps and 741ps, respectively. Note that this clock skew was obtained simply by balancing CEP-leaf pathlengths; as discussed earlier, more sophisticated delay models can

allow a better choice of balance points in the matching-based construction, and indeed recent work of [Tsa91] chooses balance points according to the Elmore model to achieve an "exact zero skew" clock routing.

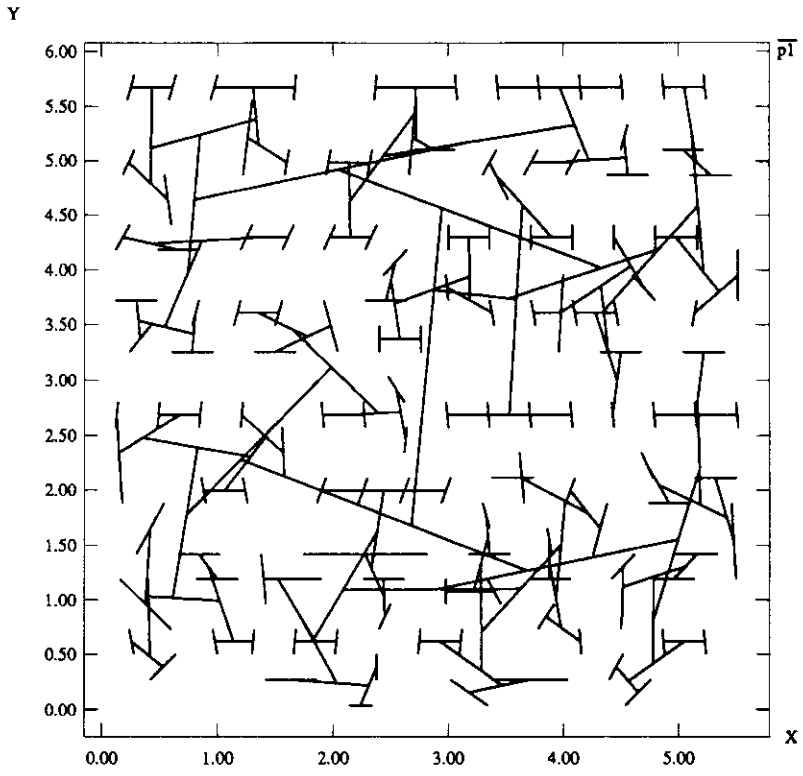


Figure 3.16: Output of variant GR+E+H on the Primary1 benchmark layout.

3.5.2 Empirical Data for General Cell Design

We have tested CLOCK2 on two sets of test cases. One set of examples contains nets of sizes 4, 8, and 16 in layouts of 16 blocks, and the other set contains nets of sizes 4, 8, and 16 in layouts of 32 blocks. Block sizes and layouts were assigned randomly in the grid, by creating a fixed number of non-overlapping blocks, with

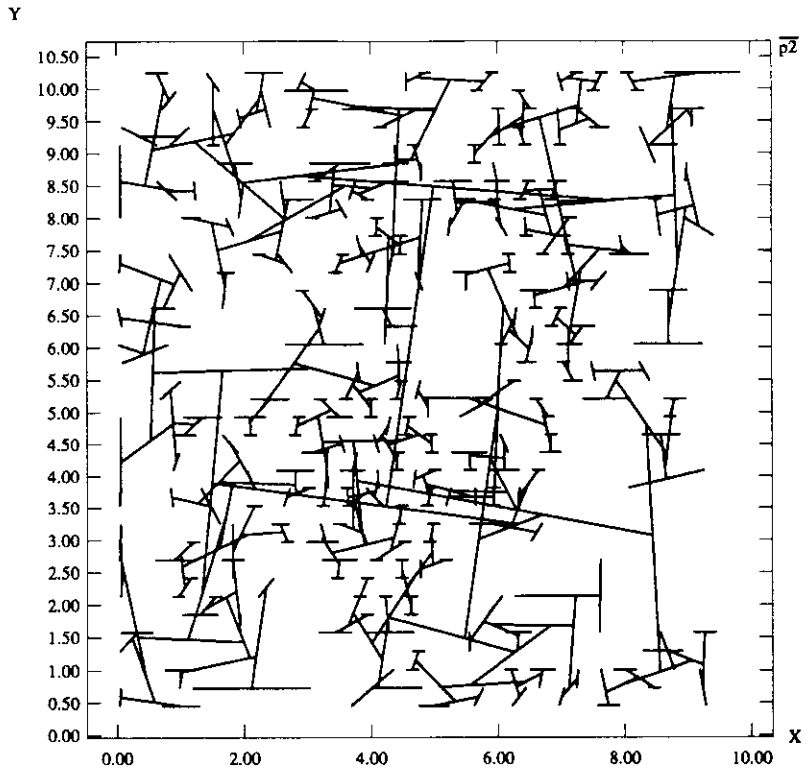


Figure 3.17: Output of variant GR+E+H on the Primary2 benchmark layout.

	Pathlength skew (STD) MMM	Tree cost MMM	Pathlength skew (STD) GR+E+H	Tree cost GR+E+H	Reduction skew (STD)	Reduction cost (%)
P1	0.29	161.7	0.00	153.9	0.29	4.8
P2	0.74	406.3	0.00	376.7	0.74	7.3

Table 3.7: Comparisons of GR+E+H and MMM on Primary1 and Primary2 benchmark layouts ("skew (STD)" denotes the standard deviation of the root-leaf pathlengths).

length, width, and lower-left coordinates all chosen from uniform distributions on the interval $[1, L]$ with $L = 1000$.

For each combination of net size and block cardinality, 100 instances were generated randomly, and we compared the pathlength skew and tree cost of T_{CLOCK2} with those produced by the Steiner tree heuristic of Kou, Markowsky and Berman [KMB81] (cf. the discussion in Section 4.5.1 below). Results are shown in Tables 3.8 and 3.9. Clearly, the average pathlength skew of T_{CLOCK2} is very close to zero, and it is never more than 2% of the pathlength skew in the heuristic Steiner tree routing. The increase in tree cost of our routing tree varies from 24% to 77% when compared with the Steiner tree. The data in the tables are given in grid units.

As with the cell-based layout benchmarks, we ran HSPICE simulations on a number of examples (again using MOSIS 2.0μ CMOS technology and 0.3pF gate loading capacitance). The actual skew of our clock tree is consistently much smaller than that of the Steiner tree. For a typical 16-pin net in a 16-block design, the clock skews of our clock tree and the Steiner tree are 18ps and 69ps, respectively.

T_{CLOCK2} may have overlapping edges in a channel because matching paths at different levels may use the same channel. However, by Lemma 3.7, no channel will appear in more than one path in a single matching. Therefore, there are at most $\lceil \log n \rceil$ overlapping edges in each channel. The last column in Table 3.9 shows the average edge density in channels, computed as the average of non-zero local column densities over all columns in all channels.

3.6 Remarks and Extensions

In summary, we have presented a bottom-up approach for constructing pathlength-balanced trees, with applications to clock routing in both cell-based and general

		Pathlength skew		Tree cost	
# modules	$\|N\|$	Steiner	CLOCK2	Steiner	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 3.8: Average tree costs and pathlength skews, in grid units, of CLOCK2 and the Steiner tree heuristic, respectively. For each row, 100 random instances in the 1000 by 1000 grid were generated and routed.

b	$\|N\|$	Pathlength skew	Tree density	Edge density in channels
16	4	0.00	1.26	1.24
16	8	0.02	1.49	1.40
16	16	0.02	1.77	1.63
32	4	0.01	1.24	1.21
32	8	0.01	1.52	1.36
32	16	0.01	1.74	1.48

Table 3.9: Average tree costs and pathlength skews of CLOCK2 output, normalized (per instance) to corresponding heuristic Steiner tree values. For each row, 100 random instances in the 1000 by 1000 grid were generated and routed.

cell designs. Pathlength skew minimization is achieved by constructing the clock tree iteratively through geometric or graph matchings, while carefully balancing the pathlengths from the root to all leaves at each level of the construction. We verified our algorithm on numerous random examples, on industry benchmark circuits, and by timing simulations; the results indicate that our methods yield trees with near-zero average pathlength skew and reasonably low tree cost.

We recommend that our global clock routing be performed before other wiring, following established layout practice. In this way, there are no wire-crossing conflicts, since two layers of metal may be used, one for horizontal wires, and the other for vertical wires. The exact routing of the clock tree topology may be later refined in the detailed routing step.

For cell-based design, we can realize additional tree cost savings by varying the geometrical embedding of individual wires in the layout. 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 3.18), with the extremes corresponding to the two L-shaped wire orientations. Our current implementation sets the balance point of a segment to be its "Euclidean" midpoint, but sometimes this is not necessarily an optimal choice in the pathlength-balanced tree construction. Using a graph-theoretic formulation, we can easily derive a polynomial-time method, based on general graph matching, for finding the optimal set of balance points within these loci.

At each level of our algorithm we may use *lookahead* of one or more levels, or equivalently, when we reach a situation where pathlength skew can not be eliminated even by using an H-flip, we can "go back" one or two levels on the subtrees involved and try different H-flips during previous iterations on those subtrees.[3] In our experience, this strategy easily allows complete elimination of pathlength skew at the current level, and requires only a constant amount of computation per lookahead, provided that the lookahead depth (i.e., number of

[3]Tsay [Tsa91] recently gave an algorithm which uses ideas similar to both [JSK90] and the present work, and incorporates one level of look-ahead to achieve a zero skew tree with respect to the Elmore delay model [Elm48] [RPH83]. In the same spirit as our method, Tsay's method combines pairs of zero skew trees at "tapping points" to yield larger zero skew trees. His tapping points are the roots of the recursively merged subtrees, analogous to our "balance points".

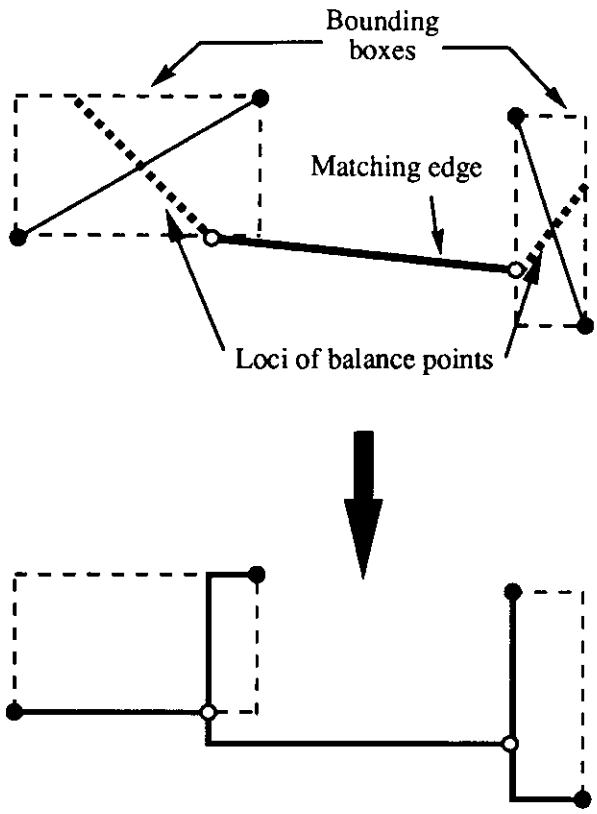


Figure 3.18: Further optimizations are possible by matching over the *loci* of balance point candidates.

levels) is bounded by a constant.[4]

Another important extension lies in the selection of the CEP at each level. Instead of using the linear delay model to select a CEP, we may use a more accurate (but more computationally expensive) RC tree delay model, in order to bias the selection of the CEP so that clock skew is minimized as much as possible. This is a strictly local modification of our method (cf. comments

[4]The recent "Deferred-Merge Embedding" algorithm of [BK92a] [BK92b] utilizes look-ahead to reduce the cost of an initial clock tree computed by any previous method, while maintaining exact zero clock skew. In regimes where the linear delay model applies, this method produces the *optimal* (i.e., minimum-cost) zero-skew clock tree with respect to the prescribed topology and this tree will also enjoy optimal source-terminal delay.

regarding [Tsa91] above) and does not affect the execution of the rest of the algorithm or any of its variants. This extension, which applies to both CLOCK1 and CLOCK2, is particularly useful when varying gate capacitive loadings exist at the terminals of the clock net. Since our algorithm operates in a bottom-up fashion, and since we treat each level independently, our method is able to accommodate variable gate loading very naturally.

Finally, the PBT problem is interesting from a theoretical standpoint: the tradeoff between pathlength balance and tree cost appears important not only for clock skew minimization, but also for a number of applications in areas ranging from computational geometry to network design.

CHAPTER 4

Bounded-Radius Trees

4.1 Introduction

In previous chapters we have proposed algorithms for constructing trees with the objective of minimizing either tree cost or pathlength skew. Here, we examine the problem of constructing an interconnection tree with bounded radius (the radius of a tree is its maximum root-leaf pathlength). The bounded-radius formulation is particularly relevant to circuit routing applications, where signal delay is proportional to the routing tree radius.[1] Interconnection delay contributes up to 70% of the clock cycle in the design of dense, high performance circuits [DNA90] [SS90b] and this motivates the bounded-radius formulation. Thus, performance-driven placement and routing have received increased attention in the past several years [CR91] [DAD84] [HNY87] [JKM87] [JK89] [LD90] [ML89] [OPK90].

This chapter begins by proposing a new method for bounded-radius tree construction. Our method constructs a spanning tree with radius at most $(1+\epsilon)\cdot R$ by using an analog of the classical Prim minimum spanning tree construction [Pri57], where R is the minimum possible tree radius and ϵ is a non-negative user-specified parameter. Such an approach offers a natural, smooth tradeoff between the tree radius (maximum signal delay) and the tree cost. For circuit

[1] See the linear delay model discussion in Section 3.2.1.

layouts, this implies a great deal of algorithmic flexibility, as the parameter ϵ can be varied depending on performance constraints. The method is easy to describe and implement, and empirical performance results are very promising. However, the tree cost using this method can be an unbounded factor worse than optimal.

We therefore also propose a second, provably good method for bounded-radius tree construction which *simultaneously* minimizes both tree cost and tree radius. Given a positive real parameter ϵ and a set of points, this method produces a spanning tree with radius at most $(1+\epsilon) \cdot R$, and with cost at most $(1+\frac{2}{\epsilon})$ times the MST cost. Thus, both the cost and the radius are simultaneously bounded by *constant* factors away from their optimal values. Our method generalizes to Steiner tree formulations in arbitrary weighted graphs, where we achieve a cost bound of $2 \cdot (1+\frac{2}{\epsilon})$ times the optimal *Steiner* tree cost while still observing the $(1+\epsilon) \cdot R$ radius limit.

We then show that geometry helps: in the Manhattan plane, the cost bound for Steiner trees can be improved to $\frac{3}{2} \cdot (1+\frac{1}{\epsilon})$ times optimal, and in the Euclidean plane, the Steiner tree cost bound improves even further, to $\frac{2}{\sqrt{3}} \cdot (1+\frac{1}{\epsilon})$ times optimal. This series of results is in some sense surprising since construction of a minimum spanning tree with bounded diameter in a general graph is NP-complete [HLC89], as is the Steiner problem in graphs [GJ77].

Our construction can minimize either tree cost (yielding a minimum spanning tree) or the longest source-sink path (yielding a minimum-radius tree), depending respectively on whether we set $\epsilon = \infty$ or $\epsilon = 0$. Between these two extremes, the method offers a continuous, smooth tradeoff between the competing requirements of minimum radius and minimum tree cost.

In VLSI circuit design, timing is actually path-dependent, rather than net-

dependent. In other words, timing constraints are established with respect to the delays between primary inputs and primary outputs. We may therefore wish to use varying constraints on the different root-leaf paths within the tree (e.g., a source-sink connection on a timing-critical path will require a small value of ϵ, while a connection not on any critical path can allow large ϵ). Similarly, we may wish to impose different delay bounds R_i on each terminal. Our method extends to handle these cases, and we establish analogous constant-factor bounds on both the cost and the radius of the resulting tree.

The remainder of this chapter is organized as follows. In Section 4.2, we present a general formulation of the bounded-radius tree problem. In Section 4.3, we give a natural heuristic construction (as well as several simple variants) for computing bounded-radius trees which exhibits good empirical performance. In Section 4.4, we present another effective algorithm for computing bounded-radius trees, and show that the algorithm is provably good in the sense that it has constant-factor performance bounds with respect to both the radius and cost of the resulting tree. Section 4.5 generalizes this latter method to Steiner trees, and Section 4.6 extends the approach in two ways: (i) where different values of ϵ are allowed within a given tree, and (ii) where different delay bounds R_i are associated with each terminal. Experimental results are reported in Section 4.7.

4.2 The Problem Formulation

Recall from Chapter 3 that circuit routing applications motivate two flavors of problem embedding. In the case of highly granular cell-based layouts, the cost of routing between two terminals is essentially given by geometric distance, while in the case of general cell design, the routing cost is the cost of the shortest path in a

graph of routing costs (the channel intersection graph [DAK85]) between nodes. While the general graph formulation subsumes the geometric formulation, the latter has special structure that may be exploited to yield tighter performance bounds. Thus, in this discussion we treat the graph and geometric formulations separately.

We begin by defining the underlying *routing graph* to be a connected weighted graph $G = (V, E)$, where a net corresponds to a subset $N \subseteq V$ of the nodes in this graph, with one node designated a source and the rest sinks. A routing solution for the net is a tree in G spanning N, which we call the *routing tree* of the net.

Recall that the *cost* of a path in G is the sum of costs of its edges, and a *shortest path* in G between two terminals $x, y \in N$, denoted by $minpath_G(x, y)$, is a path connecting x and y with minimum cost. In a routing 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 $minpath_G(x, y)$. Note that in the geometric case, the cost of $minpath_G(x, y)$ is simply geometric distance, and we use the notation $dist(x, y)$ for clarity.

Definition: The *radius* R of a signal net is the cost of a shortest path in G from the source to the farthest sink, i.e., $R = \max_{x \in N} dist_G(s, x)$.

Definition: The *radius* of a routing tree T, denoted by $r(T)$, is the cost of a (shortest) path in T from the source s to the furthest sink. Clearly, $r(T) \geq R$ for any routing tree T.

According to the linear RC delay model, we minimize the interconnection delay of a net by minimizing the radius of the routing tree. On the other hand, we also prefer a routing tree with small cost. Without this latter consideration, we could simply use the *shortest paths tree* (SPT) of the net, i.e., the union of

all shortest source-sink paths computed by Dijkstra's single-source shortest paths algorithm [PS82]. Although the SPT has the smallest possible radius $r(SPT)$ of any routing tree, the SPT cost might be very high: Figure 4.1 shows a case where the cost of the shortest paths tree can be $\Omega(|N|)$ times greater than the cost of the minimum spanning tree.

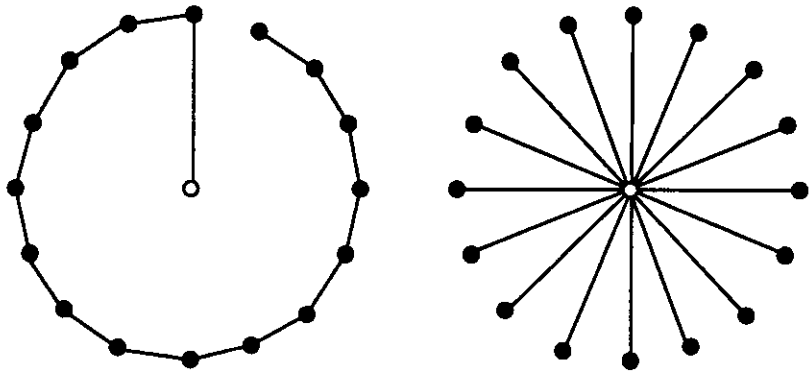


Figure 4.1: An example where the cost of a shortest paths tree (right) is $\Omega(|N|)$ times larger than the cost of a minimum spanning tree (left).

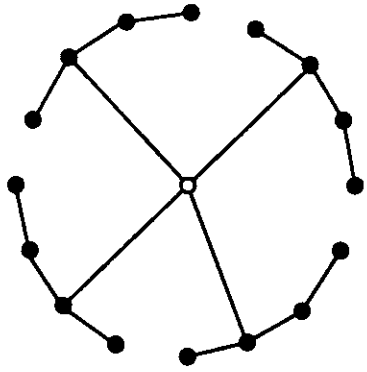


Figure 4.2: A bounded-radius tree, achieving both low cost as well as low radius.

A routing tree with high cost may increase the overall routing area. Moreover, high cost also contributes to the interconnection delays, as well as to system power requirements, which is not captured in the linear RC model. Therefore, neither

tree shown in Figure 4.1 is particularly desirable, and instead, we may prefer a tree such as the one shown in Figure 4.2, which has both low overall cost and low radius. In this chapter we propose a general scheme for achieving this ideal.

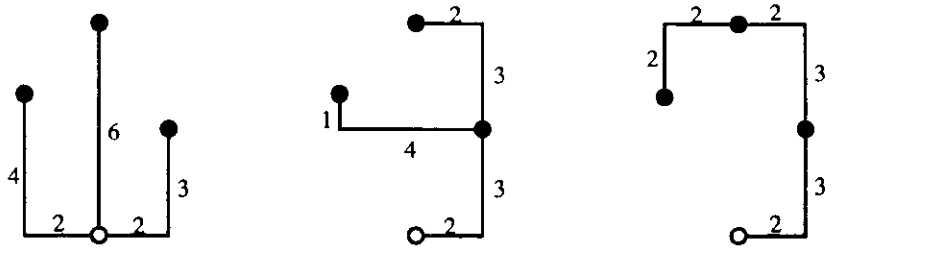
In order to consider both the radius and the cost in the routing tree construction, we use the following bounded-radius minimum routing tree formulation:

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 controls the tradeoff between the radius and the cost of the tree. As we noted earlier, when $\epsilon = 0$ we minimize the radius of the routing tree and thus obtain a minimum-radius tree for the signal net; on the other hand, when $\epsilon = \infty$ we minimize the cost of the tree and obtain a minimum spanning tree. Intuitively, as ϵ grows there is less restriction on the radius, allowing further reduction of tree cost. Figure 4.3 gives an example where three distinct spanning trees are obtained using different values of ϵ: Figure 4.3(a) shows a minimum-radius spanning tree corresponding to the case $\epsilon = 0$, with $r(T) = 6$; Figure 4.3(b) shows a solution with $\epsilon = 1$ and $r(T) = 10$; and Figure 4.3(c) shows the minimum spanning tree corresponding to the case $\epsilon = \infty$, with $r(T) = 14$.

4.3 Bounded-Radius Minimum Spanning Tree Routing

In global routing for cell-based design, all routing costs between terminals are simply given by geometric distance, and so the underlying routing graph is $G = (V, E)$ with $V = N$. For this case, many global routing methods are based on constructing a spanning tree for each net (e.g., see [CP88]). Therefore, the



(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 4.3: An example in the Manhattan plane: increasing ϵ may result in decreased tree cost, but increased tree radius.

BRMRT problem becomes the **Bounded Radius Minimum Spanning Tree** (BRMST) problem.

We now give a simple heuristic that finds a bounded-radius minimum spanning tree solution by growing a single component, following the general scheme of Prim's minimum spanning tree construction [Pri57].

4.3.1 The Basic Algorithm: BPRIM

Our basic algorithm grows a tree $T = (V', E')$ which initially contains only the source s. At each step, we choose $x \in V'$ and $y \in N - V'$ such that $dist(x, y)$ is minimum. If adding the edge (x, y) to T does not violate the radius constraint, i.e., $dist_T(s, x) + dist(x, y) \leq (1+\epsilon) \cdot R$, we include the edge (x, y) in T. Otherwise, we "backtrace" in T along the path from x to s to find the first terminal x' such that the edge (x', y) is *appropriate* (i.e., $dist_T(s, x') + dist(x', y) \leq R$), and add (x', y) to the tree. In the worst case, the backtracing will terminate with $x' = s$, since the edge(s, y) is always appropriate.

Note that in the backtracing we could choose x' such that $dist_T(s, x') + dist(x', y) \leq (1 + \epsilon) \cdot R$. However, our choice of appropriate edges leads to

fewer backtracing operations, while guaranteeing that backtracing is still always possible. In other words, we intentionally introduce some "slack" at y so that terminals within an ϵR neighborhood of y will not cause additional backtracing. Limiting the amount of backtracing in this way will keep the cost of the resulting tree close to that of the minimum spanning tree.

We call this algorithm the **Bounded Prim** (BPRIM) construction; a high-level description is given in Figure 4.4. The BPRIM algorithm is very efficient with the most direct implementation having $O(n^2)$ time complexity since each new terminal can force examination of most of the terminals that have already been added to the tree. This algorithm has several salient properties. First, we can show that the radius of the resulting tree T_{BPRIM} is never greater than the radius of the MST whenever the MST is unique.

BPRIM: computing a bounded-radius spanning tree
Input: A net N with radius R, source s; parameter $\epsilon \geq 0$
Output: A spanning tree T_{BPRIM} with $r(T_{BPRIM}) \leq (1+\epsilon) \cdot R$
$T = (V', E') = (\{s\}, \emptyset)$
While $\|V'\| < \|N\|$
Select two terminals $x \in V'$ and $y \in N - V'$ minimizing $dist(x, y)$
If $dist_T(s, x) + dist(x, y) \leq (1+\epsilon) \cdot R$ **Then** $x' = x$
Else find the first terminal x' along the path in T from x to s such that $dist_T(s, x') + dist(x', y) \leq R$
$V' = V' \cup \{x'\}$
$E' = E' \cup \{(x', y)\}$
Output $T_{BPRIM} = T$

Figure 4.4: Algorithm BPRIM: computing a bounded-radius spanning tree T_{BPRIM} for a given net N, with source $s \in N$ and radius R, using parameter $\epsilon \geq 0$.

Lemma 4.1 *If the MST is unique, then* $r(T_{BPRIM}) \leq r(MST)$.

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

If the MST is *not* unique, then the radii of different minimum spanning trees can vary by an unbounded amount, and $r(T_{BPRIM})$ may be greater than $r(MST)$, i.e., Lemma 4.1 will not hold for some choice of the MST. Figure 4.5 shows a point set where a Prim-like minimum spanning tree algorithm may choose a connection to point y_1 instead of point x_1; or y_2 instead of x_2, etc. so that the tree radius is much greater than optimum. In this way, for some MST it may be possible for an unfortunate sequence of choices by BPRIM to yield $r(T_{BPRIM}) \gg r(MST)$ even though the two trees have identical cost. However, $r(T_{BPRIM})$ cannot be greater than the maximum possible MST radius.

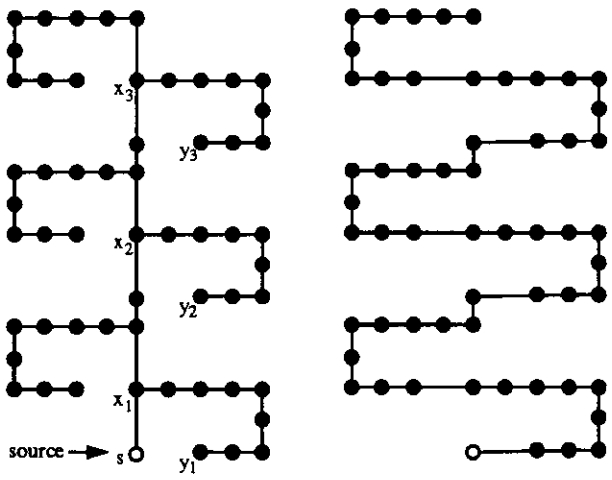


Figure 4.5: An example where the radius of the routing tree (MST) produced by a Prim-like construction (right) is arbitrarily larger than a minimum-radius MST (left).

With regard to tree cost, we note that the difference between BPRIM and MST costs will depend on the parameter ϵ. In practice, most nets will have

between two and four terminals. Furthermore, it is unlikely that a single gate will be used to drive more than six gates in CMOS design. In this case, we can show that the cost of the resulting tree is within a small constant factor of the cost of the MST for nets of practical size. Table 4.1 gives the worst-case ratio of BPRIM cost over MST cost for small values of $|N|$, as a function of ϵ.

Lemma 4.2 *Let $B(\epsilon)$ be the worst-case ratio of the cost of T_{BPRIM} to the MST cost. Then the bounds listed in Table 4.1 hold.*

Net size	Bound B(ϵ)	$\epsilon = 0$	$\epsilon = \frac{1}{2}$	$\epsilon = 1$
$\|N\| = 2$	1	1	1	1
$\|N\| = 3$	$\frac{2}{1+\epsilon}$	2	$\frac{4}{3}$	1
$\|N\| = 4$	$\max(\frac{2+\epsilon}{1+\epsilon}, \frac{3}{1+2\epsilon})$	3	$\frac{5}{3}$	$\frac{3}{2}$
$\|N\| = 5$	$\max(\frac{3+\epsilon}{1+\epsilon}, \frac{4}{1+3\epsilon})$	4	$\frac{7}{3}$	2
$\|N\| = 6$	$\max(\frac{4+\epsilon}{1+\epsilon}, \frac{5}{1+4\epsilon})$	5	3	$\frac{5}{2}$

Table 4.1: Analysis of $B(\epsilon)$ for small nets in the Manhattan plane.

Proof: These results are obtained by studying the number of backtracings that can occur. We give the proof for $|N| = 5$. Other cases are similarly proved.

Assume that the coordinates of the net have been scaled so that the net has unit radius. If backtracing occurs, then $cost(MST) \geq 1 + \epsilon$. Suppose that there is only one backtracing. Let $cost(e)$ be the cost of the edge which caused the backtracing. Then

$$B(\epsilon) \leq \frac{cost(MST) - cost(e) + 1}{cost(MST)}$$

$$\leq 1 + \frac{1}{cost(MST)} \leq 1 + \frac{1}{1+\epsilon} = \frac{2+\epsilon}{1+\epsilon}$$

If backtracing occurs twice, let $cost(x)$ and $cost(y)$ be the costs of the edges which cause the backtracings. Then,

$$B(\epsilon) \leq \frac{cost(MST) - cost(x) - cost(y) + 2}{cost(MST)}$$

$$\leq 1 + \frac{2}{cost(MST)} \leq 1 + \frac{2}{1+\epsilon} = \frac{3+\epsilon}{1+\epsilon}$$

If backtracing occurs three times, T_{BPRIM} must be a star graph and it is easy to see that $cost(MST) \geq 1 + 3\epsilon$. Thus,

$$B(\epsilon) \leq \frac{4}{cost(MST)} \leq \frac{4}{1+3\epsilon}$$

Therefore,

$$B(\epsilon) \leq \max(\frac{2+\epsilon}{1+\epsilon}, \frac{3+\epsilon}{1+\epsilon}, \frac{4}{1+3\epsilon})$$

$$= \max(\frac{3+\epsilon}{1+\epsilon}, \frac{4}{1+3\epsilon})$$

□

Experimental results show that $B(\epsilon)$ is in practice still bounded by a small constant even for large nets (see the tables in Section 4.7). However, examples exist which show that the worst-case performance ratio of BPRIM is not bounded by a constant for any value of ϵ.

Theorem 4.3 *For any ϵ there exists a net for which BPRIM will have an arbitrarily large performance ratio.*

Proof: On the net shown in Figure 4.6, BPRIM will have an unbounded performance ratio. The optimal solution is shown on the left, where source-leaf pathlengths are 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. If ϵ is large, y can be replaced by many closely spaced terminals so that BPRIM creates a long path between s and x; this forces arbitrarily large performance ratio for *any* value of ϵ. □

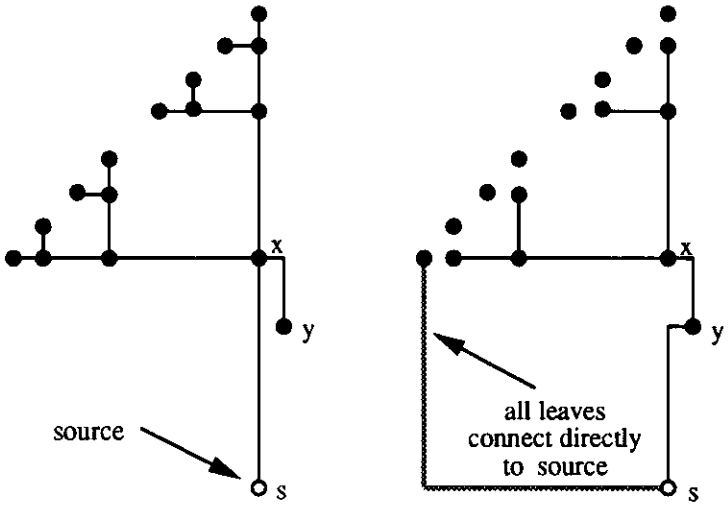


Figure 4.6: Example where the performance ratio of BPRIM is not bounded by a constant for any ϵ. The optimal solution is shown on the left, while the BPRIM output is shown on the right.

4.3.2 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 4.7.

This general template gives rise to a number of distinct variants, depending

Extended-BPRIM: computing a bounded-radius spanning tree
Input: A net N with radius R, source s; parameter $\epsilon \geq 0$
Output: A spanning tree T with $r(T) \leq (1+\epsilon) \cdot R$
$T = (V', E') = (\{s\}, \emptyset)$
While $\|V'\| < \|N\|$
Select two terminals $x \in V'$ and $y \in N - V'$
with $dist_T(s,x) + dist(x,y) \leq (1+\epsilon) \cdot R$
$V' = V' \cup \{x\}$
$E' = E' \cup \{(x,y)\}$
Output T

Figure 4.7: A more general BPRIM template: computing a bounded-radius spanning tree T for a given net N, with source $s \in N$ and radius R, using parameter $\epsilon \geq 0$.

upon how the pair of terminals x and y are selected inside the inner loop. The following variants yield significant performance improvements over the BPRIM algorithm:

- **H1** - Find x and y as in BPRIM, and select a terminal x' along the path in T from x to s which yields a minimum-length appropriate edge (x', y).
- **H2** - Find a terminal $y \in N - V'$ minimizing $dist(x,y)$ for any $x \in V'$, and select the terminal $x' \in V'$ which yields a minimum-length appropriate edge (x', y).
- **H3** - Find a pair of terminals $x \in V'$ and $y \in N - V'$ that yield a minimum-length appropriate edge (x, y).

Lemma 4.1 also holds for each of the variants H1, H2, and H3. The time complexity of variants H1 and H2 is $O(|N|^2)$, while variant H3 can be easily implemented within time $O(|N|^3)$. Empirical results of the BPRIM method are very

promising, as can be seen in Section 4.7. However, Figure 4.6 shows that variant H1 will also have unbounded worst-case performance ratio, and similar examples exist (see Figure 4.8) which establish an unbounded performance ratio for variants H2 and H3. With this in mind, in the next section we develop a provably good approach to performance-driven global routing based on a combination of the minimum spanning tree and shortest paths tree constructions.

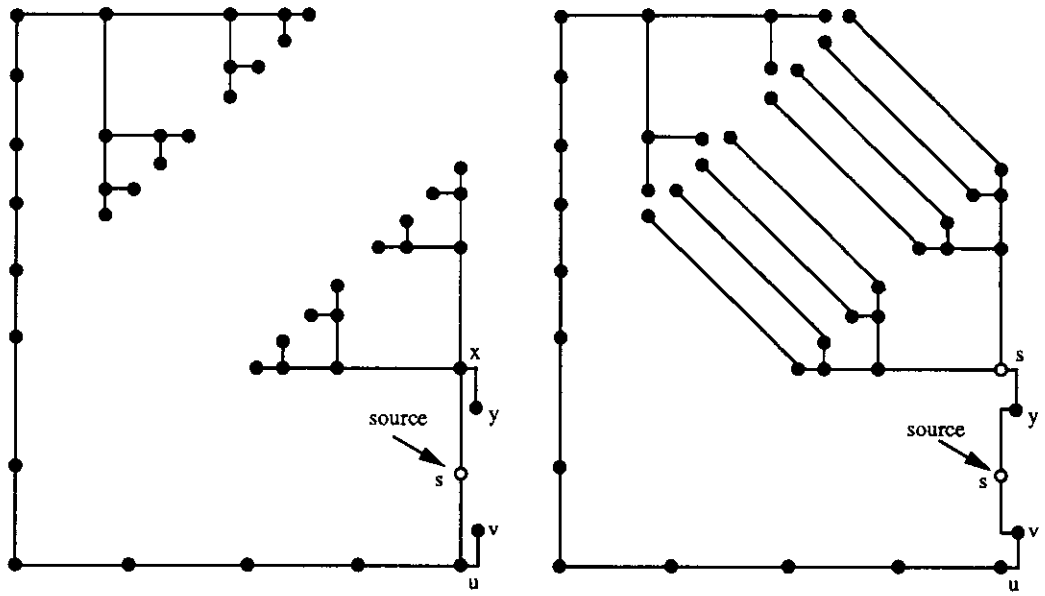


Figure 4.8: Example where the performance ratio of both H2 and H3 is not bounded by a constant for any ϵ. The optimal solution is shown on the left; both T_{H2} and T_{H3} will be identical to the tree shown on the right.

4.4 Bounded-Radius Spanning Trees

The basic idea of our provably good bounded-radius minimum spanning tree algorithm is to construct a subgraph Q of G, such that Q spans N and has both small cost and small radius. Therefore, the shortest paths tree of Q will also have small cost and radius, and will correspond to a good routing solution. We again

use the routing graph $G = (V, E)$ with $V = N$. Our algorithm is as follows:

- Compute the shortest paths tree SPT_G of G, and compute the minimum spanning tree MST_G of G. Also, initialize the graph Q to be equal to MST_G.
- Let L be the sequence of vertices corresponding to a depth-first tour of MST_G; the tour will traverse each edge of MST_G exactly twice (see Figure 4.9), and hence the cost of this tour is $2 \cdot cost(MST_G)$.
- Traverse L while keeping a running total S of traversed edge costs. As this traversal reaches each node L_i, check whether $S > \epsilon \cdot dist_G(s, L_i)$. If so, reset S to 0 and merge $minpath_G(s, L_i)$ into Q. Continue traversing L while repeating this process.
- Our final routing tree is SPT_Q, the shortest paths tree of Q.

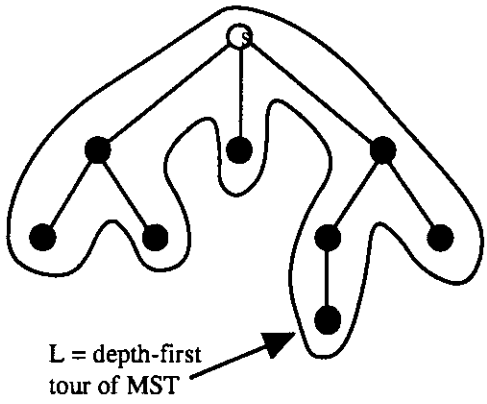


Figure 4.9: A spanning tree and its depth-first tour.

Because our method yields a bounded-radius, bounded-cost routing tree, we call this the BRBC algorithm. The formal description of the algorithm is given in

Figure 4.10. We now prove that for any fixed ϵ the BRBC algorithm produces a routing tree T_{BRBC} with radius and cost simultaneously bounded by small constants times optimum.

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

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

Figure 4.10: Computing a bounded-radius spanning tree T_{BRBC} for $G = (V, E)$, with source $s \in V$ and radius R, using parameter $\epsilon \geq 0$. T_{BRBC} will have radius at most $(1+\epsilon) \cdot R$, and cost at most $(1 + \frac{2}{\epsilon}) \cdot cost(MST_G)$.

Theorem 4.4 *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 $minpath_G(s, v_{i-1})$ to Q (see Figure 4.11). By the construction of the algorithm, we know that $dist_L(v_{i-1}, v) \leq \epsilon \cdot R$. We then have

$$dist_T(s, v) \leq dist_T(s, v_{i-1}) + dist_L(v_{i-1}, v)$$

$$\leq dist_G(s, v_{i-1}) + \epsilon \cdot R$$

$$\leq R + \epsilon \cdot R = (1+\epsilon) \cdot R$$

□

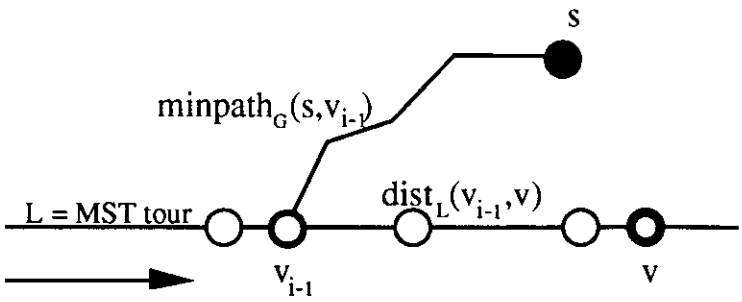


Figure 4.11: Depiction of the bounded-radius construction.

Theorem 4.5 *For any weighted graph G and parameter $\epsilon \geq 0$, $cost(T_{BRBC}) \leq (1 + \frac{2}{\epsilon}) \cdot cost(MST_G)$.*

Proof: Let $v_1, v_2, \ldots, v_m$ be the set of nodes to which the algorithm added shortest paths from the source node, and let $v_0 = s$. We have

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

since T_{BRBC} is a subtree of the union of the MST with all of the added shortest paths. By the algorithm construction, $dist_L(v_{i-1}, v_i) \geq \epsilon \cdot dist_G(s, v_i)$ for all $i = 1, \ldots, m$ and so we obtain

$$cost(T_{BRBC}) \leq cost(MST_G) + \sum_{i=1}^{m} \frac{1}{\epsilon} \cdot dist_L(v_{i-1}, v_i)$$

$$\leq cost(MST_G) + \frac{1}{\epsilon} \cdot cost(L)$$

Since $cost(L) = 2 \cdot cost(MST_G)$, we have

$$cost(T_{BRBC}) \leq cost(MST_G) + \frac{2}{\epsilon} \cdot cost(MST_G)$$

$$= (1 + \frac{2}{\epsilon}) \cdot cost(MST_G)$$

□

Theorem 4.5 suggests that for $\epsilon = 0$, the ratio $\frac{cost(T_{BRBC})}{cost(MST_G)}$ is not bounded by any constant, and indeed this is illustrated in the example of Figure 4.1 above, where $\frac{cost(T_{BRBC})}{cost(MST_G)}$ is $\Omega(|N|)$.

It should be noted that a method similar to the BRBC construction was recently used in the distributed computation literature by Awerbuch, Baratz and Peleg [ABP90] for constructing spanning trees with small diameter and small weight. However, our algorithm treats the bounded-radius minimum spanning tree problem, while they treat the tree diameter instead. Moreover, our method entails a simpler construction with tighter performance bounds.

4.5 Bounded-Radius Steiner Trees

In the previous section, we treated the bounded-radius minimum spanning tree problem, where each net N is routed in an underlying routing graph $G = (V, E)$ with $V = N$. In this section we treat the more general version of the problem, where Steiner points are allowed.

4.5.1 Algorithm for Graphs With Steiner Points

When we seek to connect a subset of the nodes in a graph and are allowed to use the remaining nodes as Steiner points[2], the BRMRT problem becomes the **Bounded-Radius Optimal Steiner Tree** (BROST) problem. The following result is immediate:

Lemma 4.6 *The BROST problem is NP-complete.*

[2]e.g., in building-block VLSI design where the underlying routing graph is based on the channel intersection graph [DAK85].

Proof: Setting $\epsilon = \infty$ yields the graph Steiner problem, which is known to be NP-complete [GJ77]. □

Hence, in the BROST problem, even the construction of a "minimum spanning tree" for N in G is equivalent to the Steiner problem in graphs. This means that if we are to apply the BRBC approach to the BROST problem while maintaining polynomial complexity, we must begin by approximating the minimum-cost tree spanning N within G, i.e., the minimum Steiner tree over N.

Recall that the BRBC algorithm used the MST to obtain a reasonably short tour of the vertices. Observe that *any* tour of the vertices having reasonably small cost will suffice (e.g., traveling salesman, Chinese postman), and moreover in constructing this tour our only requirement is that the tour visits every node in N.

Our approximation algorithm for the bounded-radius optimal Steiner tree problem is similar to the algorithm presented in Section 4.4: given any approximate Steiner tree $\hat{T}$, we can use the approach of Section 4.4 to construct a routing tree having radius bounded by $(1+\epsilon)\cdot r(\hat{T})$, and cost bounded by $(1+\frac{2}{\epsilon})\cdot cost(\hat{T})$. We use a heuristic of Kou, Markowsky and Berman (KMB) [KMB81] [WWW86] to build a Steiner tree $\hat{T} = T_{KMB}$ in the underlying routing graph; T_{KMB} will have cost within a factor 2 of optimal.[3]

We construct a depth-first tour L of the heuristic Steiner tree T_{KMB}. Next, we traverse L, adding to T_{KMB} shortest paths from the source to selected vertices of

[3]Given a graph $G = (V, E)$ and a net of terminals $N \subseteq V$, the method of Kou, Markovsky and Berman is as follows. First, construct the complete graph G' over N with each edge weight equal to the cost of the corresponding shortest path in G. Compute $MST_{G'}$, the minimum spanning tree of G', and expand each edge of $MST_{G'}$ into the corresponding shortest path yielding a subgraph G'' that spans N. Finally, compute the minimum spanning tree $MST_{G''}$ of G'', and delete pendant edges from $MST_{G''}$ until all leaves are members of N. Output the resulting tree.

L, as in Section 4.4. Finally, we compute the shortest paths tree in the resulting graph and output the union of all shortest paths from the source to terminals in N (note that this will include intermediate non-terminal nodes on the shortest paths as Steiner points). We call this Steiner version the BRBC_S algorithm. Since the cost of L will be at most 4 times the *optimal* Steiner tree (T_{opt}) cost, the cost of T_{BRBC_S} is at most $2 \cdot (1 + \frac{2}{\epsilon})$ times optimal.[4]

Theorem 4.7 *For any weighted graph $G = (V, E)$, node subset $N \subseteq V$ 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 our 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 $cost(T_{BRBC\_S}) \leq 2 \cdot (1 + \frac{2}{\epsilon}) \cdot cost(T_{opt})$. □

4.5.2 Geometry Helps in Routing

If we are routing in a metric space and are allowed to introduce arbitrary Steiner points to reduce the tree cost/radius, we can slightly modify the basic algorithm (of Figure 4.10) to introduce Steiner points on the tour L whenever $S = 2\epsilon \cdot R$. From each of these Steiner points we construct shortest paths to the source and add them to Q as in the original $BRBC$ algorithm. Thus, each node in the tour L will be within $\epsilon \cdot R$ of a Steiner point, i.e., within $(1+\epsilon) \cdot R$ of the source. Because it exploits inherent geometry, we call this version the BRBC_G algorithm. The following radius and cost bounds hold, with the proofs of these bounds similar to those of Theorems 4.4 and 4.5.

[4] Using a recent graph Steiner heuristic of Zelikovsky [Zel92], this cost bound may be further reduced to $\frac{11}{6} \cdot (1 + \frac{2}{\epsilon})$ times optimal.

Theorem 4.8 *In the geometric plane, for given parameter* $\epsilon \geq 0$, $r(T_{BRBC\_G}) \leq (1+\epsilon) \cdot R$ *and* $cost(T_{BRBC\_G}) \leq 2 \cdot (1+\frac{1}{\epsilon}) \cdot cost(T_{opt})$.

In addition, well-known results which bound the $\frac{MST}{Steiner}$ ratio in various geometries can be used with the above scheme to yield even better bounds whenever the edge weights are induced by an underlying norm (e.g., Manhattan or Euclidean). To illustrate how these observations can be combined to yield improved bounds for Steiner routing in metric spaces, we give two immediate examples:

Corollary 4.9 *Given a set of terminals* N *in the Manhattan plane and a real parameter* $\epsilon \geq 0$, $r(T_{BRBC\_G}) \leq (1+\epsilon) \cdot R$ *and* $cost(T_{BRBC\_G}) \leq \frac{3}{2} \cdot (1+\frac{1}{\epsilon}) \cdot T_{opt}$.

Proof: By a result of Hwang [Hwa76], the rectilinear minimum spanning tree gives a $\frac{3}{2}$ approximation to the optimal rectilinear Steiner tree.[5] We then apply arguments similar to those used for Theorems 4.4 and 4.5. □

Corollary 4.10 *Given a set of terminals* N *in the Euclidean plane and a real parameter* ϵ, $r(T_{BRBC\_G}) \leq (1+\epsilon) \cdot R$, *and* $cost(T_{BRBC\_G}) \leq \frac{2}{\sqrt{3}} \cdot (1+\frac{1}{\epsilon}) \cdot T_{opt}$.

Proof: By a recent result of Du and Hwang [DH90], 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 4.4 and 4.5. □

Note that this result generalizes when we have increased flexibility in the wiring geometry, e.g., 30-60-90 degree wiring instead of rectilinear. By applying a recent result of [SW92] 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 4.10 above.

[5] Using recent results of Berman and Ramaiyer [BR92], this constant may be further reduced to $\frac{11}{8}$.

4.6 Generalization to Non-Uniform Values of ϵ

Often we may wish to use varying wirelength constraints on the different source-sink paths within a given signal net, since timing in VLSI circuits is actually path-dependent, rather than net-dependent. For example, a source-sink connection on a timing-critical path will require a small value of ϵ, whereas for a connection not on any critical path, we may allow large ϵ in order to reduce tree cost. This yields the following generalization of the BRMRT formulation:

The Non-Uniform Bounded-Radius Minimum Routing Tree (NBRMRT) Problem: Given parameters $\epsilon_i \geq 0$ associated with each sink terminal n_i of a signal net having source s and radius R, find a minimum-cost routing tree T such that $dist_T(s, n_i) \leq (1 + \epsilon_i) \cdot R$ for each n_i.

In this section, we extend our method to handle this case, and establish constant-factor bounds on both the cost and radius of the routing solution. Although we restrict the discussion to spanning tree routing, extensions to (geometric) Steiner routing are straightforward using the techniques of Sections 4.4 and 4.5.

To handle a different pathlength constraint ϵ_i for each terminal n_i in the net N, we modify the original algorithm of Figure 4.10 by changing the conditional inside the loop from "$S \geq \epsilon \cdot dist_G(s, L_{i+1})$" to "$S \geq \epsilon_{i+1} \cdot dist_G(s, L_{i+1})$". We call this modified algorithm BRBC_ϵ_i, and show the following bound on the pathlengths using an argument analogous to that in the proof of Theorem 4.4:

Lemma 4.11 *For an arbitrary weighted graph G, with source s and radius R, and terminal radius parameters $\epsilon_1, \epsilon_2, \ldots, \epsilon_{|N|}$, $dist_{T_{BRBC\_\epsilon_i}}(s, n_i) \leq (1 + \epsilon_i) \cdot R$ for each terminal n_i.*

Arguments similar to those used earlier yield the bound $cost(T_{BRBC\text{-}\epsilon_i}) \leq (1 + \frac{2}{min(\epsilon_1,\epsilon_2,\ldots,\epsilon_{|N|})}) \cdot cost(MST_G)$. However, we may improve this bound as follows. Without loss of generality, we can assume that all of the ϵ_i's are sorted in non-decreasing order: $\epsilon_1 \leq \epsilon_2 \leq \ldots \leq \epsilon_{|N|}$. Let $\epsilon = \max_{i=1}^{|N|} \epsilon_i$, and define $k = \lceil \frac{2 \cdot cost(MST_G)}{(1+\epsilon) \cdot R} \rceil$.

Lemma 4.12 *For any weighted graph G and terminal radius parameters $\epsilon_1 \leq \epsilon_2 \leq \ldots \leq \epsilon_{|N|}$, $cost(T_{BRBC\text{-}\epsilon_i}) \leq (1 + \frac{k}{k-1} \cdot \frac{2}{HM(\epsilon_1,\epsilon_2,\ldots,\epsilon_k)}) \cdot cost(MST_G)$, where HM denotes harmonic mean.*

Proof: Let $v_1, v_2, \ldots, v_m$ be the set of nodes to which the algorithm added shortest paths from the source node, as shown in Figure 4.12. As before, the routing tree produced by our modified algorithm is a subtree of Q, the union of MST_G and the added shortest paths. The routing tree cost is therefore bounded by:

$$cost(Q) = cost(MST_G) + \sum_{i=1}^{m} dist_G(s, v_i)$$

$$\leq cost(MST_G) + \sum_{i=1}^{m} \frac{1}{\epsilon_i} dist_L(v_{i-1}, v_i)$$

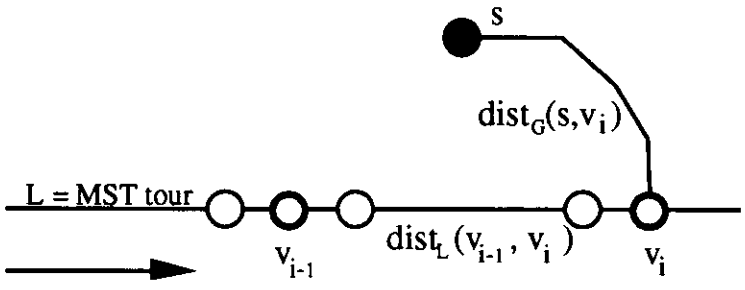


Figure 4.12: Tree construction using non-uniform values of ϵ.

Let l_i denote $dist_L(v_{i-1}, v_i)$. By the construction, we have $l_i \geq \epsilon_i \cdot dist_G(s, v_i)$. Because no edge length is greater than R, we have $l_i \leq (1 + \epsilon) \cdot R$ and

$$\sum_{i=1}^{m} l_i = 2 \cdot cost(MST_G) \leq k \cdot (1+\epsilon) \cdot R$$

Therefore,

$$\sum_{i=1}^{m} \frac{1}{\epsilon_i}\, dist_L(v_{i-1}, v_i) = \sum_{i=1}^{m} \frac{l_i}{\epsilon_i} \leq \sum_{i=1}^{k} \frac{(1+\epsilon) \cdot R}{\epsilon_i}$$

since $l_i \leq (1+\epsilon) \cdot R$, $\sum_{i=1}^{m} l_i \leq k \cdot (1+\epsilon) \cdot R$, and the ϵ_i's are in sorted order. Factoring out $(1+\epsilon)$ and using the definition of k, we obtain

$$= (1+\epsilon) \cdot \sum_{i=1}^{k} \frac{1}{\epsilon_i} \cdot R \leq (1+\epsilon) \cdot \sum_{i=1}^{k} \frac{1}{\epsilon_i} \cdot \frac{2 \cdot cost(MST_G)}{(1+\epsilon) \cdot (k-1)}$$

Cancelling $(1+\epsilon)$, multiplying by $\frac{k}{k}$, and regrouping, we get

$$\leq \frac{k}{k-1} \cdot \frac{\sum_{i=1}^{k} \frac{1}{\epsilon_i}}{k} \cdot 2 \cdot cost(MST_G)$$

$$= \frac{k}{k-1} \cdot \frac{1}{HM(\epsilon_1, \epsilon_2, \ldots, \epsilon_k)} \cdot 2 \cdot cost(MST_G)$$

It follows that

$$cost(Q) \leq cost(MST_G) + \frac{k}{k-1} \cdot \frac{1}{HM(\epsilon_1, \epsilon_2, \ldots, \epsilon_k)} \cdot 2 \cdot cost(MST_G)$$

$$= (1 + \frac{k}{k-1} \cdot \frac{2}{HM(\epsilon_1, \epsilon_2, \ldots, \epsilon_k)}) \cdot cost(MST_G)$$

□

These results are summarized as follows:

Theorem 4.13 *For any weighted graph G and terminal radius parameters $\epsilon_1 \leq \epsilon_2 \leq \ldots \leq \epsilon_{|N|}$, each terminal n_i in $T_{BRBC\text{-}\epsilon_i}$ has $dist_{T_{BRBC\text{-}\epsilon_i}}(s, n_i) \leq (1+\epsilon_i) \cdot R$, and $cost(T_{BRBC\text{-}\epsilon_i}) \leq (1 + \frac{k}{k-1} \cdot \frac{2}{HM(\epsilon_1, \epsilon_2, \ldots, \epsilon_k)}) \cdot cost(MST_G)$, where $k \lceil \frac{2 \cdot cost(MST_G)}{(1+\epsilon) \cdot R} \rceil$ and HM denotes harmonic mean.*

In the case that we wish to impose different delay bounds R_i on each terminal so that $dist_T(s, n_i) \leq (1 + \epsilon) \cdot R_i$, we need only introduce non-uniform ϵ_i values, with $\epsilon_i = \epsilon \cdot \frac{R_i}{R}$. With this transformation, the $BRBC\_\epsilon_i$ algorithm may now be directly applied. Such timing constraints are typically induced by performance-driven placement tools [SCK91a], and must be satisfied by the global router.

4.7 Experimental Results

The BPRIM algorithm and variants H1, H2 and H3, as well as the BRBC and BRBC_S algorithms, were implemented in ANSI C for the Sun-4, Macintosh and IBM environments.

The BPRIM algorithm and variants H1, H2, and H3 were tested on a large number of random nets of up to 50 terminals, generated from a uniform distribution in the 1000 x 1000 grid. As noted in Chapter 2, any set of approximation heuristics induces a *meta-heuristic* which returns the best solution found by any heuristic in the set and which has asymptotic complexity equal to that of the slowest heuristic. We implemented the meta-heuristic over BPRIM, H1, H2 and H3, denoted by Meta(BPRIM,H1,H2,H3), which returns the routing tree with minimum cost.

Although there exist examples where the BPRIM algorithm outperforms the more complicated variants (e.g., see Figure 4.13), the data shown in the Tables of this section indicate that on average, variant H1 dominates BPRIM, H2 dominates H1, and H3 dominates H2 (Tables 4.2 through 4.5). Figures 4.14 and 4.15 show that the BPRIM approach produces a smooth tradeoff between tree cost and tree radius.

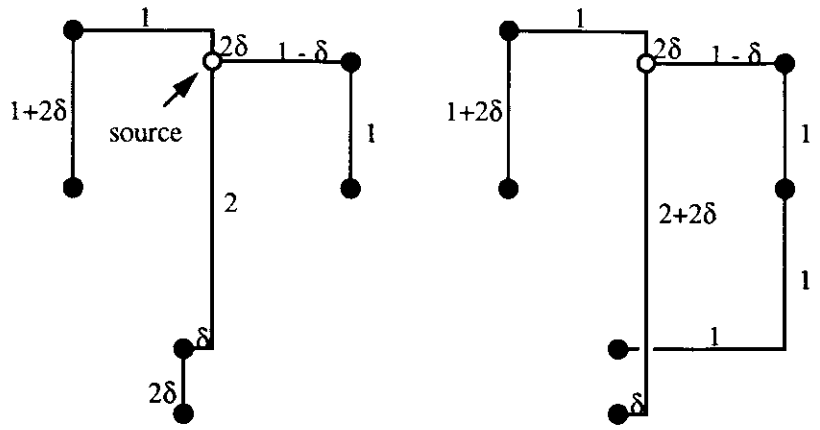


Figure 4.13: An example where BPRIM (left) outperforms variants H2 and H3 (right); here δ is a very small real number and $\epsilon = (2 - 3\delta)/(2 + 3\delta)$.

The BRBC algorithm was also tested on a large number of random nets generated from a uniform distribution in the grid. Results are summarized in Figures 4.16 and 4.17, and clearly show the smooth tradeoff between cost and radius. As ϵ decreases, both the cost and radius curves shift monotonically from that of the minimum spanning tree to that of the shortest paths tree. A more detailed account of the performance of BRBC is given in Tables 4.6 and 4.7.

The BRBC_S algorithm was tested on random block layouts in the grid; these were generated by adding a fixed number of non-overlapping blocks, with length, width and lower-left coordinates all chosen randomly from a uniform distribution. Given a block design, nets with terminals on the block peripheries were routed within the corresponding channel intersection graph [DAK85]. An example of the output from the *BRBC_S* algorithm is shown in Figure 4.18.

A detailed summary of experimental results for the performance of BRBC_S in block designs is given in Tables 4.8 and 4.9. Once again, the simulations confirm the tradeoffs inherent in the bounded-radius routing approach. Note

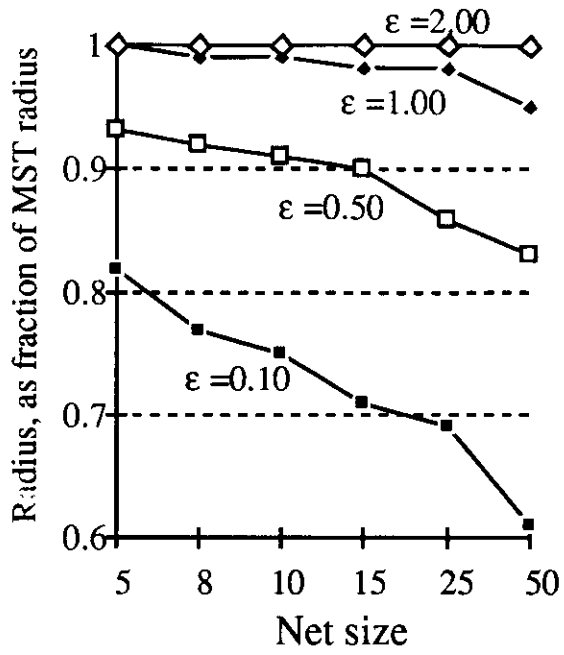


Figure 4.14: This chart illustrates the smooth tradeoff between tree cost and tree radius produced by the BPRIM algorithm. The parameter ϵ determines the tradeoff between the shortest paths tree and the minimum spanning tree; as ϵ increases, the resulting tree increasingly reflects the shortest paths tree in terms of radius.

that although our construction starts with the heuristic Steiner tree of Kou, Markovsky and Berman, our routing solution may in some cases have smaller cost than the KMB tree. In all cases, the radius of T_{BRBC_S} is no larger than that of the KMB tree. This too is reflected in the experimental data.

To validate the use of the linear delay model, the HSPICE circuit simulation package was used to examine a number of routing trees. As an example, Figure 4.19 shows how the optimal delay routing indeed embodies a tradeoff between the shortest paths tree routing and the minimum spanning tree routing.

From the tables that follow, we observe the following. For any given value of ϵ, the BPRIM approach, being inherently greedy, will yield a routing solution

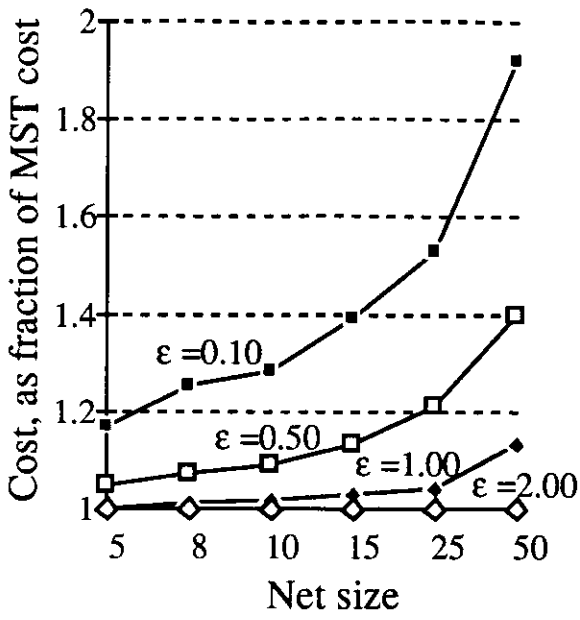


Figure 4.15: This chart illustrates the smooth tradeoff between tree cost and tree radius produced by the BPRIM algorithm. The parameter ϵ determines the tradeoff between the shortest paths tree and the minimum spanning tree; as ϵ increases, the resulting tree increasingly reflects the minimum spanning tree in terms of cost.

with radius approaching $(1+\epsilon)\cdot R$, but having small cost. On the other hand, the BRBC approach, being more conservative, will yield a routing solution with radius noticeably smaller than $(1+\epsilon)\cdot R$, at the expense of slightly larger tree cost. Therefore, the BRBC algorithm will have a slightly shifted cost-radius curve compared to the BPRIM algorithm. In practice, the asymptotic efficiency of implementation and the provably good output would suggest choosing the BRBC algorithm over BPRIM.

Tables 4.2 and 4.3 give the minimum, maximum and average radius performance ratios $\frac{r(T)}{r(MST)}$, for T computed by BPRIM and its variants H1, H2, H3. as well as Meta(BPRIM,H1,H2,H3). The data shown represent averages of 500 cases generated from a uniform distribution in the unit square. The source node

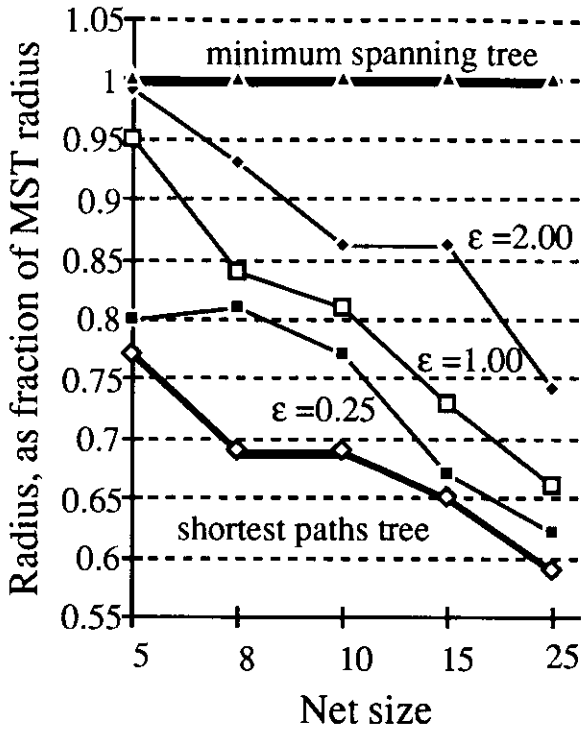


Figure 4.16: This chart illustrates the smooth tradeoff between tree cost and tree radius produced by the BRBC algorithm. The envelope of performance lies between the shortest paths tree and the minimum spanning tree, and the parameter ϵ determines the exact tradeoff.

was selected to be one of the terminals at random. Note that the radius of the Meta(BPRIM,H1,H2,H3) solution may be larger than the radius produced by any single method, because the meta-heuristic selects the lowest-cost tree.

Tables 4.4 and 4.5 give the minimum, maximum and average cost performance ratios $\frac{cost(T)}{cost(MST)}$, for T computed by BPRIM and its variants H1, H2, H3, as well as Meta(BPRIM,H1,H2,H3). Again, the data represent averages of 500 random cases, each with a randomly chosen source node.

Tables 4.6 and 4.7 show the cost and radius of T_{BRBC} and the SPT, as compared to the corresponding MST values. For each ϵ value and net cardinality,

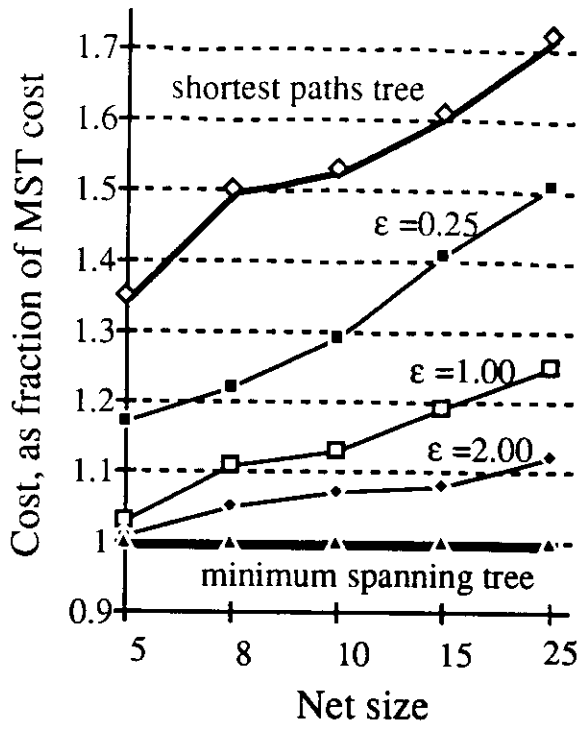


Figure 4.17: This chart illustrates the smooth tradeoff between tree cost and tree radius produced by the BRBC algorithm. The envelope of performance lies between the shortest paths tree and the minimum spanning tree, and the parameter ϵ determines the exact tradeoff.

50 random test cases were generated from a uniform distribution in the unit square, and the minimum, average and maximum values computed. The source was selected to be one of the terminals at random.

Finally, Tables 4.8 and 4.9 show the cost and radius of T_{BRBC_S} and the SPT, as compared to the corresponding KMB values. For each ϵ value and net cardinality, 50 test cases were generated, each with 15 randomly placed modules and a randomly selected source. Routing was then performed in the induced channel intersection graph.

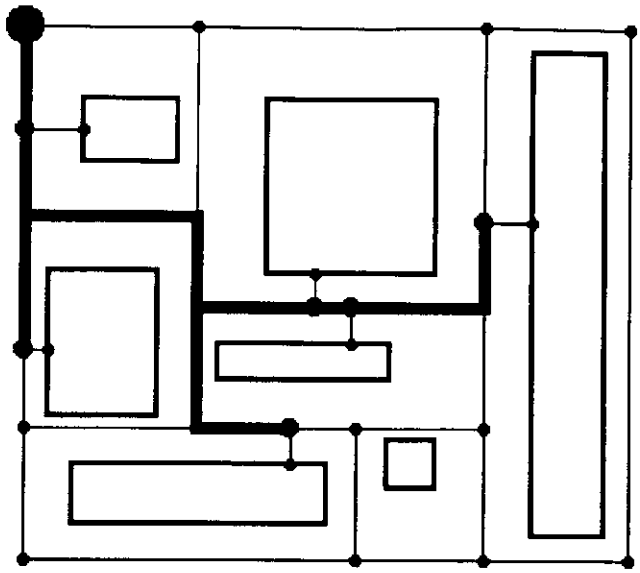


Figure 4.18: A set of placed modules and their channel intersection graph. The highlighted tree is the routing produced by the BRBC_S algorithm.

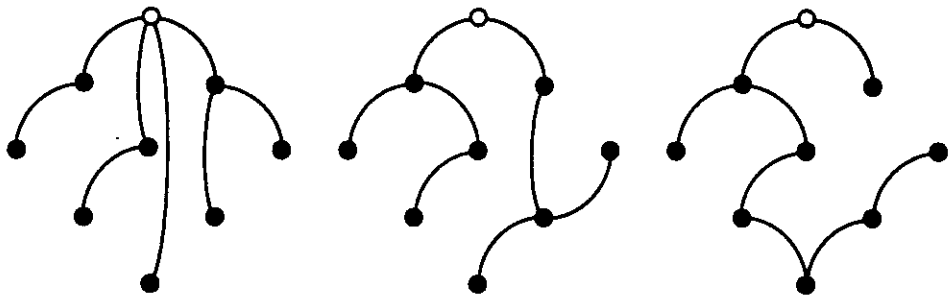


Figure 4.19: An example in the Manhattan plane where an HSPICE simulation indicated that T_{BRBC} (middle) outperforms an MST (right) by 81 picoseconds, and outperforms the SPT routing (left) by 414 picoseconds. The coordinates of the terminals are { (102,98), (147,153), (202,202), (153,249), (53,147), (253,153), (153,52), (100,203), (200,103) }, and $\epsilon = 1.5$. The simulation assumes a generic CMOS design: MOSIS 2.0μ CMOS technology, layout normalized to a 1cm die, and 0.3pF capacitive gate loading.

		BPRIM			H1			H2		
ϵ	$\|N\|$	Min	Ave	Max	Min	Ave	Max	Min	Ave	Max
0.10	5	0.42	0.82	1.00	0.42	0.82	1.00	0.42	0.82	1.00
0.10	8	0.26	0.77	1.00	0.26	0.77	1.00	0.28	0.77	1.00
0.10	10	0.36	0.75	1.00	0.36	0.74	1.00	0.36	0.75	1.00
0.10	15	0.34	0.71	1.00	0.34	0.71	1.00	0.34	0.71	1.00
0.10	25	0.33	0.69	1.00	0.33	0.68	1.00	0.33	0.69	1.00
0.10	50	0.30	0.61	0.99	0.30	0.61	0.99	0.31	0.61	0.99
0.50	5	0.41	0.93	1.00	0.41	0.92	1.00	0.41	0.92	1.00
0.50	8	0.48	0.92	1.00	0.33	0.92	1.00	0.33	0.92	1.00
0.50	10	0.46	0.91	1.00	0.45	0.90	1.00	0.45	0.90	1.00
0.50	15	0.44	0.90	1.00	0.44	0.89	1.00	0.44	0.89	1.00
0.50	25	0.38	0.86	1.00	0.37	0.86	1.00	0.37	0.86	1.00
0.50	50	0.39	0.83	1.00	0.39	0.83	1.00	0.38	0.82	1.00
1.00	5	0.58	1.00	1.00	0.58	0.99	1.00	0.58	0.99	1.00
1.00	8	0.67	0.99	1.00	0.56	0.99	1.00	0.56	0.99	1.00
1.00	10	0.65	0.99	1.00	0.57	0.98	1.00	0.57	0.98	1.00
1.00	15	0.65	0.98	1.00	0.54	0.98	1.00	0.54	0.97	1.00
1.00	25	0.48	0.98	1.00	0.48	0.97	1.00	0.48	0.97	1.00
1.00	50	0.53	0.95	1.00	0.53	0.94	1.00	0.53	0.94	1.00
2.00	5	1.00	1.00	1.00	1.00	1.00	1.00	0.69	1.00	1.00
2.00	8	0.86	1.00	1.00	0.80	1.00	1.00	0.67	1.00	1.00
2.00	10	0.96	1.00	1.00	0.96	1.00	1.00	0.78	1.00	1.00
2.00	15	1.00	1.00	1.00	1.00	1.00	1.00	0.85	1.00	1.00
2.00	25	0.84	1.00	1.00	0.84	1.00	1.00	0.81	1.00	1.00
2.00	50	0.82	1.00	1.00	0.82	1.00	1.00	0.78	1.00	1.00

Table 4.2: Minimum, average and maximum radius ratios for various values of ϵ, expressed as a fraction of MST radius.

		H3			Meta		
ϵ	$\|N\|$	Min	Ave	Max	Min	Ave	Max
0.10	5	0.42	0.82	1.00	0.42	0.82	1.00
0.10	8	0.28	0.77	1.00	0.28	0.77	1.00
0.10	10	0.36	0.75	1.00	0.36	0.75	1.00
0.10	15	0.34	0.71	1.00	0.34	0.71	1.00
0.10	25	0.32	0.69	1.00	0.32	0.69	1.00
0.10	50	0.30	0.61	0.99	0.30	0.61	0.99
0.50	5	0.41	0.92	1.00	0.41	0.92	1.00
0.50	8	0.44	0.92	1.00	0.44	0.92	1.00
0.50	10	0.48	0.90	1.00	0.48	0.90	1.00
0.50	15	0.41	0.89	1.31	0.41	0.89	1.31
0.50	25	0.37	0.86	1.07	0.37	0.86	1.07
0.50	50	0.39	0.82	1.04	0.39	0.82	1.04
1.00	5	0.58	0.99	1.00	0.58	0.99	1.00
1.00	8	0.53	0.99	1.00	0.53	0.99	1.00
1.00	10	0.57	0.98	1.00	0.57	0.98	1.00
1.00	15	0.54	0.97	1.06	0.54	0.97	1.06
1.00	25	0.46	0.97	1.10	0.46	0.97	1.10
1.00	50	0.53	0.94	1.06	0.53	0.94	1.06
2.00	5	0.69	1.00	1.00	0.69	1.00	1.00
2.00	8	0.67	1.00	1.00	0.67	1.00	1.00
2.00	10	0.78	1.00	1.18	0.78	1.00	1.18
2.00	15	0.85	1.00	1.10	0.85	1.00	1.10
2.00	25	0.69	1.00	1.26	0.69	1.00	1.26
2.00	50	0.58	0.99	1.16	0.58	0.99	1.16

Table 4.3: Minimum, average and maximum radius ratios for various values of ϵ, expressed as a fraction of MST radius (continued).

		BPRIM			H1			H2		
ϵ	$\|N\|$	Min	Ave	Max	Min	Ave	Max	Min	Ave	Max
0.10	5	1.00	1.17	2.22	1.00	1.17	2.22	1.00	1.17	2.22
0.10	8	1.00	1.25	2.20	1.00	1.23	1.94	1.00	1.22	2.26
0.10	10	1.00	1.28	2.33	1.00	1.26	2.33	1.00	1.25	2.18
0.10	15	1.00	1.39	2.79	1.00	1.32	2.77	1.00	1.28	2.53
0.10	25	1.00	1.53	2.71	1.00	1.39	2.45	1.00	1.33	2.30
0.10	50	1.00	1.92	3.49	1.00	1.52	2.91	1.00	1.41	2.92
0.50	5	1.00	1.05	1.60	1.00	1.04	1.56	1.00	1.04	1.56
0.50	8	1.00	1.07	1.97	1.00	1.05	1.59	1.00	1.05	1.59
0.50	10	1.00	1.09	1.73	1.00	1.06	1.59	1.00	1.06	1.59
0.50	15	1.00	1.13	2.08	1.00	1.08	1.60	1.00	1.06	1.53
0.50	25	1.00	1.21	2.91	1.00	1.10	1.97	1.00	1.08	1.88
0.50	50	1.00	1.40	3.67	1.00	1.15	1.93	1.00	1.10	1.75
1.00	5	1.00	1.00	1.27	1.00	1.00	1.27	1.00	1.00	1.27
1.00	8	1.00	1.01	1.73	1.00	1.01	1.54	1.00	1.01	1.54
1.00	10	1.00	1.02	1.47	1.00	1.01	1.32	1.00	1.01	1.31
1.00	15	1.00	1.03	1.79	1.00	1.02	1.30	1.00	1.01	1.30
1.00	25	1.00	1.04	2.38	1.00	1.02	1.39	1.00	1.01	1.37
1.00	50	1.00	1.13	2.66	1.00	1.04	1.71	1.00	1.03	1.47
2.00	5	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
2.00	8	1.00	1.00	1.34	1.00	1.00	1.07	1.00	1.00	1.07
2.00	10	1.00	1.00	1.08	1.00	1.00	1.08	1.00	1.00	1.08
2.00	15	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
2.00	25	1.00	1.00	1.39	1.00	1.00	1.14	1.00	1.00	1.14
2.00	50	1.00	1.00	1.71	1.00	1.00	1.13	1.00	1.00	1.11

Table 4.4: Minimum, average and maximum cost ratios for various values of ϵ, expressed as a fraction of MST cost.

		H3			Meta		
ϵ	$\|N\|$	Min	Ave	Max	Min	Ave	Max
0.10	5	1.00	1.16	2.22	1.00	1.16	2.22
0.10	8	1.00	1.20	2.26	1.00	1.20	1.94
0.10	10	1.00	1.23	2.18	1.00	1.22	2.18
0.10	15	1.00	1.25	2.28	1.00	1.23	2.28
0.10	25	1.00	1.28	2.16	1.00	1.25	2.00
0.10	50	1.00	1.33	2.22	1.00	1.30	2.22
0.50	5	1.00	1.04	1.56	1.00	1.04	1.56
0.50	8	1.00	1.04	1.84	1.00	1.04	1.59
0.50	10	1.00	1.05	1.59	1.00	1.05	1.59
0.50	15	1.00	1.05	1.53	1.00	1.05	1.53
0.50	25	1.00	1.05	1.72	1.00	1.05	1.72
0.50	50	1.00	1.06	1.77	1.00	1.06	1.74
1.00	5	1.00	1.00	1.27	1.00	1.00	1.27
1.00	8	1.00	1.01	1.54	1.00	1.01	1.54
1.00	10	1.00	1.01	1.31	1.00	1.01	1.31
1.00	15	1.00	1.01	1.30	1.00	1.01	1.30
1.00	25	1.00	1.01	1.33	1.00	1.01	1.33
1.00	50	1.00	1.02	1.31	1.00	1.02	1.31
2.00	5	1.00	1.00	1.00	1.00	1.00	1.00
2.00	8	1.00	1.00	1.07	1.00	1.00	1.07
2.00	10	1.00	1.00	1.08	1.00	1.00	1.08
2.00	15	1.00	1.00	1.00	1.00	1.00	1.00
2.00	25	1.00	1.00	1.09	1.00	1.00	1.09
2.00	50	1.00	1.00	1.11	1.00	1.00	1.09

Table 4.5: Minimum, average and maximum cost ratios for various values of ϵ, expressed as a fraction of MST cost (continued).

		$r(T_{BRBC})$			$r(SPT)$		
ϵ	$\|N\|$	Min	Ave	Max	Min	Ave	Max
0.10	5	0.44	0.82	1.00	0.44	0.81	1.00
0.10	8	0.43	0.74	1.00	0.43	0.74	1.00
0.10	10	0.38	0.71	1.00	0.38	0.70	1.00
0.10	15	0.27	0.65	1.00	0.27	0.65	1.00
0.10	25	0.34	0.64	0.94	0.34	0.63	0.93
0.50	5	0.57	0.90	1.00	0.47	0.85	1.00
0.50	8	0.48	0.74	0.99	0.46	0.69	0.99
0.50	10	0.42	0.81	1.00	0.37	0.75	1.00
0.50	15	0.44	0.72	0.99	0.42	0.69	0.99
0.50	25	0.33	0.66	0.97	0.31	0.63	0.97
1.00	5	0.66	0.95	1.00	0.56	0.83	1.00
1.00	8	0.56	0.84	1.00	0.44	0.74	0.95
1.00	10	0.51	0.81	1.00	0.50	0.73	1.00
1.00	15	0.44	0.73	1.00	0.30	0.66	0.97
1.00	25	0.32	0.66	0.99	0.31	0.61	0.93
2.00	5	0.84	0.99	1.00	0.50	0.78	1.00
2.00	8	0.47	0.93	1.00	0.40	0.72	0.94
2.00	10	0.51	0.86	1.00	0.38	0.69	1.00
2.00	15	0.48	0.86	1.00	0.35	0.69	1.00
2.00	25	0.36	0.74	1.00	0.23	0.60	1.00

Table 4.6: T_{BRBC} and SPT radius statistics for random nets, expressed as a fraction of the MST radius.

		$cost(T_{BRBC})$			$cost(SPT)$		
ϵ	$\|N\|$	Min	Ave	Max	Min	Ave	Max
0.10	5	1.00	1.25	1.84	1.00	1.30	1.96
0.10	8	1.03	1.35	1.99	1.03	1.41	1.99
0.10	10	1.00	1.39	1.96	1.05	1.45	2.25
0.10	15	1.20	1.53	2.66	1.20	1.60	2.71
0.10	25	1.25	1.57	2.03	1.25	1.66	2.16
0.50	5	1.00	1.15	1.60	1.00	1.25	2.04
0.50	8	1.00	1.22	1.66	1.02	1.37	1.94
0.50	10	1.00	1.23	1.57	1.02	1.45	2.05
0.50	15	1.11	1.29	1.53	1.13	1.54	1.94
0.50	25	1.17	1.34	1.73	1.32	1.60	2.14
1.00	5	1.00	1.03	1.30	1.00	1.22	1.90
1.00	8	1.00	1.11	1.31	1.12	1.37	1.96
1.00	10	1.00	1.13	1.47	1.03	1.45	1.96
1.00	15	1.00	1.19	1.41	1.14	1.61	2.28
1.00	25	1.11	1.25	1.43	1.37	1.71	2.38
2.00	5	1.00	1.01	1.15	1.00	1.30	2.03
2.00	8	1.00	1.05	1.22	1.03	1.48	2.06
2.00	10	1.00	1.07	1.23	1.10	1.50	2.28
2.00	15	1.00	1.08	1.19	1.18	1.49	1.95
2.00	25	1.01	1.12	1.27	1.25	1.68	2.31

Table 4.7: T_{BRBC} and SPT cost statistics for random nets, expressed as a fraction of the MST cost.

		$r(T_{BRBC\_S})$			$r(SPT)$		
ϵ	$\|N\|$	Min	Ave	Max	Min	Ave	Max
0.10	3	0.63	0.93	1.00	0.63	0.93	1.00
0.10	4	0.50	0.90	1.00	0.50	0.90	1.00
0.10	5	0.43	0.84	1.00	0.43	0.84	1.00
0.10	7	0.42	0.82	1.00	0.42	0.82	1.00
0.10	10	0.51	0.82	1.00	0.51	0.82	1.00
0.10	15	0.31	0.81	1.00	0.31	0.80	1.00
0.50	3	0.60	0.94	1.00	0.60	0.94	1.00
0.50	4	0.55	0.88	1.00	0.52	0.86	1.00
0.50	5	0.43	0.89	1.00	0.43	0.87	1.00
0.50	7	0.48	0.86	1.00	0.45	0.82	1.00
0.50	10	0.42	0.80	1.00	0.42	0.77	1.00
0.50	15	0.40	0.78	1.00	0.40	0.75	1.00
1.00	3	0.65	0.99	1.00	0.57	0.93	1.00
1.00	4	0.64	0.99	1.00	0.54	0.91	1.00
1.00	5	0.67	0.95	1.00	0.48	0.86	1.00
1.00	7	0.55	0.92	1.00	0.55	0.84	1.00
1.00	10	0.53	0.90	1.00	0.47	0.81	1.00
1.00	15	0.47	0.85	1.00	0.47	0.78	1.00
2.00	3	1.00	1.00	1.00	0.62	0.92	1.00
2.00	4	0.71	0.99	1.00	0.55	0.89	1.00
2.00	5	0.61	0.99	1.00	0.59	0.85	1.00
2.00	7	0.49	0.97	1.00	0.43	0.80	1.00
2.00	10	0.49	0.93	1.00	0.45	0.81	1.00
2.00	15	0.46	0.88	1.00	0.45	0.76	1.00

Table 4.8: T_{BRBC_S} and SPT radius statistics for random block designs, expressed as a fraction of the KMB tree radius.

		$cost(T_{BRBC\_S})$			$cost(SPT)$		
ϵ	$\|N\|$	Min	Ave	Max	Min	Ave	Max
0.10	3	0.91	1.12	1.42	0.91	1.12	1.42
0.10	4	0.96	1.14	1.69	0.96	1.14	1.69
0.10	5	0.99	1.17	1.51	0.99	1.18	1.57
0.10	7	0.99	1.14	1.45	0.99	1.15	1.47
0.10	10	1.00	1.22	1.58	1.00	1.22	1.58
0.10	15	1.02	1.21	1.53	1.02	1.22	1.53
0.50	3	0.89	1.09	1.57	1.00	1.14	1.61
0.50	4	0.98	1.11	1.43	1.00	1.16	1.51
0.50	5	0.97	1.15	1.68	0.97	1.23	1.61
0.50	7	0.96	1.11	1.41	0.96	1.20	1.61
0.50	10	0.98	1.17	1.50	1.00	1.27	1.58
0.50	15	0.96	1.15	1.41	0.96	1.19	1.51
1.00	3	0.89	1.02	1.27	0.89	1.14	1.72
1.00	4	0.97	1.02	1.19	1.00	1.15	1.71
1.00	5	1.00	1.09	1.38	1.00	1.23	1.87
1.00	7	1.00	1.09	1.37	1.00	1.21	1.58
1.00	10	0.96	1.10	1.32	1.01	1.22	1.69
1.00	15	0.98	1.11	1.30	0.97	1.21	1.71
2.00	3	1.00	1.00	1.00	1.00	1.13	1.51
2.00	4	0.92	1.01	1.26	0.92	1.15	1.59
2.00	5	1.00	1.02	1.23	1.00	1.19	1.64
2.00	7	0.95	1.03	1.22	0.97	1.22	1.59
2.00	10	1.00	1.05	1.25	1.02	1.26	1.75
2.00	15	0.99	1.06	1.21	1.06	1.25	1.49

Table 4.9: T_{BRBC_S} and SPT cost statistics for random block designs, expressed as a fraction of the KMB tree cost.

4.8 Remarks and Extensions

We have proposed a new bounded-radius minimum spanning tree formulation, with our discussion focusing on applications to VLSI global routing. An effective method, called BPRIM, based on an analog of Prim's minimum spanning tree construction was given. Furthermore, we have also proposed a provably good general algorithm for bounded-radius tree construction. This method is based on a routing tree construction which in some sense melds the notions of shortest paths tree and minimum spanning tree, and which achieves a solution having both cost and radius bounded by constant factors away from optimal.[6] Our approach readily extends to Steiner tree routing in arbitrary weighted graphs, where again the routing tree is only a small constant factor away from optimal in terms of both cost and radius. Extensive simulations confirm that our approach gives good performance: the results of Section 4.7 indeed exhibit a smooth tradeoff between the competing requirements of small radius and low tree cost.

Based on our methods for constructing bounded-radius trees, a global routing procedure may work as follows. We route all nets, one by one, according to their priorities. For each net, we construct a bounded-radius minimum spanning tree or bounded-radius minimum Steiner tree using the algorithms presented in Sections 4.4 and 4.5. The parameter ϵ is either given by the user or computed based on an estimation of the timing constraints for the net. As noted in Section 4.6, different values ϵ_i can be used within a single net to reflect varying timing constraints

[6] Recently, Hu et al. [HHK92] have developed an explicit tradeoff between the Dijkstra (SPT) and Prim (MST) constructions, based on the observation that the respective dynamic programming recurrences $\min_{j \notin T} \min_{k^* \in T} [l_k^* + d_{jk}]$ and $\min_{j \notin T} \min_{k^* \in T} [d_{jk}]$ are very similar. In particular, the SPT-MST tradeoff is achieved using the recurrence $\min_{j \notin T} \min_{k^* \in T} [c \cdot l_k^* + d_{jk}]$ with $0 \leq c \leq 1$, and this has been found to yield somewhat improved results over the BRBC method.

for distinct various input-output paths, or an even stronger constraint may be satisfied, where different pathlength bounds R_i are imposed for each terminal n_i (e.g., given the output from a timing-driven placement tool such as RITUAL [SCK91b]). The cost of each edge in the routing graph is a function of wirelengths, channel capacities, and the distribution of current channel densities. Following the routing of each net, we update the edge costs in the routing graph. After all nets are routed, we may compute the timing-critical paths and, if necessary, further reduce the interconnection delay by re-routing some critical nets based on more accurate distributed RC delay models.

Our algorithms readily extend to other norms and to alternate geometries (e.g., 45- or 30-60-90-degree routing regimes). There are several remaining open problems, such as the complexity of computing the minimum-cost bounded-radius spanning tree in the Manhattan plane, or the complexity of choosing an MST with minimum radius when the MST is not unique.

CHAPTER 5

Verifying Interconnections

5.1 Introduction

In the last three chapters, we have proposed methods for constructing tree topologies which optimize various objectives. In this chapter, we will focus on how to efficiently test a tree once it has been constructed. This is a very general problem that arises in, e.g., communication networks and distributed computation. However, we will couch our discussion within the framework of multi-chip module (MCM) substrate testing, a currently active area of study and motivating application for this work.

MCM technology has recently emerged as an economically viable means of packaging complex, high-performance systems [BMH89] [Dai91] [Her90] [Sha91] [TD91] [Web89]. A typical MCM consists of a substrate containing inter-chip wiring, upon which are mounted a number of bare die (see Figure 5.1). The increased use of multi-chip module packaging has highlighted several new and challenging CAD problems in such areas as layout, thermal reliability, and testing [GWR90] [Sha91]. Testing in particular presents one of the most persistent challenges of the MCM approach [BGS91] [Her90] [Web89]. It is desirable to discover defects in the MCM substrate as early as possible, since the cost of locating and fixing a system fault increases significantly with each successive stage

of the system manufacturing and marketing process. While the fully assembled MCM package can be tested using combinatorial IC testing techniques, the pre-assembly MCM substrate contains only disjoint wiring connections with no active devices; thus, the substrate cannot be tested using conventional methods.

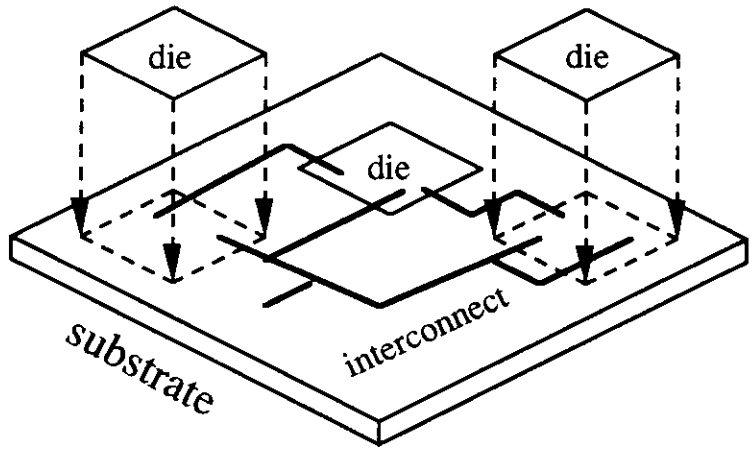


Figure 5.1: An example of a multi-chip module, showing the underlying substrate containing the interconnect, as well as several mounted dice.

Our model of MCM substrate interconnect is as follows. Each signal net is routed on multiple routing layers using a tree topology, where we assume without loss of generality that each leaf is a net terminal, each edge is a wire segment on a single wiring layer, and each internal node is a via between two or more routing layers (Figure 5.2). We wish to verify that the routing topology of each net is properly implemented, with no faults. For technological reasons detailed in [KRW91b] [KRW92], it is sufficient to test for so-called *open faults.*[1]

An open fault is an electrical disconnection between two points that are to be connected. As will be discussed in Section 5.2 below, two types of open faults

[1]Probes which are designed to detect open faults are also able to detect short faults and high-resistance faults by comparing the measured resistance and capacitance values to the expected ones.

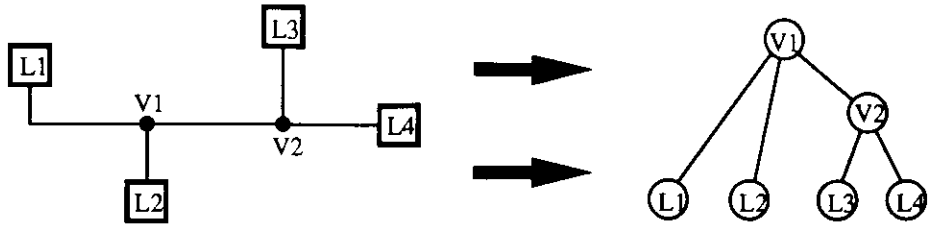


Figure 5.2: A sample net (left) and its corresponding tree representation (right); pins become leaf nodes while vias become internal nodes.

can arise in the routing tree: wire opens, which we term *edge faults*, and cracked vias, which we term *node faults*. Node faults correspond to a physical form of node failure that can arbitrarily disconnect subtrees and which is not necessarily detected when we test for edge faults.

In the manufacture of digital systems, traditional methods for connectivity checking involve either parallel probing of the circuit under test, or combinatorial exercising of the logic, neither of which can be used for MCM substrate testing [BMH89]. In verifying connectivity for PCBs, a bed-of-nails tester will simultaneously access every grid point, yielding an efficient, parallel checking procedure. However, this cannot be applied to MCMs since feature sizes are too small to allow use of such a grid-based methodology. A combinatorial testing approach such as the boundary-scan method for hierarchical design [Gro88] [WMM89] requires system-specific, built-in test circuitry. As noted above, combinatorial testing in general applies only to a completely assembled MCM; it is not applicable to substrates which contain isolated interconnect with no active circuit elements.

Given this failure of traditional methods, MCM substrate testing is currently accomplished on a net-by-net basis via a sequential net-by-net methodology. Several groups have used electron-beam methods to test MCM substrates [GWR90

but this can be prohibitively slow [SSB91]. All other sequential probing approaches involve variants of k-probe testing, where k "flying" probe heads simultaneously move around the circuit, measuring resistance and capacitance values to determine the existence of faults. Formally, we define a *k-probe* to be the simultaneous visitation of k distinct terminals by the k probe heads.[2] A single k-probe simultaneously verifies all paths between pairs of terminals in the probe set. In particular, when $k = 2$ the unique path between the two visited terminals is checked.

A sequential testing approach using 2-probes was developed by Crowell et al. [CKC84] [McW91], but could not guarantee detection of all open faults. Because the economics of system reliability dictate that complete fault coverage be achieved, Yao et al. [YCC91] recently proposed an algorithm that determines a minimum set of 2-probes which checks for all possible faults. In this chapter, we give a linear-time algorithm which for any $k \geq 2$ determines a k-probe set that accomplishes complete fault coverage of any net. The number of probes used by our method is the minimum possible.

Once probes are found which adequately test all nets for faults, the probes must be scheduled for execution on the test apparatus. Previous groups [CKC84] [YCC91] have used generic greedy or iterative traveling salesman heuristics to attack this problem. We propose two effective heuristics for probe scheduling based on new observations concerning the metricity and allowable structure of the probe set. Empirical results demonstrate reductions in testing cost of up to 21% over the best previous method [YCC91].

The remainder of this chapter is organized as follows. In Section 5.2, we for-

[2]Current probe technology generally uses $k = 2$, but probe machines with higher values of k are currently under development [Rus91].

mulate open fault detection as a tree testing problem, then present linear-time algorithms which find an optimal number of probes to cover all possible open faults. Section 5.3 shows that probe scheduling to minimize total travel time is a form of metric traveling salesman problem (TSP); we present two effective heuristics, one of which has small constant-factor error bound. Section 5.4 gives experimental results on both random and industry circuit benchmarks, and Section 5.5 concludes with directions for future research.

5.2 Fault Detection

In this section we address the case $k = 2$ of the following problem:

The Minimal Probe Generation (MPG) Problem: Given a tree with l leaves (i.e., pins), determine a minimum set of k-probes needed to verify that the tree contains no faults.

We consider two levels of fault coverage: (i) coverage of all edge faults, and (ii) coverage of all node faults in addition to all edge faults (see below). This section presents optimal solutions for the two corresponding versions of the MPG problem.

5.2.1 Optimal Detection of Edge Faults

In order to test the integrity of all tree edges, certainly every edge which is incident to a leaf must be tested. Thus, the number of leaves l in the routing topology induces a lower bound of $\lceil \frac{l}{2} \rceil$ probes when $k = 2$. Our probe generation algorithm PROBE1 orders the leaves of a tree as $p_1, \ldots, p_l$ using an arbitrary in-order traversal of the tree. Choosing the $\lfloor \frac{l}{2} \rfloor$ probes $\{p_i, p_{i+\lfloor \frac{l}{2} \rfloor}\}$, $1 \leq i \leq \lfloor \frac{l}{2} \rfloor$, will

cover all edges of the tree, as illustrated in Figure 5.3. If l is odd, an additional probe $\{p_1, p_l\}$ is generated. Figure 5.4 gives the formal algorithm statement.

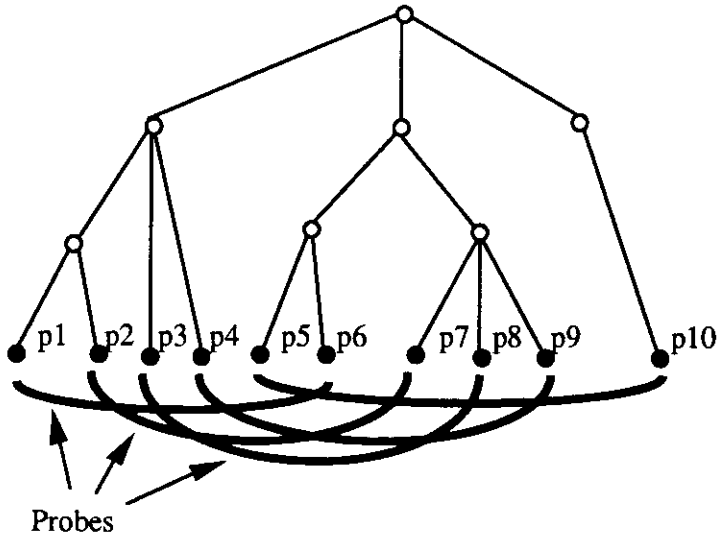


Figure 5.3: Selecting a minimal set of probes to detect the existence of any edge faults. The probes $\{p_i, p_{i+\lfloor \frac{l}{2} \rfloor}\}$, $1 \leq i \leq \lfloor \frac{l}{2} \rfloor$, provide complete edge fault coverage.

PROBE1: Optimal probe set generator for edge fault detection
Input: A tree with l leaves **Output:** A minimum set of probes for detecting all edge faults
Root the tree arbitrarily at an internal node **Induce** an in-order labeling $p_1, \ldots, p_l$ of the leaves **Output** the probes $\{p_i, p_{i+\lfloor \frac{l}{2} \rfloor}\}$, $1 \leq i \leq \lfloor \frac{l}{2} \rfloor$ **If** l is odd **Then Output** the probe $\{p_1, p_l\}$

Figure 5.4: PROBE1: Optimal detection of all edge faults.

Theorem 5.1 *Given a tree with l leaves, $\lceil \frac{l}{2} \rceil$ 2-probes are both necessary and sufficient for complete edge fault testing.*

Proof: A graph is *bridge connected* if every edge lies on some simple (i.e., vertex-disjoint) cycle. Starting with the original tree, for each probe we add into the graph an edge connecting the two leaves tested by the probe. A set of probes is sufficient to test for all edge faults if and only if it induces a bridge connected graph, since each edge will thus lie on some cycle and be tested by the probe that formed that cycle. In order to convert a tree into a bridge connected graph via the addition of a minimum number of new edges, it suffices to add the $\lceil \frac{l}{2} \rceil$ new edges $\{p_i, p_{i+\lfloor \frac{l}{2} \rfloor}\}$ for all $1 \leq i \leq \lfloor \frac{l}{2} \rfloor$, where $p_1, \ldots, p_l$ is the sequence of leaves in any in-order traversal of the tree (when l is odd, the additional edge $\{p_1, p_l\}$ is used as well).

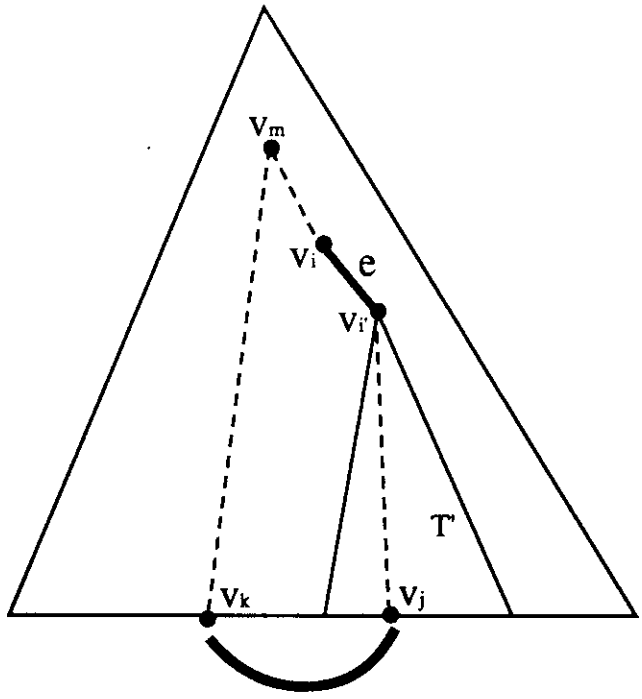


Figure 5.5: PROBE1 checks each edge $e = \{v_i, v_{i'}\}$ for a fault using a $\{v_j, v_k\}$ probe which forms a simple cycle containing e, as shown.

To see that every edge in the resulting graph $G = (V, E)$ lies on some simple cycle, observe that for every proper subtree in the original tree, there exists an edge in G connecting one of the leaves of that subtree to a leaf *not* in that subtree. Given an arbitrary edge $e = \{v_i, v_{i'}\}$ in the original tree (Figure 5.5), where v_i is the father of $v_{i'}$, one simple cycle that surely contains the edge e is $v_{i'}, \ldots, v_j, v_k, \ldots, v_m, \ldots, v_i, v_{i'}$ where (i) v_j is any leaf in the subtree T' rooted at $v_{i'}$ such that v_j is connected by a probe edge to a leaf v_k not in T', and (ii) v_m is the lowest common ancestor of both v_i and v_k. To prove that nodes v_j and v_k must exist, assume toward a contradiction that every leaf $v_{j'}$ in T' is connected by a probe edge to $v_{k'}$ which is also in T'. Because of the in-order labeling, leaves $v_{\lfloor \frac{l}{2} \rfloor}$ and $v_{\lfloor \frac{l}{2} \rfloor + 1}$ are also in T'. Our assumption that all probes involving leaves of T' are internal to T' then implies that v_1 must also be in T', along with v_l when l is even or v_{l-1} when l is odd. In the case where l is even, T' will contain all leaves of the input tree topology, contradicting the fact that T' is a proper subtree of the topology. For l odd, the probe which tests the sole leaf v_l not in T' must connect v_l to a leaf in T', contradicting the assumption that probes involving leaves in T' are internal to T'. □

5.2.2 Optimal Detection of Node Faults

In manufacturing the MCM substrate, a via can physically "crack" due to such factors as misalignment in lithography or thermal stress (see Figure 5.6) [YCC91]. In other words, subtrees rooted at an internal "cracked" node of the net can become electrically separated, so that certain sets of probes may detect this node fault, while other sets will fail to find the cracked node. This section gives a linear-time algorithm, called PROBE2, that finds a minimum probe set which

completely tests for both edge faults and node faults.

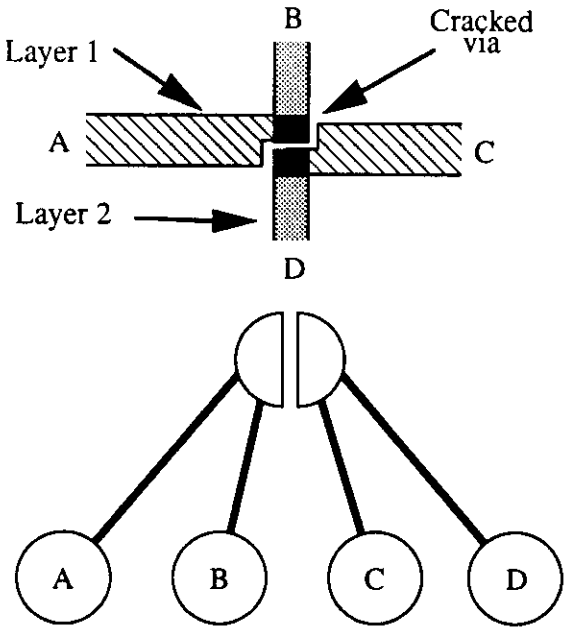


Figure 5.6: A cracked via, i.e., node fault, in a routing. Top: the two routing layers are depicted using different shadings, while the cracked via which disconnects the circuit is depicted in black. Bottom: the equivalent tree representation. Note that the two probes (A, B) and (C, D) do not detect the node fault.

PROBE2 begins by rooting the tree at an internal node R of maximum degree d and then orienting all edges towards R. The algorithm then continues with each leaf node sending to its parent a message list containing its label. When a given node has received message lists from all of its children, it iteratively generates probes by pairing labels from distinct incoming lists, at least one of which contains more than one node label; when the sum total of remaining labels at that node has been reduced to less than $d+1$, all remaining labels are concatenated and sent to the node's parent. This process is repeated at each node until only the root R remains unprocessed, where a simple cleanup step is then performed. Figure 5.7 traces the execution of PROBE2 on a small example, while Figure 5.8 gives the

formal statement of PROBE2.

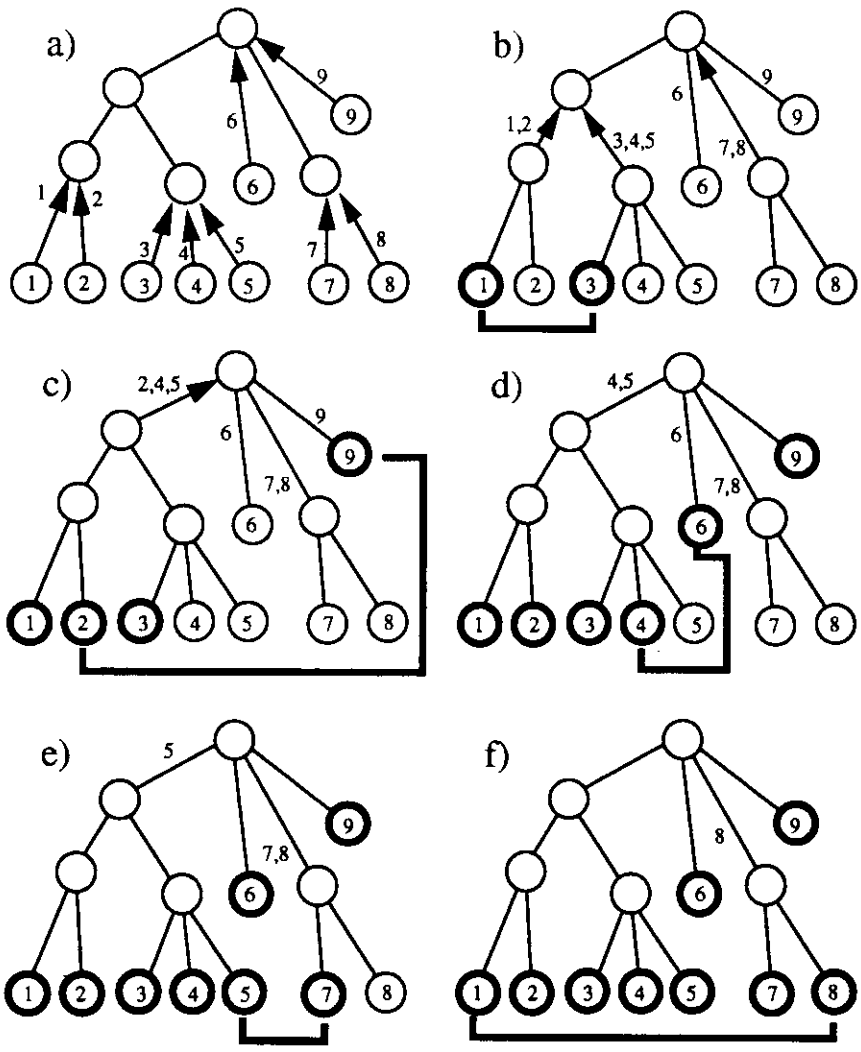


Figure 5.7: A sample run of PROBE2 on a tree containing 9 leaves and 5 internal nodes; a total of 5 probes are generated (thick arcs).

Theorem 5.2 *Given a tree, PROBE2 generates a set of probes which completely tests for all edge and node faults.*

Proof: By induction on the number of leaves in the tree.

Basis: Any tree of depth one is fully tested by PROBE2, since in that case we

PROBE2: Optimal probe set generator

Input: A tree (V, E)

Output: A minimal probe set (for edge faults and node faults)

Let $W = V$
Root tree at internal node $R \in W$ of max degree d
Direct all edges towards R
Each leaf node v sends the message $\{v\}$ to $parent(v)$
While $\exists v \in W$, $v \neq R$ having received messages $M_1, \ldots, M_{deg(v)-1}$
  While $\Sigma_{k=1}^{deg(v)-1} |M_k| > d$ and $\exists i \ \ni \ |M_i| \geq 2$
    Let $x \in M_i$
    Let $y \in M_j$ for some $j \neq i$, $|M_j| \geq 1$
    Generate probe $\{x, y\}$
    $M_i = M_i - \{x\}$; $M_j = M_j - \{y\}$
  Let $L = M_1 \cup \ldots \cup M_{deg(v)-1}$
  Send L to $parent(v)$
  $W = W - \{v\}$
/* now $W = \{R\}$ has received messages $M_1, \ldots, M_d$ from its children */
While $\exists i, j, 1 \leq i, j \leq d$, $i \neq j$ such that $|M_i| \geq 2$, $|M_j| \geq 1$
  Reorder $M_1, \ldots, M_d$ such that $|M_i| \geq |M_{i+1}|$ for all $1 \leq i < d$
  Let $k \leq d$ be as small as possible such that $|M_k| > 0$
  Let $x \in M_1$, $y \in M_k$
  Generate probe $\{x, y\}$
  $M_1 = M_1 - \{x\}$; $M_k = M_k - \{y\}$
Let $L = M_1 \cup \ldots \cup M_d$
If $|L| > 1$ **Then** generate probes $\{L_1, L_i\}$ $\forall$ $2 \leq i \leq |L|$ and terminate
  Else Choose $v \in V \ \ni \ v$, L_1 were not passed by same child
    Generate probe $\{ L_1, v\}$ and terminate

Figure 5.8: **PROBE2:** Optimal detection of edge and node faults.

test one leaf with all others.

Induction: Assuming that our algorithm completely tests any tree with k leaves, we show that it also completely tests any tree with $k+2$ leaves. Let T be a tree with $k+2$ leaves and let l_1 and l_2 be the first leaves paired together to generate a probe when we apply PROBE2 to T. Let T' be the tree with k leaves which results

when we remove l_1 and l_2 from T. Let v be the internal node whose message lists we are examining when we generate the probe $\{l_1, l_2\}$. After generating the probe $\{l_1, l_2\}$, the message lists at v are precisely those that arrive at v' (the node in T' that corresponds to v in T), and so the rest of the execution of PROBE2 on T is the same as the execution of PROBE2 on T'. Since v is the first node at which we generate a probe, it must be the case that just before l_1 and l_2 are matched, the message lists at v contain the names of all leaves which are descendants of v. When l_1 and l_2 are removed, the new lists contain the names of all descendants of v' and so PROBE2 proceeds as for the tree T' with k leaves. Clearly our probe sequence will test that l_1 and l_2 are connected, and the continuation of PROBE2 on T' will generate a probe sequence which tests that T' is connected. Therefore, we need only show that the connectivity between l_1 and T' is also tested. If v is not the parent of both l_1 and l_2, then some edge on the path from l_1 to l_2 in T coincides with an edge of T'. Since the mutual connectivity of all edges on this path is tested, by transitivity the connection of all of these edges to T' is also tested. Conversely, suppose v is the parent of both l_1 and l_2. Both l_1 and l_2 pass to v message lists of length one, but the only time we generate a probe from two singleton message lists is at the root when all message lists have length one, and this can occur only as part of the basis case.

Note that while our induction is on the number of leaves in the tree, every tree with more than one leaf can be built from the basis case. There is only one tree with two leaves: the tree with a root and two leaves at depth one. There are two possible trees with three leaves: the tree with a root and three leaves at depth one; and the tree with a root which has one leaf child at depth one, and one internal node at depth one, which in turn has two children which are leaves

at depth two. Of these three trees, only the last does not fit the basis case, but it will after one more probe is generated. Any other tree with two or three leaves will never be given as input to the algorithm since its root will not have maximum degree d, and will not arise during the induction since that would violate Lemma 5.9 below. □

We now use a sequence of lemmas to prove that PROBE2 uses the minimum possible number of probes.

Lemma 5.3 *A node of degree d requires $d-1$ probes to test whether it is cracked.*

Proof: Testing an internal node for "cracks" using 2-probes is equivalent to connecting a set of d vertices by edges until a single tree connects all the vertices. The result follows immediately since a tree with d vertices contains exactly $d-1$ edges. □

Lemma 5.4 *No node passes up more than d leaf names in its message list.*

Proof: By induction on the maximum distance from a node to any one of its leaves.

Basis: If the distance is zero then the node is a leaf and passes up one name in its message list.

Induction: Suppose the node does send up a message list containing more than d leaf names. Since by the induction hypothesis each child of the node sent up at most d leaf names, the propagated leaf names must come from the message lists of two or more children. But under such conditions, the algorithm will generate probes using leaf names from different message lists until either each child's message list has length one, or the total message count is d or less. In the

first case, since the node received messages from at most $d-1$ children, it can send up a message list of length at most $d-1$. In the second case, the node sends up a list of length at most d. □

Lemma 5.5 *No internal node passes up fewer than $d-1$ leaf names in its message list, unless the list contains the names of all leaves that are descendants of that node.*

Proof: By induction on the maximum distance from a node to any one of its leaves.

Basis: If the distance is zero then the node is a leaf and it passes up the message list containing its name.

Induction: Suppose a node passes up k leaf names where $k < d-1$, but that the node has t descendants where $t > k$. Since $k < d-1$, no probes could have been generated at that node, and k must be the sum of the lengths of all message lists sent by the node's children. Each of these children must have sent lists of length no greater than k, and so by the induction hypothesis, each must have sent lists containing all their leaf descendants. The union of these lists is precisely the list of the descendants of the node, and so we have a contradiction. □

Lemma 5.6 *If any combination of $2(d-1)$ or more leaf names are present among d non-empty message lists at the root node, and the difference in length between the longest two message lists is no more than $d-1$, then either all leaf names appear in the probe sequence exactly once, or else one leaf name appears twice and all others appear once.*

Proof: By induction on d.

Basis: $d = 2$. There are two message lists. If they both have the same length,

then they are completely matched with each other and no leaf names are repeated. If one list has length one more than the length of the other, then some leaf will need to appear a second time in order to match the single leaf remaining after all the matches have been made.

Induction: Matches at the root are made by the algorithm until one of the incoming message lists has length one. At this point there are at least $2(d-1)$ leaves distributed among the message lists. The next match leaves $d-1$ non-empty message lists. The difference in lengths between the two longest message lists has decreased by one unless these lengths were already equal. The total number of leaves remaining is at least $2(d-1-1)$ among $d-1$ lists. We then invoke the induction hypothesis for $d-1$. □

Lemma 5.7 *If any combination of* $2(d-1)-k$ *leaf names is present among* d *non-empty message lists at the root node, then exactly one of the leaf names generated in the probe sequence will appear* $k+1$ *times in the generated probe sequence, and every other leaf name will appear exactly once.*

Proof: By induction on d.

Basis: If $d=2$ and $k=0$ then we have two singleton lists, and we match them together without duplicating any leaves.

Induction: If there exists any message list with length greater than one, we match it with a list of length one and invoke the induction hypothesis. (There must be a list of length one, else we would have $2(d-1)$ or more leaf names present.) If no list has length greater than one, then $2(d-1)-k=d$, or $d=k+2$. We then must use one of the leaves $d-1=k+1$ times. □

Lemma 5.8 *If $k > 0$ and $2(d-1) - k$ leaf names arrive at the root, then there are exactly $2(d-1) - k$ leaves in the tree.*

Proof: Suppose that at least one child of the root sent up a message list of length $d-1$ or more. There are $d-1$ other children, each of which must have sent at least one leaf name. This would imply a total of at least $2(d-1)$ leaf names at the root, which is too many; hence, no child of the root sent up $d-1$ leaf names. Applying Lemma 5.5, we find that all leaf descendants of the root must appear in the message lists received by the root, and so there must be exactly $2(d-1) - k$ leaves in the tree. □

Lemma 5.9 *No child of the root is left with a message list of length greater than one when all other lists have reached length zero.*

Proof: Suppose that when we reach the PROBE2 step "Let $L = M_1 \cup \ldots \cup M_d$" we have $V = \{R\}$ and that one of the M_i's has length $|M_i| > 1$. There must be only one such M_i, else we would have continued to generate probes. Consider the last $d-1$ probes generated: each must have taken one element from M_i, and M_i must always have been the maximum-length message list at the root since no other M_j has length within one of M_i. Hence, M_i must have started with length $d+1$, but Lemma 5.4 guarantees that this is impossible. □

Theorem 5.10 *PROBE2 generates the minimum number of probes which achieves complete edge and node fault testing.*

Proof: Let l be the number of leaves in the tree. If $2(d-1)$ or more leaf names arrive at the root, then by Lemma 5.6, we have either l or $l+1$ leaf names used in the sequence of probes; this will be optimal since every leaf name must appear

in at least one probe, or else not all edges would have been tested. If $2(d-1)-k$ leaf names arrive at the root, then by Lemma 5.8 we have $l = 2(d-1)-k$, and by Lemma 5.7 the sequence of probes contains $l+k = 2(d-1)$ leaf names. There are then $d-1$ probes, which is optimal by Lemma 5.3. □

Except at the root, each probe generated by PROBE2 will remove two distinct leaf names from the messages (i.e., lists of leaf names) being passed. At most d leaf names will remain to be processed at the root, requiring at most $d-1$ additional probes. Therefore, to test an l-pin net the number of probes that our algorithm generates is bounded by $\frac{l-d}{2} + (d-1) = \frac{l}{2} + \frac{d}{2} - 1$; this bound is optimal and is tight for a star topology (which is theoretically possible given the multi-layer interconnect). Each node v passes no more than d leaf names up to its parent, and thus each node will receive fewer than d^2 leaf names from its children. Assuming that d is a constant dependent on technology, the amount of processing at each node is a constant, and since each node is processed only once, the overall time complexity of PROBE2 is linear in the size of the tree, which is clearly optimal.

5.3 Efficient Probe Scheduling

As noted above, efficient probe scheduling algorithms are necessary because testing cost is largely dependent on the total travel time of the probe heads. Typically, individual stepper motors will control the $x-$ and $y-$coordinates of each moving probe head, and we say that the distance traveled by the i^{th} probe head is given by

$$dist(A_i, B_i) = max[\ |A_{i_x} - B_{i_x}|\ ,\ |A_{i_y} - B_{i_y}|\ ].$$

This distance function, which is simply the Chebyshev or L_∞ norm, reflects the fact that the maximum time interval for which any motor is engaged will determine the delay between consecutive probes; this distance measure is typical in manufacturing applications and is quite accurate despite second-order effects such as acceleration and deceleration of the moving heads. For k-probes with $k = 2$, the *cost* of moving the probe heads from a set of pin locations $A = \{A_1, A_2\}$ to another set of locations $B = \{B_1, B_2\}$ is given by

$$c(A, B) = \min\{\max[dist(A_1, B_1), dist(A_2, B_2)], \max[dist(A_1, B_2), dist(A_2, B_1)]\}$$

More generally, for $k > 2$ the cost of moving the probe heads from $A = \{A_1, \ldots, A_k\}$ to $B = \{B_1, \ldots, B_k\}$ is given by

$$c(A, B) = \min_{\{\sigma\}} \; max[dist(A_1, B_{\sigma(1)}) \; , \; dist(A_2, B_{\sigma(2)}) \; , \; \ldots \; , \; dist(A_k, B_{\sigma(k)})]$$

where $\{\sigma\}$ denotes the set of all permutations of the probe indices $\{1, \ldots, k\}$. In other words, we choose the mapping of A onto B in such a way that the maximum travel time of any probe head is minimized (see Figure 5.9). We refer to this as the *generalized* distance function.

In some technologies, each probe head may be carried by its own moving horizontal bar [YCC91]. If the probing apparatus is constructed so that all such probe carriers lie in the same plane, collisions between probes become a concern in the sense that no two bars can cross each other's path. In other words, the y coordinates of the k probe heads must satisfy $y_1 \leq y_2 \leq \ldots \leq y_k$ at all times. Thus, the probe head coordinates are always sorted lexicographically [YCC91]. This constraint clearly yields a different distance function, which we call the *collision-free* distance. The collision-free distance function is more restrictive than the generalized distance function, since only one permutation of the probe heads

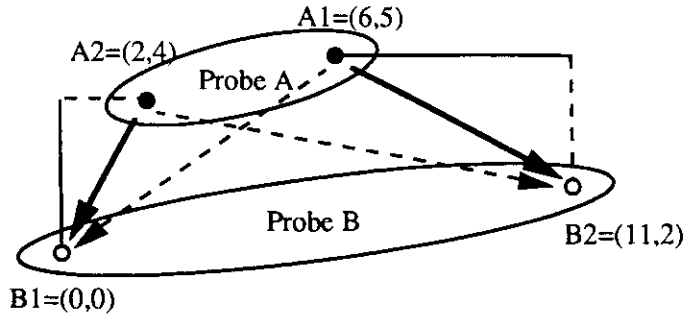


Figure 5.9: An example illustrating the distance function between the two probes $A = \{A1, A2\}$, and $B = \{B1, B2\}$. We have $dist(A1, B1) = 6$, $dist(A2, B2) = 9$, $dist(A2, B1) = 4$, and $dist(A1, B2) = 5$; thus, the distance between the two probes A and B is $\min(\max(6, 9), \max(4, 5)) = \min(9, 5) = 5$ (i.e., the best strategy will move one probe head from $A2$ to $B1$ while the other probe head moves from $A1$ to $B2$).

is feasible in traveling from one set of locations to another. In particular, for $k = 2$ the collision-free cost of moving the probe heads from $A = \{(x_1, y_1), (x_2, y_2)\}$ to $B = \{(x_3, y_3), (x_4, y_4)\}$, where $y_1 < y_2$ and $y_3 < y_4$, is given by $\max\{\ |x_1 - x_3|\ ,\ |y_1 - y_3|\ ,\ |x_2 - x_4|\ ,\ |y_2 - y_4|\}$.

The problem of efficient probe scheduling is stated as follows:

The Minimal k-Probe Scheduling (k-MPS) Problem: Given a set of k-probes, minimize the total probe moving cost required in executing all probes.

A reduction from the geometric traveling salesman problem [GJ79] yields:

Theorem 5.11 *The k-MPS problem is NP-hard.*

Proof: We can transform a geometric instance of TSP into an instance of k-MPS by introducing k copies of each site, then considering each set of k identical copies of a site as a single k-probe. Distances between probes will correspond to the original distances between the corresponding sites in the TSP instance. □

The probe scheduling problem seems quite unapproachable, both due to its theoretical intractability and because the distance and travel cost functions are not easily intuited. Thus, previous work relies on generic traveling salesman heuristics to optimize the probe schedule. For example, when $k = 2$ probe heads are available, Crowell et al. [CKC84] use a bandsort algorithm to optimize the movement of *one* of the probe heads. Of course, the other probe head may be forced to travel very large distances between probes, and indeed the resulting schedule is often exceedingly inefficient. Yao et al. [YCC91] use simulated annealing and the Kernighan-Lin 2-*opt* operator [Lin65] as the basis of an iterative interchange heuristic; their probe schedules save up to 83% of travel costs over the method of [CKC84]. Note that all of the heuristics proposed in [CKC84] and [YCC91] have unbounded error.

In this section, we first show that the k-probe travel costs are actually metric (although clearly not geometric), i.e., distances between k-probes satisfy the triangle inequality for all values of $k \geq 1$. As a consequence, traveling salesman heuristics with constant-factor error bound may be applied [PS82]. Second, we exploit flexibility in the choice of probes to find probe sets which can coexist in an efficient probe schedule.

5.3.1 Metricity of the k-MPS Problem

With the collision-free distance function, the travel costs of the probe heads are easily seen to satisfy the triangle inequality, since the probe head coordinates are always in lexicographic order. Thus, moving the probe heads from A to C via an intermediary B yields the same final probe permutation as would result by moving directly from A to C. Metricity follows from the metricity of the

Chebyshev norm.

For arbitrary k-probes A, B and C, we may view the generalized travel costs $c(A,B)$, $c(B,C)$ and $c(A,C)$ as being respectively determined by the optimal (minimum-cost) permutations $\sigma_1 : A \rightarrow B$, $\sigma_2 : B \rightarrow C$ and $\sigma_3 : A \rightarrow C$. Comparing the composed permutation $\sigma_1 \circ \sigma_2 : A \rightarrow C$ with the permutation $\sigma_3 : A \rightarrow C$ yields the following:

Theorem 5.12 *For any three k-probes A, B and C, the generalized travel costs $c(A,B)$, $c(B,C)$ and $c(A,C)$ satisfy the triangle inequality, i.e., $c(A,B)+c(B,C) \geq c(A,C)$.*

Proof: Compare the permutation $\sigma_3 : A \rightarrow C$ that defines $c(A,C)$, with the induced permutation $\sigma_1 \circ \sigma_2 : A \rightarrow C$ (see Figure 5.10). Define $\max(\sigma)$ to be the maximum distance traveled by any probe head according to the permutation σ. Clearly $c(A,C) \leq \max(\sigma_1 \circ \sigma_2)$, since $\sigma_1 \circ \sigma_2$ is not necessarily the minimum-cost permutation between A and C. On the other hand, $\max(\sigma_1 \circ \sigma_2) \leq c(A,B) + c(B,C)$ by the triangle inequality and the metricity of the Chebyshev norm. It follows that $c(A,C) \leq c(A,B) + c(B,C)$. □

Theorem 5.12 allows us to apply heuristics which achieve bounded error for *metric* TSP instances. In particular, invoking Christofides' combination of a minimum spanning tree construction and matching [PS82] yields:

Corollary 5.13 *Given a fixed set of n k-probes, for any $k \geq 1$, a heuristic probe schedule with cost at most $\frac{3}{2}$ times optimal can be found in $O(n^3)$ time.*

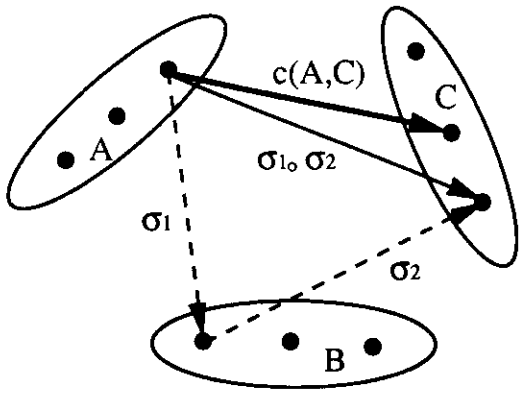


Figure 5.10: Metricity of the probe travel cost function.

5.3.2 Varying the Probe Set

A further optimization of the tour schedule is possible because the set of probes is itself variable. Figure 5.11 depicts an instance where a "smarter" choice of probes reduces the optimal tour cost by one-quarter. Most tree topologies can be tested with the minimum number of probes in many distinct ways. For example, each 3-pin net in Figure 5.11 can be tested by a minimal set of 2-probes in three distinct ways (i.e., any two probes can be used); in fact, the 2-pin net is the only connection topology with a unique minimum probe set. For special nets such as power and ground nets, the usual MCM architecture allows even more freedom: such nets can be viewed as being implemented by vias to dedicated routing planes, so that *any* partition of the pins into sets of cardinality k will cover all edge faults.

We thus obtain a new type of *compatibility TSP* problem, where a set of k-probes is selected to cover each net such that the optimal tour cost for the *union* of all probe sets is minimized. Such a formulation, where there is a synergy between the choice of probes and the optimal tour cost, seems to be new in the

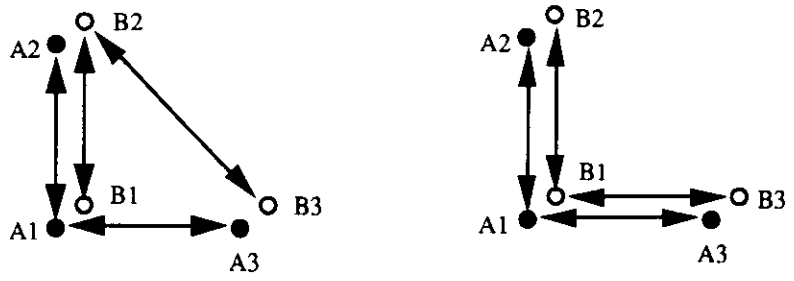


Figure 5.11: An example showing how careful probe selection can reduce the total tour length by as much as one-quarter: four probes are required for complete edge fault coverage in the two 3-pin nets $\{A_1 = (0,0),\ A_2 = (0,1),\ A_3 = (1,0)\}$ and $\{B_1 = (\epsilon,\epsilon),\ B_2 = (\epsilon,1+\epsilon),\ B_3 = (1+\epsilon,\epsilon)\}$. Assuming that the probe tour must start and end at the origin, the probe set on the left will be optimally ordered as $\{(A_1,A_2)\ ,\ (B_1,B_2)\ ,\ (B_2,B_3)\ ,\ (A_1,A_3)\}$, requiring about four units of travel time. The probe set on the right may be ordered as $\{(A_1,A_2)\ ,\ (B_1,B_2)\ ,\ (B_1,B_3)\ ,\ (A_1,A_3)\}$, requiring only about three units of travel time.

literature and is of independent interest.

The Minimal Probe Generation/Scheduling (MPG/S) Problem: Given a tree, determine and schedule a set of k-probes so that the total probe moving cost is minimized.

In order to hybridize the probe-generation phase with the tour-scheduling phase to take advantage of the non-uniqueness of minimum probe sets, we propose the heuristic PROBE3 (Figure 5.12). PROBE3 is based on a minimum-cost insertion strategy [Law85], i.e., it schedules all probes for a small subset S of nets, then iteratively adds the probe which has lowest insertion cost in the tour while still allowing a minimum probe set.[3] In executing PROBE3, we typically choose S to be all nets other than power, ground, and 3-pin nets. That is,

[3]Note that a probe set which allows us to minimize travel cost may have more than the minimum possible number of probes. However, the heuristics discussed in this section require that the number of probes is minimum; the more general optimization is open.

we specifically exploit the high degree of freedom in choosing probes for power, ground and 3-pin nets. As seen in the next section, PROBE3 yields significantly shorter schedules than existing methods.

PROBE3: Insertion-based method for probe selection
Input: A collection N of trees **Output:** A probe schedule which tests all edge and node faults
Compute minimal probe set P verifying a subset $S \subset N$ of the trees **Compute** a heuristic schedule (tour) $P_1, \ldots, P_m, P_1$ of P **While** $\exists$ a tree not having complete fault coverage **Find** a probe P^* for any such tree N_i subject to (i) N_i still coverable by minimum number of probes after P^* is added (ii) the probe's minimum insertion cost between consecutive probes is minimized, i.e., $\min_{feasible\ P^*} \min_{i} \{c(P_i, P^*) + c(P^*, P_{i+1}) - c(P_i, P_{i+1})\}$ **Insert** P^* into the tour between probes P_i and P_{i+1}, where i is tour index at which P^* had minimum insertion cost

Figure 5.12: **PROBE3:** An insertion-based heuristic for probe selection.

5.4 Experimental Results

We tested our algorithms on an MCM benchmark design obtained from Hughes Aircraft Co., containing 44 components and 199 nets. This is the same benchmark used by Yao et al. in [YCC91]. We also used two randomized versions of the Hughes benchmark, where the same net topologies were retained but pin coordinates were reassigned randomly from a uniform distribution in the layout region. PROBE2 was used to generate minimal probe sets which cover all possible edge and node faults. The schedules for these probe sets were optimized using the 2-opt TSP heuristic, as well as by 2-opt followed by 3-opt (in a separate run).

We also tested a variant of PROBE3 on the same benchmark, as described above. We first generated a minimal set of probes for all nets other than power, ground, and nets with three or fewer pins, then computed a heuristic tour for these probes, using the 2-opt TSP heuristic (again, in a separate run, we used 2-opt followed by 3-opt). Finally, for the remaining nets, we iteratively added additional probes which (i) could be inserted into the current tour with minimum cost, and (ii) were compatible with previously chosen probes in some minimum probe set. In all cases, a total of 634 probes were generated by our algorithm, the same number as that generated by the algorithm of [YCC91]. With each of the PROBE3 experiments, 226 probes were initially chosen to cover the nets which had > 3 pins and which were neither power nor ground; the remaining 408 probes were added incrementally. In the PROBE3 experiments, we optionally ran 2-opt improvement after every 10 probes added, and optionally ran 3-opt improvement after every 50 probes added. All of the above benchmarks were run with both the collision-free and generalized distance functions. These results are summarized in Table 5.1.

As expected, the PROBE3 variants, being able to carefully choose probes while constructing the heuristic tour, outperformed PROBE2 by a considerable margin. For the benchmark design, the best tour obtained in [YCC91] using simulated annealing had cost 150,525,000; in comparison, our PROBE3 variants obtain up to 21% improvement over this result. Since simulated annealing usually gives solutions quite close to optimal [JAM89], our results indeed confirm that careful choice of compatible probes is an important issue.[4]

[4]Recently, Chou et al. [CCR92] have used the same compatibility-TSP idea, along with simulated annealing, to achieve an additional 9% reduction in probing cost.

MCM	metric	PROBE2 + 2-Opt	PROBE2 + 2-Opt + 3-Opt	PROBE3 + 2-Opt	PROBE3 + 2-Opt + 3-Opt
Hughes	general	160,435,000	153,185,000	126,210,000	118,497,000
	coll-free	163,202,000	157,600,000	131,010,000	126,637,500
Rnd1	general	294,164,000	286,679,000	265,276,000	257,838,000
	coll-free	302,684,000	289,843,000	269,346,000	260,897,000
Rnd2	general	295,956,000	285,379,000	271,869,000	260,150,000
	coll-free	304,885,000	294,421,000	270,767,000	263,113,000

Table 5.1: Performance of PROBE2 and PROBE3 variants on the industry benchmark and on random examples, using both the collision-free and generalized distance functions. Note that the best probe schedule cost obtained by Yao et al. for the industry benchmark, using simulated annealing, was 150,525,000 units. The tour obtained by PROBE3 + 2-opt + 3-opt gives savings of 21% over this value.

5.5 Remarks and Extensions

We have examined the problem of verifying interconnection trees, with particular emphasis on applications to multi-chip module substrate testing. We have formulated tree testing as a problem of connectivity verification using k-probes, and presented linear-time algorithms for optimal probe generation. Our algorithms yield minimum probe sets which cover all possible edge and node faults. Since the associated probe scheduling problem is metric, a scheduling heuristic with constant-factor error-bound can be obtained. Furthermore, we presented an insertion-based heuristic which addresses a "compatibility TSP" formulation in exploiting synergy between the choice of probes and the associated scheduling problem. This heuristic significantly improves probing costs over previous methods.

There are a number of interesting open problems. The fact that many different

probe sets can test a given tree yields an interesting TSP variant, as noted above. It is possible that a "prize-collecting salesman" formulation (e.g., at least two of the three possible probes must be "collected" for each 3-pin net) can be solved with constant-factor error via an LP-relaxation scheme. This would be quite useful, as the bounded-error heuristic of Corollary 5.13 in Section 5.3 applies only when all of the probes have been fixed. Analyzing the maximum error inherent in arbitrarily fixing a probe set is also of interest.

Developments in probe technology will soon allow $k > 2$ probe heads to move simultaneously, affording even greater freedom in choosing the probe sets. Thus, the synergy between choice of probes and the resulting optimal schedule cost will continue to be of significance. More sophisticated strategies for the efficient insertion of probes into a partial tour are possible; for example, we may look for the best combination of probe additions with probe deletions, then iterate this tour improvement until no further cost reduction is possible. Finally, the concept of verifying connectivity by checking paths, rather than edges, is quite novel, as is the "physical" model of node failure; these notions can be applied to both trees and arbitrary graphs which arise in other fields of study.

CHAPTER 6

Prescribed-Width Routing

6.1 Introduction

Thus far, our connectivity formulations have implicitly assumed that interconnections (i.e., edges and paths) have zero width. However, we often require a path to have a prescribed minimum width. For example, printed circuit board designs may require varying minimum interconnection widths either because of high current loads or because several wires (e.g., of a bus) must be routed parallel to each other [McL90].

The problem of prescribed-width routing also arises in robotics, particularly for kinematic path planning (as distinguished from dynamic planning; see [Lat91] for a survey of robot motion planning) subject to two very practical requirements: (i) the need to incorporate uncertainty into the formulation, and (ii) the need to construct an error-tolerant solution. These extensions respectively yield the notion of a general cost function in a given environment, and the notion of a prescribed path width that can accommodate the physical size of an agent. We note that existing path planning approaches [ABF88] [Can88] [Con90] [Lat91] [MMP87] [OY83] [Pap85] [SSH87] are not applicable since they make an implicit zero-width assumption, or else do not allow arbitrarily costed environments.

In this chapter, we formulate and solve the problem of finding minimum-cost

source-destination paths having prescribed widths. Our main contribution is a polynomial-time algorithm for prescribed-width routing in arbitrarily costed environments. Essentially, we discard the usual mix of shortest-path algorithms and graph search techniques, and instead employ a more general combinatorial approach. We use network flow algorithms to exploit the duality between connecting paths and separating sets: a minimum-cost *path* which *connects* two locations is equivalent to a minimum-cost *cut-set* which *separates* two other locations. Experimental results confirm the viability of our approach, and our method generalizes to solve the classical Plateau problem on minimal surfaces [Tsu86], a result of independent interest [HKR92b].

The remainder of this chapter is organized as follows. In Section 6.2, we formalize the prescribed-width routing problem for arbitrarily costed regions. Section 6.2 also develops our solution for prescribed-width routing via network flows, and describes an efficient implementation. Section 6.3 gives experimental results showing optimal prescribed-width paths in environments ranging from random cost maps to the more traditional class of maps with solid polygonal obstacles. Section 6.4 extends our approach to solve the classical Plateau problem. We conclude in Section 6.5 with several directions for future research.

6.2 Prescribed-Width Routing by Network Flows

We begin by establishing notation and terminology. Our development focuses on the duality between connection and separation which motivates the network flow approach.

6.2.1 Problem Formulation

We say that a subset R of the plane is *simply connected* if it is homeomorphic to a disk (i.e., R contains no holes). We define a *region* to be a simply-connected compact subset of $\Re^2$. By the Jordan curve theorem [CR41], the boundary B of a region 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: Given a region R with a 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) P itself. This is illustrated in Figure 6.1. 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. More precisely, 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, the goal of this chapter is to optimally solve the path planning problem when two practical constraints are incorporated: an *arbitrary* cost function is defined over the region, and a prescribed path width is given. The capability to deal with a general environment is quite powerful, particularly for

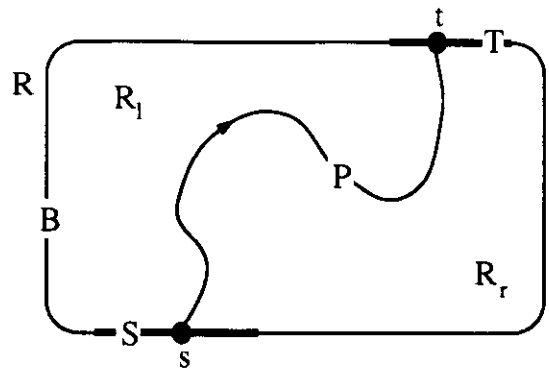


Figure 6.1: A path P between two points $s \in S$ and $t \in T$, where S and T are disjoint subsets of the boundary B of a region R.

robotics. For example, consider the following scenario: given probabilistic information about environment cost/difficulty (e.g., rivers, swamps, fields of landmines) and the locations of hostile entities, an agent wishes to travel with minimum probability of incurring a casualty or other damage. In this scenario, each point in the region will have an associated *weight*, or cost of traversal, corresponding to the level of danger. This formulation subsumes the traditional binary[1] cost function of an environment with solid polygonal obstacles and free space.

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* 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 be tempted to suppose that Dijkstra's shortest paths algorithm [CLR90] would provide a natural solution. However, application of Dijkstra's algorithm relies on an implicit assumption that the solution is an *ideal* path, i.e., a path of zero width. This caveat becomes clear

[1] i.e., the range of the function is either 0 (free space) or 1 (solid obstacle).

as we consider our second extension to the basic formulation – the requirement of a prescribed-width solution.

For robotics applications, the prescribed-width formulation is motivated by the inability due to errors in location, orientation and motion of the agent to exactly follow a given path. A path is error-tolerant if the agent will not come to harm when it makes a small deviation from the path; in the above scenario, a path which requires the agent to tiptoe precisely between landmines is not as error-tolerant as a path which avoids minefields altogether. We achieve error-tolerance by imposing a prescribed minimum-width constraint on the path. This prescribed-width formulation also addresses the fact that robot agents are physical entities with physical dimensions: because physical agents do not occupy only point locations in the environment, the minimum-cost path of zero width that is computed by a shortest-path algorithm may be infeasible. Note that in general, the optimum path for an agent of width d_1 cannot be obtained by simply widening or narrowing the optimum path for an agent of width d_2.

We now establish the relationship between a prescribed-width path requirement and the concept of *d-separation* [GHY74]. In what follows, 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: 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 ball of diameter d centered about a point of P.

Just as the path P between S and T partitions 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 point set $\hat{P}$ itself. We now obtain the definition of a d-separating path (see Figure 6.2):

Definition: 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 $\bar{R}_r$.

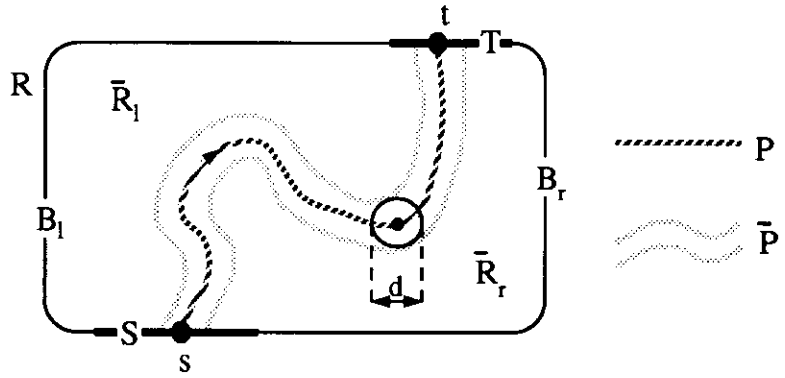


Figure 6.2: A d-separating path $\bar{P}$ of width d between two points $s \in S$ and $t \in T$ of the boundary of a region R. Here $\bar{R}_l$ is separated from $\bar{R}_r$ by a distance of d.

A d-separating path $\bar{P}$ between S and T is *minimal* 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. Given a specific width d, the prescribed-width routing problem can now be stated as follows:

The Prescribed-Width Routing Problem (PWR): Given an arbitrary region R with boundary B, weight function $w : R \rightarrow \Re^+$, a source $S \subseteq B$, a

destination $T \subseteq B$, and a width d, find a d-separating path $\bar{P} \subseteq R$ between S and T which has minimum cost.

While our formulation specifies an arbitrary weight function that is integrated to yield path cost, in most practical situations the region is discretized relative to a given fixed grid or sampling granularity (see, e.g., [BL91]). Thus, in the present work we will further assume a fixed-grid representation $\tilde{R}$ of the environment region R. With this discrete PWR formulation, 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: Given a region R, a *discrete* d-separating path $\tilde{P}$ in the gridded region $\tilde{R}$ is the set of gridpoints of $\tilde{R}$ which are contained in some d-separating path $\bar{P}$ in R (Figure 6.3).

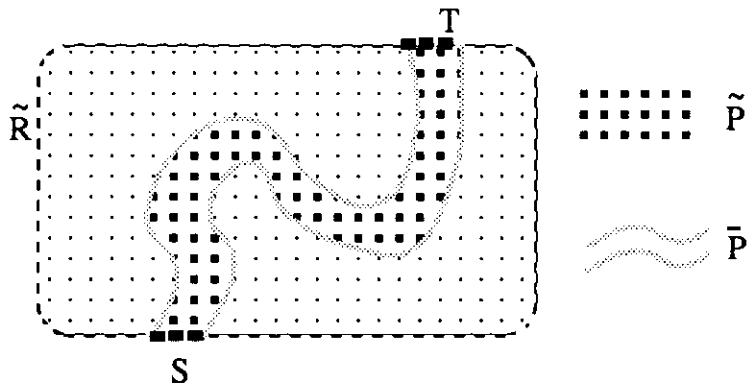


Figure 6.3: A discretized representation $\tilde{R}$ of a region R, and a discrete d-separating path $\tilde{P}$ in $\tilde{R}$. Note that $\tilde{P}$ is the set of lattice points covered by the continuous d-separating path $\bar{P}$ in R.

As before, a discrete d-separating path is *minimal* if no subset of it satisfies this definition. Analogously to the continuous case, a d-separating path partitions the gridded environment 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:

The Prescribed-Width Routing Problem in a Grid (PWRG): 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.

Intuitively, as the granularity quantum approaches zero, the solution for the PWRG instance will converge to the solution for the corresponding continuous PWR instance. At this point, we observe that although PWRG is a very natural problem formulation, it cannot be efficiently solved by traditional methods. In particular, efficient graph algorithms such as Dijkstra's shortest-path algorithm fail because the optimal d-separating path may self-intersect: when the shortest-path algorithm attempts to increment a path, it cannot determine how much of the increment should be added to the path cost (see Figure 6.4).[2]

In the special case where the cost function is binary, Dijkstra's algorithm is applicable using the well-known technique of augmenting the environment by growing each obstacle (as well as the region boundary) isotropically by $\frac{d}{2}$ units [Lat91]. The weight of each node in the free area is then set to some constant, while the weight of any node in an area covered by an obstacle is set to infinity. A minimum-cost zero-width path in such an augmented environment corresponds to

[2]Recall that in an n-node arc-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 PWRG formulation above assumes a node-weighted G, we may easily obtain an arc-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, as shown in Figure 6.4 [Hu69].

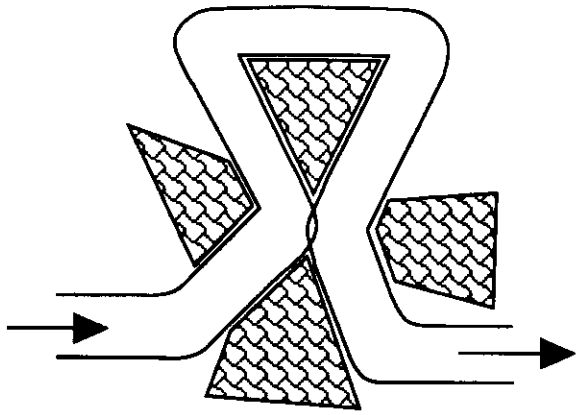


Figure 6.4: An optimal path with non-zero width can intersect itself (in this example the center gap is too narrow to allow the path to pass through it); efficient shortest-path algorithms cannot solve prescribed-width routing for such instances.

the center P of the d-separating path $\bar{P}$ that we seek (Figure 6.2). Unfortunately, this simple transformation fails when the formulation involves an arbitrary weight function.

Recall that our path cost model charges only once for each node covered by the path, reflecting the original motivation of minimizing the "path integral" of cost. Arguably, this is not unreasonable. For example, once a minefield has been cleared, the hazard associated with a second visit is zero. Similarly, when a path is plowed through snow or when a road is paved, the work is proportional to the actual area plowed or paved. Given this path cost formulation, Dijkstra's algorithm cannot be applied since it requires fixed edge costs in the underlying graph. Figure 6.5 illustrates this, where we let the cost of a grid edge (y, z) be $\hat{w}(Z - Z \cap Y)$, with Y and Z being the sets of gridpoints in $ball(y, d)$ and $ball(z, d)$ respectively. The weight function w is naturally extended to sets of nodes, e.g., $\hat{w}(Z) = \sum_{z \in Z} w(z)$. Unfortunately, this cost definition can still charge more than once for a single visited region, as indicated in Figure 6.5.

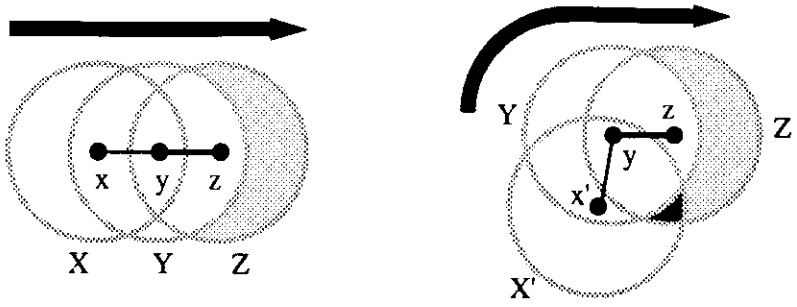


Figure 6.5: Dijkstra's algorithm fails for the general prescribed-width routing problem because the cost of an edge $e = (y, z)$ may depend on *many* previous nodes (e.g., x') on the path: in this example (right), the cost of incrementing the path by adding the edge e should be $\hat{w}(Z-(Z\cap(Y\cup X')))$ rather than $\hat{w}(Z - (Z \cap Y))$, otherwise the cost of the small dark region (right) will be charged twice.

We now use a network flow approach to develop an efficient, optimal algorithm for the PWRG problem.

6.2.2 A Network Flow Based Approach

Before describing our method in detail, we review several key concepts from the theory of network flows [FF61] [Law76]. 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 $s \in N$ and a distinguished sink $t \in N$; 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 $n_i \in N$. 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

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 [FF61]:

Theorem 6.1 *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 our initial observation 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

whose capacities correspond to costs in R. The remainder of this section describes how we accomplish this.

To transform prescribed-width routing in a region R into network flow, we first superimpose a mesh network topology over R, then assign node weights in this network according to the weighting function w. This yields a representation that is essentially equivalent to the underlying PWRG instance.

We guarantee a prescribed-width solution, by ensuring that any separating node set in the mesh topology will satisfy the width-d requirement. To this end, define the d-neighborhood of a node v in the mesh to be the set of all nodes that are at distance of d or less units away from v, and 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 can be a d-separating set. An illustration of this construction for $d = 2$ is given in Figure 6.6. The concept of a d-neighborhood was first investigated by Gomory and Hu [Hu69].

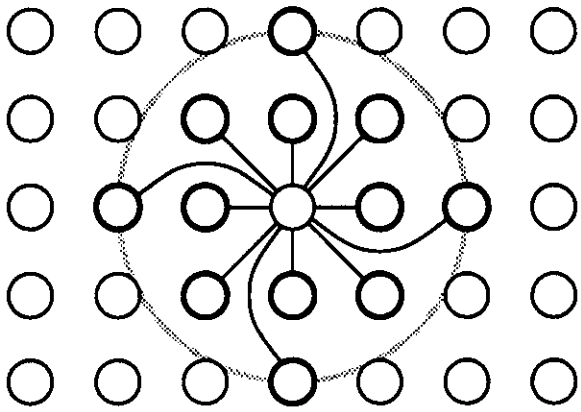


Figure 6.6: A node and its d-neighborhood for $d = 2$.

Finally, we choose the nodes s' and t' such that the 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 outline of our transformation; Figure 6.7 gives a high-level illustration of the construction.

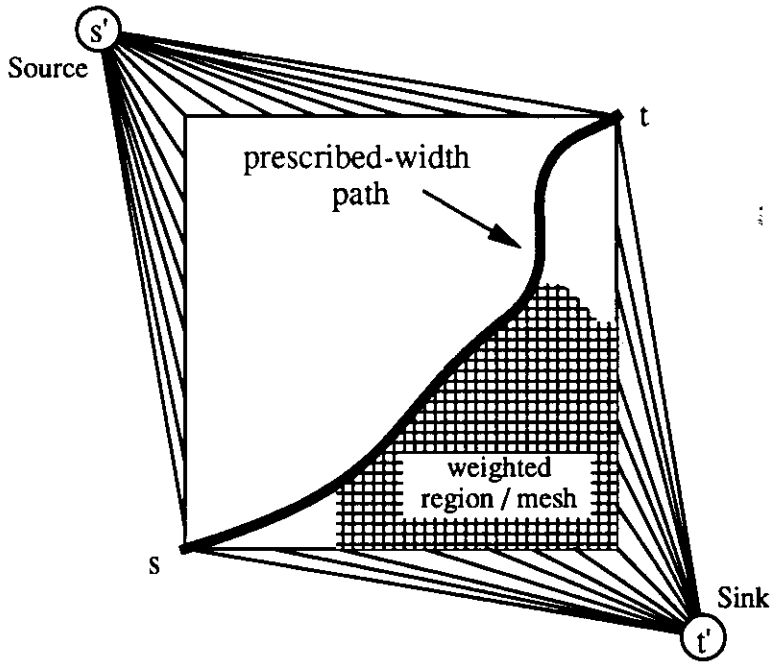


Figure 6.7: A prescribed-width routing instance transformed into a network flow instance.

Observe that up to this point, we have converted a prescribed-width routing instance to 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 on our network, 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 6.8.

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, the maximum flow computation in the transformed network will be asymptotically as fast as in the original network.

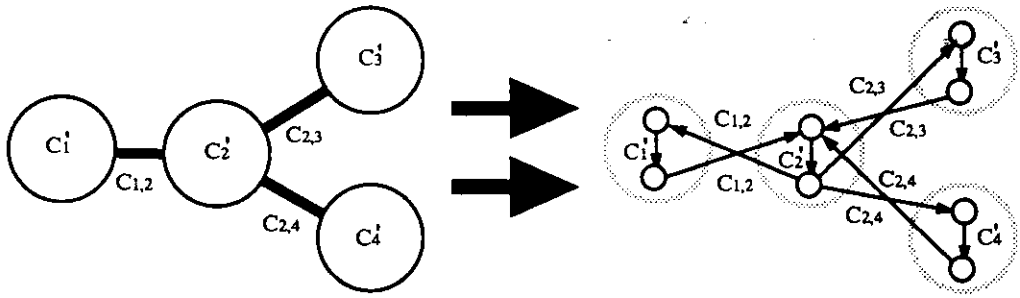


Figure 6.8: Transformation of a node- and arc-capacitated flow network to an arc-capacitated flow network (in our case $c_{i,j} = \infty$).

Note that a maximum flow in the arc-capacitated transformed graph corresponds to a minimum arc-cut in the transformed graph (via the max-flow min-cut theorem), which in turn corresponds to a minimum node-cut in the original graph since the transformation preserves minimal cutset costs. Moreover, the "width" of the cut can be no less than d since, as discussed above, the connection of each

node to all nodes in its d-neighborhood guarantees that any separating node set will have this prescribed width. A formal summary of our algorithm, which we call the d-PATH algorithm, is given in Figure 6.9.

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 d-separating path $\bar{P} \subseteq R$ connecting s and t
Create a d-connected mesh graph G of size $g \times g$ over R
Assign node weights in G according to weight function w
Set all boundary node weights to ∞
Transform node/arc-capacitated network G into arc-capacitated network G'
Add source s' and sink t' to G'
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 G'
Output all nodes incident to arcs in the minimum s'-t' cut of G'

Figure 6.9: Finding a prescribed-width path of minimum cost in an arbitrary weighted region, i.e., an optimal solution to the PWRG problem.

We conclude this section with the observation that the max-flow min-cut theorem [FF61] and the existence of efficient algorithms for maximum flow (e.g., [FF61] [CLR90]) together imply the following:

Theorem 6.2 *Algorithm d-PATH outputs an optimal solution to the PWRG problem in time polynomial in size of the mesh representation of the region R.*

6.2.3 A Practical Implementation

There are numerous algorithms for computing maximum flows in networks [AOT87] [FF61] [Hu69]. To demonstrate the viability of our approach, we have used an

existing implementation of Dinic's network flow algorithm [GG88]. 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 our mesh graph, the time complexity of the Dinic algorithm is $O(|N|^3)$. In practice, more efficient flow algorithms are available. For example, by using the network flow algorithm of [GTT89], we obtain the following:

Theorem 6.3 *For a given prescribed path width d, algorithm d-PATH solves the PWRG problem in* $O(|N|^2 \cdot \log|N|)$ *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 [GTT89] operates within time $O(|A| \cdot |N| \cdot \log(\frac{|N|^2}{|A|}))$. Assuming that d is a constant, the overall time complexity of our method is therefore $O(|N|^2 \cdot \log|N|)$. □

The time complexity may be further reduced in cases where the cost function over the region may only take on values from a fixed, bounded range. In this case, we may apply the maximum flow algorithm of [AOT87] to obtain an overall time complexity of $O(|N|^2)$ for our method.

6.3 Experimental Results

Our current implementation integrates 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 [GG88] to compute the flow, and then invoke Mathematica [Wol91] to draw the resulting path. We have tested our implementation on three classes of prescribed-width routing instances: uniformly weighted regions, environments with polygonal obstacles, and smooth randomly-costed environments. For each of these input classes, the boundary of the region is a rectangle, and we look for a path connecting s and t which are respectively in the top left and bottom right corners of the region.

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.

Our test environments with polygonal obstacles are populated by polygons of varying 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 an environment, changing the prescribed path width d may dramatically affect the optimum path topology with respect to the obstacles, as the path may be forced to take long detours in order to avoid narrow passages between objects. This phenomenon was indeed apparent in our results, as illustrated in Figures 6.10 and 6.11.

Finally, we tested our methodology on randomly-costed environments, using a mesh resolution of 100 by 100 nodes and a range of d values. Each random

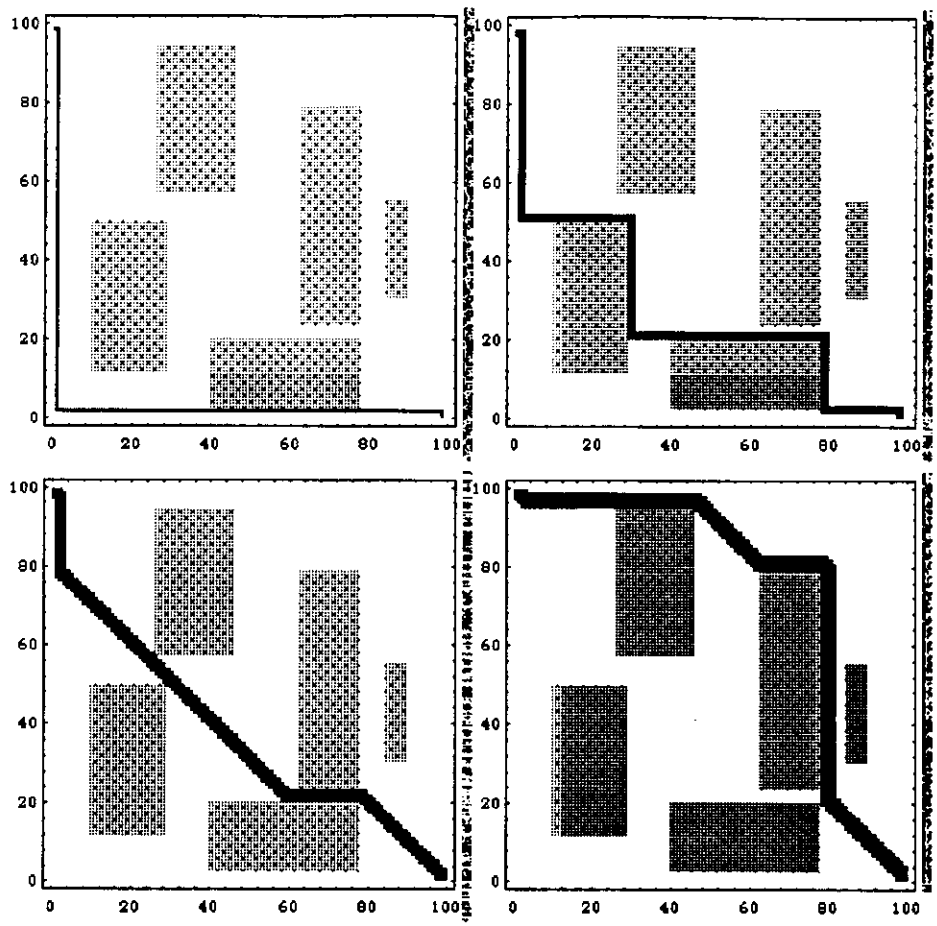


Figure 6.10: Prescribed-width paths in a region with polygonal obstacles. Note that the topology of the solutions changes as the prescribed width d is increased. The solutions shown correspond to widths $d = 1$ (top left), $d = 2$ (top right), $d = 3$ (bottom left), and $d = 4$ (bottom right).

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 large random positive weights. Then, a weight redistribution step was iteratively used to increment each node's weight by a small random fraction of the total weight of that node's immediate neighbors until a smooth randomly-costed environment was obtained. Figures 6.12 and 6.13 depict typical d-PATH output for the PWRG problem in a random environment. Regions of greater weight are

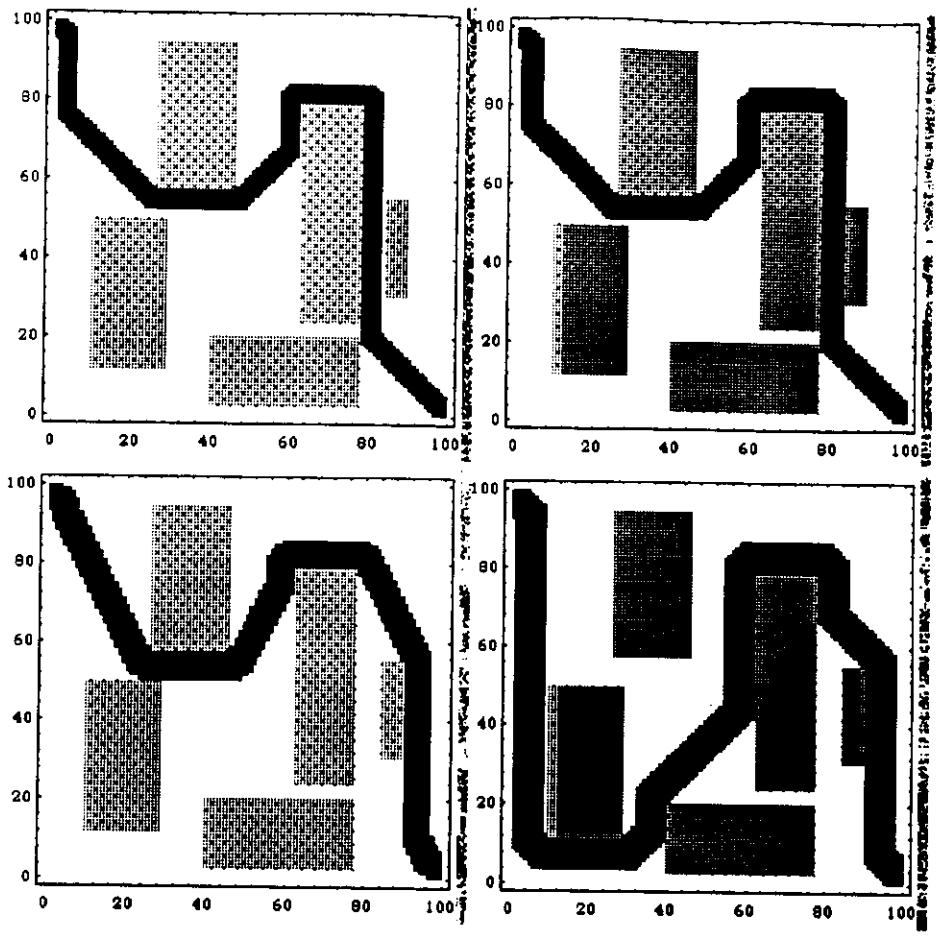


Figure 6.11: Prescribed-width paths in a region with polygonal obstacles (continued). The solutions shown correspond to widths $d = 5$ (top left), $d = 6$ (top right), $d = 7$ (bottom left), and $d = 8$ (bottom right).

denoted by darker shades, and regions of smaller weight are depicted by lighter densities. The optimum width-d path is highlighted in black. We emphasize that even with the Dinic algorithm, which is certainly not the ideal implementation for a mesh network topology, typical running times used to generate and solve all of the above classes of instances are on the order of only a few minutes on a Sun Sparc IPC. We therefore conclude that our approach constitutes a viable new method for prescribed-width routing in general environments.

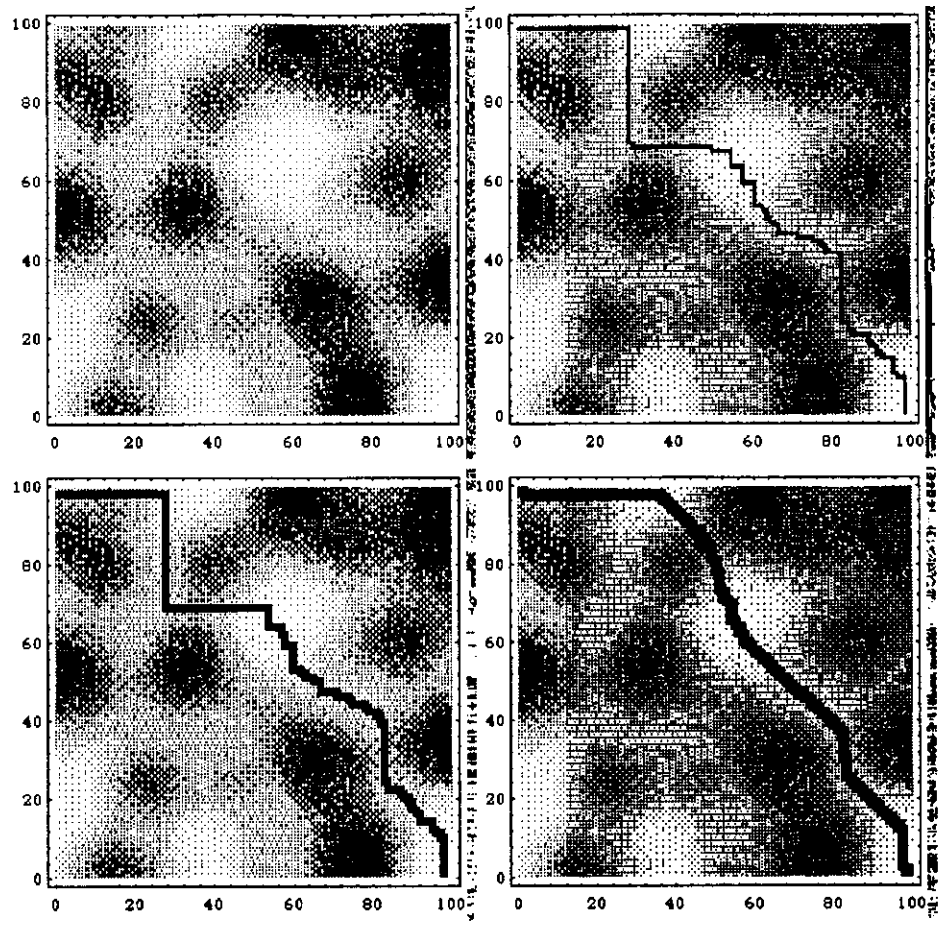


Figure 6.12: A smooth randomly-costed environment and its optimal prescribed-width path solutions. We see the environment itself (top left), as well as solutions corresponding to widths $d = 1$ (top right), $d = 2$ (bottom left), and $d = 3$ (bottom right).

6.4 Extension: Optimum Solution of the Discrete Plateau Problem

In this section, we extend our prescribed-width routing formulation and method to solve a discrete version of the classic Plateau problem. The problem of Plateau, in its simplest form, is to find the surface of minimum area that spans a given curve. The Plateau problem is part of the extensive field of minimal surfaces.

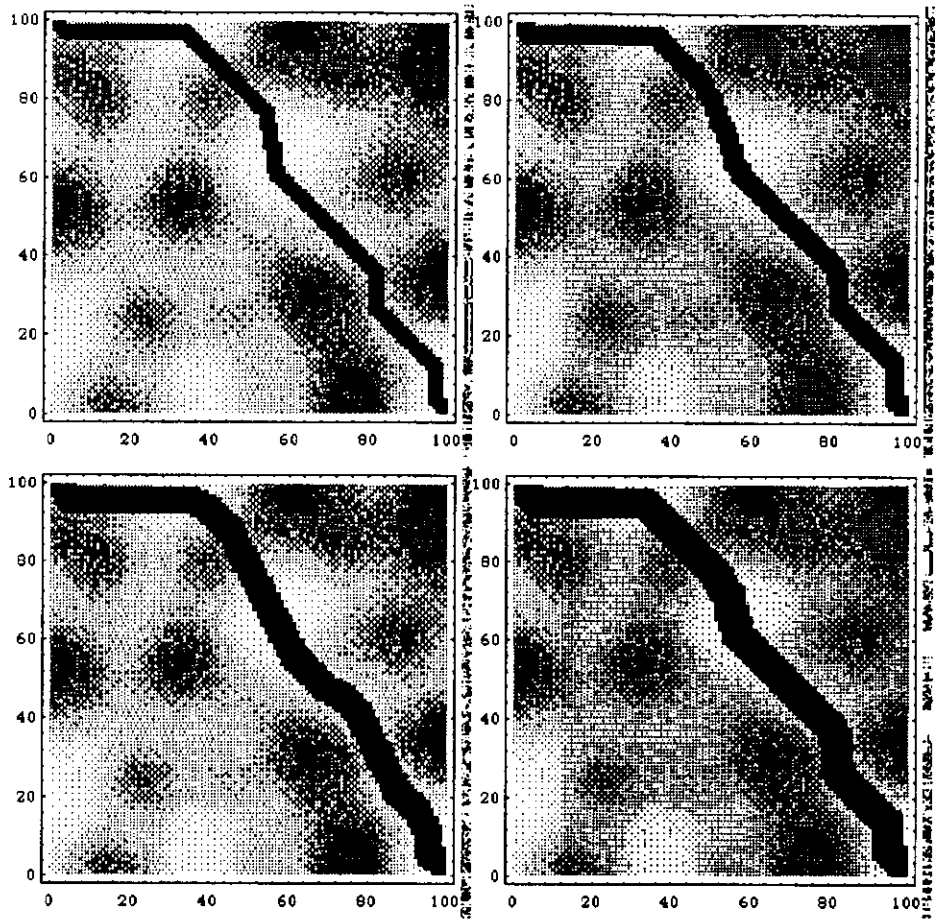


Figure 6.13: A smooth randomly-costed environment and its optimal prescribed-width path solutions (continued). Here we see the solutions corresponding to widths $d = 4$ (top left), $d = 5$ (top right), $d = 7$ (bottom left), and $d = 8$ (bottom right).

which originated with the development of the multidimensional calculus of variations [Cou50] [Fom90a] [Fom90b]. Although the study of specific two-dimensional minimal surfaces can be traced back to Lagrange (1768), the first extensive investigation of minimal surfaces was that of J. Plateau (1801-1883), who used soap films as physical models of minimal surfaces [Pla73]. Subsequently, many great mathematicians of the nineteenth and twentieth centuries, including Riemann, Weierstrass, and Schwarz, made contributions to the theory of minimal surfaces

[Str89], culminating with the discovery of general analytic solutions by Douglas [Dou31] and Rado [Rad33] in the 1930's.

It can be shown that a surface has minimal area if and only if it has zero mean curvature at each point, but this characterization is non-constructive. The problem is subtle: (i) a minimal surface may self-intersect, (ii) a given contour can bound numerous different surfaces of distinct topologies, and (iii) a very slight modification to the boundary contour can cause an enormous change in the corresponding minimal surface topology [Cou50] [Dou38]. Finding a minimal surface spanned by a given fixed boundary typically entails the solution of a system of partial differential equations. In many instances, analytic solutions are known to exist but remain virtually impossible to find, and solutions to specific cases have been individually discovered and proved over the last two centuries [Fom90a] [Oss69] [TF91]. A recent trend has been to solve instances of the Plateau problem empirically via numerical methods [Con67] [Gre65] [Gre67] [HSK74] [Tsu86] [Tsu87] [Tsu90] [Wil61].

We are primarily concerned with finding a minimal-area surface spanning a Jordan curve in three-dimensional Euclidean space (as opposed to higher-dimensional or non-Euclidean spaces):

The Plateau Problem (P1): Given a Jordan curve Γ^* in $\Re^3$, find a surface D^* of minimum area having boundary Γ^*.

The general formulation (P1) is difficult to address, and hence in the remainder of this section we deal with the specific class of instances, first described by Radó [Rad33], which satisfies:

(i) the orthogonal projection Γ of the given boundary Γ^* onto the xy-plane[3] is

[3]We adopt the convention of using starred letters to denote three-dimensional objects (e.g.

simple (i.e., non self-intersecting), and

(ii) the solution admits a functional representation $z = f(x, y)$, where f is continuous and has domain equal to the subset of the xy-plane bounded by Γ.

The first condition specifies that the projection of the boundary curve is homeomorphic to a circle, while the second condition specifies that the projection of the solution is homeomorphic to a disk. Historically, these two simplifying assumptions are often made in the in the treatment of the Plateau problem [Rad33]. We thus obtain:

The Restricted Plateau Problem (P2): Given a Jordan curve Γ^* in $\Re^3$ whose projection Γ onto the xy-plane is homeomorphic to a circle, find a surface D^* (having functional representation $z = f(x, y)$) of minimal area with boundary Γ^* (Figure 6.14).

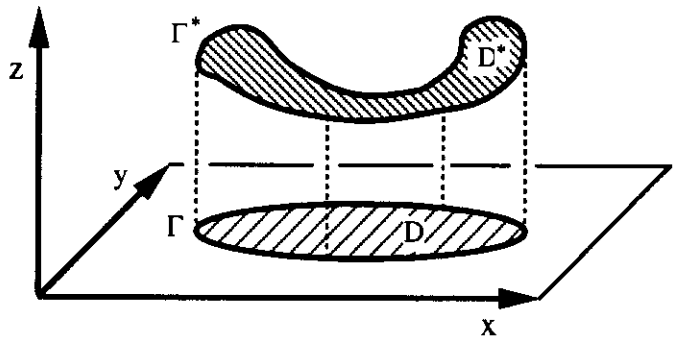


Figure 6.14: A surface D^* and its bounding contour Γ^*, as well as the corresponding projected region D and its boundary Γ.

This section develops a new constructive approach which optimally solves a class of discrete instances of the Plateau problem (P2). We generalize standard formulations in that we do not search for a minimal (zero-thickness) surface

a contour Γ^*, a surface D^*), while unstarred letters denote their respective projections (e.g., boundary Γ, a region D).

rather, we seek a minimal *slab* having some prescribed positive thickness d. Our algorithm obtains a minimum-cost slab having thickness everywhere of at least d, where cost is defined to be the total weighted volume of the slab with respect to an arbitrary weight function defined over $\Re^3$.

We diverge from the usual finite-element based approaches, and again employ instead a combinatorial method involving network flows [FF61], following ideas similar to those in Section 6.2 above. The crucial observation is again one concerning duality: a minimum-cost slab which *spans* a set of locations (e.g., the set of locations on the Jordan curve Γ) is also a minimum-cost cut-set which *separates* two other locations. Given this observation, we obtain minimal surface solutions efficiently via maximum flows, exploiting this duality between spanning and separating sets [HKR92b].

The rest of this section is organized as follows. Section 6.4.1 formally defines the problem. Section 6.4.2 formalizes our network-flow based approach to solving the Plateau problem, describes our current implementation, and illustrates the feasibility of our method on a small example.

6.4.1 Problem Formulation

By the Jordan curve theorem [CR41], the Jordan curve Γ partitions the plane into three mutually disjoint sets: Γ itself; its interior $int(\Gamma)$; and its exterior $ext(\Gamma)$. We assume that the prescribed boundary Γ of our minimal surface is always a Jordan curve. Given a three-dimensional point set $P^* \subset \Re^3$, define its *projection* to be the set of all points in the xy-plane with x and y coordinates equal to those of some point in P^*; i.e., $proj(P^*) \stackrel{\text{def}}{=} \{(x,y) \mid \exists z \ni (x,y,z) \in P^*\}$. We naturally extend this idea of projection to apply to any function $f(x,y)$ of two variables.

by considering the function f to be the set/relation $\{(x, y, f(x, y)) \mid x, y \in \Re$, where $f(x, y)$ is defined$\}$; thus, $proj(f)$ is simply the domain of the function f. A Jordan curve in the plane is called a *boundary*, and we define a contour to be a three-dimensional embedding of a Jordan curve:

Definition: A *contour* Γ^* is the set of points $\{(x, y, f(x, y)) \in \Re^3 \mid (x, y) \in \Gamma\}$ where f is a continuous real function $f : \Gamma \rightarrow \Re$ over some boundary Γ.

Thus, a contour Γ^* is a three-dimensional embedding of a curve that is homeomorphic to a circle, and the orthogonal projection of Γ^* onto the xy-plane is the boundary Γ of some region D. Similarly, the desired surface D^* satisfies $proj(D^*) = D$.

Definition: A *surface* D^* is the set of points $\{(x, y, f^*(x, y)) \in \Re^3 | (x, y) \in D\}$ where f^* is a continuous real function $f^* : D \rightarrow \Re$ defined over some region D in the xy-plane.

Any contour Γ^* induces an infinite family of distinct surfaces, each having Γ^* as boundary. Hence, the continuous surface function $f^* : D \rightarrow \Re$ is an extension of the contour function f, i.e., $f^*(x, y) = f(x, y)$ for all $(x, y) \in \Gamma \subseteq D$, where Γ^* is the contour function over the boundary Γ of the region D. The surface D^* is said to be *bounded* by its contour Γ^* (recall Figure 6.14).

Assuming that the partial derivatives $\frac{\partial f^*}{\partial x}$ and $\frac{\partial f^*}{\partial y}$ exist, it is not difficult to see that a surface D^* has well-defined area given by the integral:

$$\int_D \sqrt{\left(\frac{\partial f^*}{\partial x}\right)^2 + \left(\frac{\partial f^*}{\partial y}\right)^2 + 1}\ dx\ dy$$

The question of which surface has minimum area is precisely the Plateau problem (P2) stated above. In what follows, we develop the motivation for our network-flow based solution.

Given a surface D^*, define the *cylinder* $cyl(D^*)$ to be the set of all points directly above and below D^*; i.e., $cyl(D^*) \stackrel{\text{def}}{=} \{(x,y,z) \mid (x,y) \in proj(D^*), z \in \Re\}$. We may extend the *cyl* function to contours: $cyl(\Gamma^*) \stackrel{\text{def}}{=} cyl(int(proj(\Gamma^*)))$. Any surface D^* partitions $cyl(D^*)$ into three mutually disjoint subsets:

1. the points lying above D^*, denoted by $D_t^* \stackrel{\text{def}}{=} \{(x,y,z) \mid (x,y) \in proj(D^*), z > D^*(x,y)\}$;
2. the points of D^* itself, $\{(x,y,D^*(x,y)) \mid (x,y) \in proj(D^*)\}$; and
3. the points lying below D^*, denoted by $D_b^* \stackrel{\text{def}}{=} \{(x,y,z) \mid (x,y) \in proj(D^*), z < D^*(x,y)\}$.

In other words, the surface D^* *separates* D_b^* from D_t^*. In practice, we truncate both the top and the bottom of the cylinder $cyl(\Gamma^*)$ "far enough" above and below D^*, respectively, so that both D_t^* and D_b^* are bounded sets. We then define a weight function $w : cyl(\Gamma^*) \rightarrow \Re^+$ such that each point $s \in cyl(\Gamma^*)$ has a non-negative weight $w(s)$.

Following a similar approach to that of Section 6.4.1 we now generalize our formulation to allow a prescribed non-zero thickness to the separating surface D^*. From this, we will establish the relationship between the concept of *d-separation* and a thickness-d requirement.

Definition: Given a contour Γ^*, a *d-separating slab* $\hat{D}^* \subset cyl(D^*)$ is a superset of some surface D^* with Γ^* as the bounding contour of D^*, such that any point of $D_t^* - \hat{D}^*$ is at distance d or greater from any point of $D_b^* - \hat{D}^*$.

This is illustrated in Figure 6.15. We say that $\hat{D}^*$ is a *minimal d-separating slab* if no subset of $\hat{D}^*$ satisfies the preceding definition. The *cost* of a slab is defined

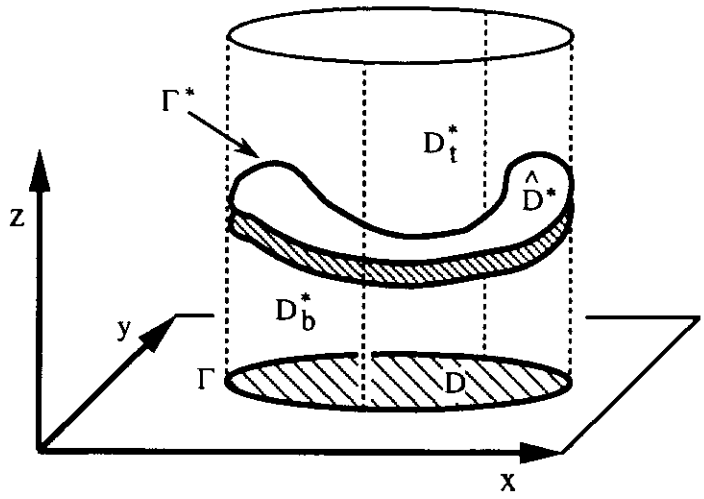


Figure 6.15: A d-separating slab $\hat{D}^*$ relative to a given contour Γ^*.

to be the integral of the weight function w over the volume of the slab. Because the weight function is non-negative and because we are interested in minimum-cost slabs, our discussion henceforth will refer only to minimal d-separating slabs. Given $d > 0$, the thickness-d Plateau problem is stated as follows:

The Thickness-d Plateau Problem (P3): Given a contour Γ^*, a weight function $w : cyl(\Gamma^*) \rightarrow \Re^+$, and a thickness $d > 0$, find a d-separating slab $\hat{D}^* \subset cyl(\Gamma^*)$ which has minimum total cost.

While the formulation specifies an arbitrary weight function that must be integrated over the volume of the slab to yield a total cost, in practical applications and in numerical approaches to the Plateau problem the space is often discretized relative to a given fixed grid or a sampling granularity [HSK74] [Tsu86] [Wil61]. In the present work, we also adopt the assumption of a fixed grid representation. With such a discrete version of problem (P3), the cost of a slab is naturally defined to be the sum of the weights of the grid points contained in it. The notion

of d-separation also naturally extends to the discrete grid:

Definition: Given a cylinder S, a *discrete* d-separating slab $\tilde{D}^*$ in the gridded space $\tilde{S}$ is a superset of the set of gridpoints of $\tilde{S}$ contained in some d-separating slab $\hat{D}^*$ in S (Figure 6.16).

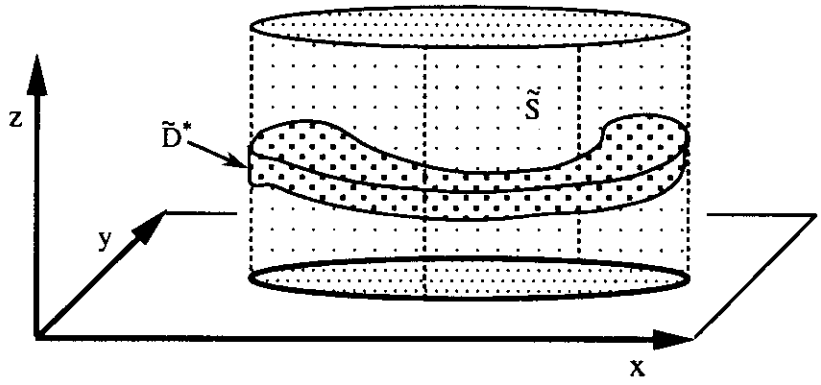


Figure 6.16: A discretized representation $\tilde{S}$ of a space S, and a discrete d-separating slab $\tilde{D}^*$ in $\tilde{S}$. Note that $\tilde{D}^*$ is the set of lattice points contained in the continuous d-separating slab $\hat{D}$ in S.

As in the continuous case, a discrete d-separating slab partitions the rest of the gridpoints into two sets, such that each gridpoint in one set is at least distance d away from any gridpoint in the other set. A discrete d-separating slab is minimal if no subset of it satisfies the preceding definition. We now have:

The Discrete Plateau Problem (P4): Given a weighted gridded space $\tilde{S}$ with boundary $B \subset \tilde{S}$, a contour Γ^* on the boundary of $\tilde{S}$, and a thickness $d > 0$, find a discrete d-separating slab $\tilde{D}^* \subseteq \tilde{S}$ which contains Γ^* and has minimum total cost.

-1zIntuitively, as the granularity quantum of the grid approaches zero, the so lution of the (P4) instance will converge to the solution for the corresponding continuous (P3) instance.

6.4.2 Applying the Network Flow Transformation

To solve the discrete Plateau problem (P4), we again use ideas from network flows in continua [Hu69] (see Section 6.2.2) and exploit the duality between a minimum cut and a maximum flow. The overview of our solution is as follows:

1. Discretize the volume of the cylinder induced by the given contour (i.e., consider only the lattice points of the cylinder with respect to a given resolution).

2. Create a d-connected mesh network over the cylinder by connecting each lattice point to all other lattice points within distance d; this guarantees that any separating set of nodes will have a minimum thickness d (we use the obvious one-to-one correspondence between nodes of the network and lattice points of the cylinder).

3. Connect a source node s (sink node t) to all nodes on the surface of the cylinder that lie below (above) the contour.

4. Use a maximum flow algorithm to compute a maximum s-t flow in the resulting network.

5. A maximum s-t flow specifies a minimum cut through the cylinder which separates s from t, and this minimum cut corresponds to a minimal thickness-d slab that contains the given contour.

Recall that any slab D^* will separate, or cut, D_t^* from D_b^*. Thus, a slab D^* with small cost will correspond to a cut between a node $s \in D_b^*$ and a node $t \in D_t^*$ with small cost (capacity). As in the prescribed-width path solution of Section

6.2, we derive the desired minimal slab via a maximum flow computation in an appropriately capacitated network.

Our first step towards this goal is to transform an instance of the discrete Plateau problem (P4) into an instance of network flow, by: (i) superimposing a discrete grid on $cyl(\Gamma^*)$, (ii) assigning capacities to nodes in the grid according to the weight function $w : cyl(\Gamma^*) \rightarrow \Re^+$, and (iii) converting the grid into a mesh network η by mapping gridpoints to capacitated nodes of η and then adding infinite-capacity arcs to join these nodes into a mesh.

To ensure that any s-t cut in the mesh created by (iii) will have the required thickness, we connect each node to all nodes in its d-neighborhood with infinite-capacity arcs, where d is the prescribed slab thickness. An illustration of this construction for $d = 1$ is given in Figure 6.17.

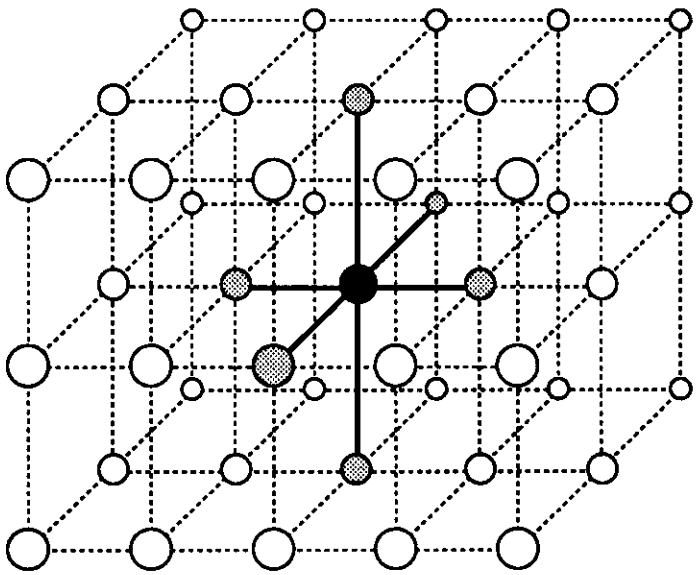


Figure 6.17: A node (black) and its d-neighborhood (grey nodes) for $d = 1$.

Finally, we introduce a source node s and a sink node t, connecting them respectively to the nodes of the boundary $B \subset \hat{S}$ lying "below" Γ^* and to the nodes of B lying "above" Γ^*. This forces any st-separating cut (which will correspond to the desired d-separating slab) to contain the given contour nodes Γ^* lying on the boundary B of the gridded space. In other words, we force the minimum slab to span the curve Γ^*. This completes the outline of our transformation; Figure 6.18 gives a high-level illustration of the construction.

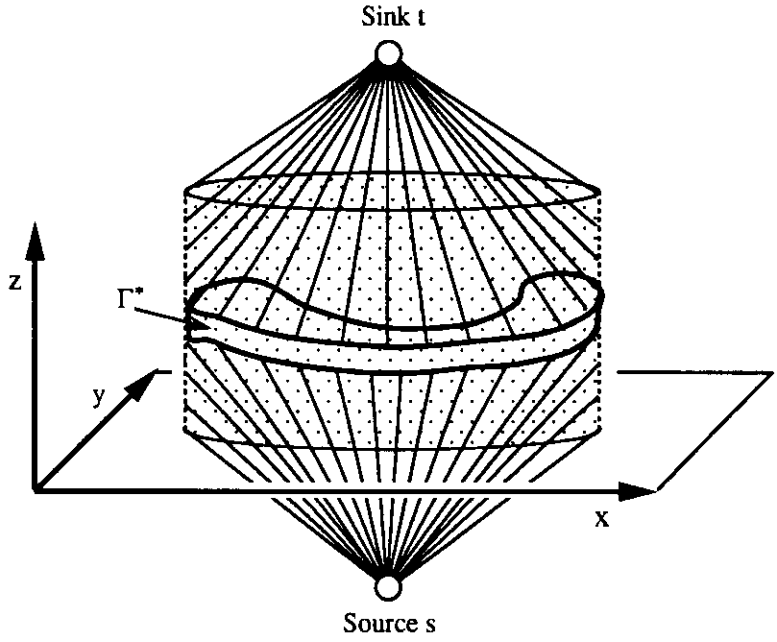


Figure 6.18: A discrete Plateau problem instance transformed into a network flow instance.

The resulting *d-connected* network has two useful properties. First, a minimum cutset of this network will consist only of nodes. This is because all arcs have infinite capacities, while there exist cuts with finite cost since all node capacities are finite. Second, the gridpoints associated with any nodeset that cuts this network must correspond to the lattice points of a discrete d-separating slab:

this property follows from the d-connectivity of the mesh.

Observe that up to this point, we have converted a discrete Plateau problem (P4) instance to a maximum flow instance on an undirected, *node-weighted* network. Thus, our final step is to construct an equivalent arc-capacitated maximum flow instance (Figure 6.8), again following Section 6.2.2. A formal summary of our algorithm, which we call the Disc_Plateau algorithm, is given in Figure 6.19.

Algorithm: Disc_Plateau
Input: contour Γ^* node weight function $w : cyl(\Gamma^*) \rightarrow \Re^+$ thickness $d > 0$ grid size g
Output: A minimal d-separating slab $\tilde{R}^*$ with boundary contour Γ^*
Create a d-connected mesh network G of grid size g over $cyl(\Gamma^*)$
Set node capacities of G according to weight function w
Set arc capacities of G to ∞
Set all boundary node capacities to ∞
Transform node-weighted network G into arc-capacitated network η
Create source node s and sink node t in η
Connect s to boundary nodes $(x, y, z) \in cyl(\Gamma^*) \ni z < \Gamma^*(x, y)$
Connect t to boundary nodes $(x, y, z) \in cyl(\Gamma^*) \ni z > \Gamma^*(x, y)$
Set capacities of all arcs adjacent to s or t to ∞
Compute a maximum s-t flow in η
Output all nodes incident to arcs in a minimum cut of η

Figure 6.19: Algorithm Disc_Plateau finds a d-separating slab of minimum cost in an arbitrarily weighted discrete space, i.e., an optimal solution to problem (P4).

We conclude this section with the observation that the max-flow min-cut theorem [FF61] and the existence of polynomial-time algorithms for maximum flow together imply the following:

Theorem 6.4 *Algorithm Disc_Plateau outputs an optimal solution to problem (P4) in time polynomial in the size of the gridded space $\check{S}$.*

In verifying the practicality of the Disc_Plateau methodology, we again used our existing implementation of the maximum flow algorithm of Dinic [GG88]; this code has $O(|N|^3)$ time complexity, where $|N|$ is the number of nodes in the discrete mesh representation of the space. More efficient flow algorithms may be used:

Theorem 6.5 *For a given prescribed slab thickness d, the Disc_Plateau algorithm solves problem (P4) in $O(|N|^2)$ time, where $|N|$ is the number of nodes in the gridded representation of the space.*

Proof: The degree of each node in the mesh is bounded by d^3, so that $|A| = O(d^3 \cdot |N|)$. The network flow algorithm of [AOT87] operates within time $O(|A| \cdot |N| \cdot \log(\frac{|A|}{|N|}))$. Assuming that d is a constant, the overall time complexity of our method is therefore $O(|N|^2)$. □

Our current implementation integrates ANSI C code to transform an arbitrary Plateau problem instance satisfying conditions (i) and (ii) of the (P2) formulation into a maximum-flow instance; we again use the Fortran-77 Dinic code of [GG88] to compute the flow and invoke Mathematica [Wol91] to draw the resulting surface. We have tested our implementation on several classes of problem instances, involving underlying spaces that are both uniformly weighted and non uniformly weighted. Figure 6.20 shows Disc_Plateau output for a small example with boundary contour consisting of four diagonals on the faces of a cube, uniform node weights, and $d = 2$; the "saddle" shown is optimal for the resolution used.

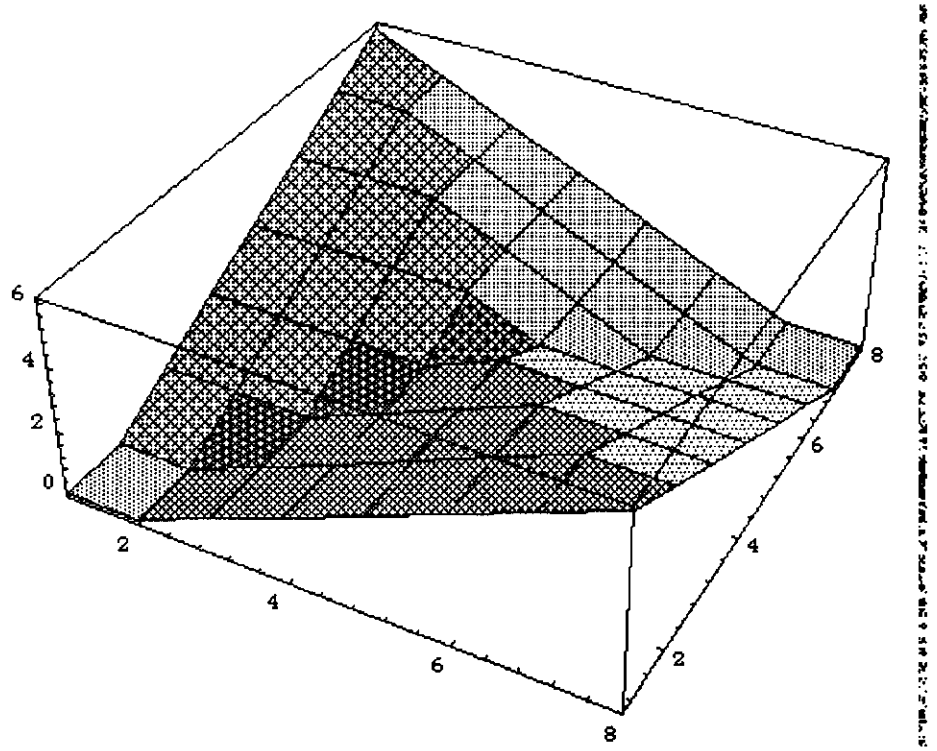


Figure 6.20: Minimal surface computation in a uniformly weighted space.

Based on our experimental results, we conclude that our approach constitutes a viable new method for solving the discrete Plateau problem in arbitrarily weighted spaces. Although Dinic's algorithm is certainly not the ideal maximum flow algorithm for a mesh topology, typical running times used to generate and solve our test cases range from only several seconds (for the example of Figure 6.20) to an hour on a low-end Sun-4 workstation with modest RAM/swap space. The observed runtimes show clear dependencies on the mesh resolution, the minimum slab thickness d, and the memory/swap configuration available.

6.5 Remarks and Extensions

We have developed a polynomial time algorithm which gives optimal solutions to the prescribed-width path problem in a discretized environment. Our method is based on the duality between connecting paths and separating sets, and relies on a maximum-flow computation to find a minimum-cost path of prescribed width in an arbitrarily weighted region. The accuracy of the solution with respect to the continuous version of the problem depends on the gridsize, which is intrinsic to the input. We have extended our method to three dimensions, where it yields a polynomial time algorithm which gives optimal solutions to a well-studied class of instances of the discrete Plateau problem, by finding a minimum-cost d-separating slab in an arbitrarily weighted space.

Chief among the 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, symmetric network that admits a concise representation. Additional research might also address more general prescribed-width routing issues, such as (i) incorporation of kinematic and dynamic considerations in robotics applications, (ii) use of hierarchical approaches as a heuristic speedup, and (iii) addressing the case where the endpoints of the path are not on the boundary of the region, and (iv) extension of the prescribed-width constraint to multi-point (tree) interconnections. One may also examine minimal surface computations using hierarchical approaches as a heuristic speedup; as with the lower-dimensional formulation, addressing the case where the contour does not necessarily lie on the boundary of the space is of interest. Finally, we believe that our methodology can address a larger class of Plateau instances by deomposing a spanning surface into patches which may then be individually optimized.

REFERENCES

[ABF88] F. Avnaim, J. D. Boissonnat, and B. Faverjon. "A Practical Exact Motion Planning Algorithm for Polygonal Objects Moving Amidst Polygonal Obstacles." Technical Report No. 890, INRIA, Sophia-Antipolis, France, 1988.

[ABP90] B. Awerbuch, A. Baratz, and D. Peleg. "Cost-Sensitive Analysis of Communication Protocol." In *Proc. ACM Symp. on Principles of Distributed Computing*, pp. 177–187, 1990.

[AOT87] R. K. Ahuja, J. B. Orlin, and R. E. Tarjan. "Improved Time Bounds for the Maximum Flow Problem." Technical Report CS-TR-118-87, Dept. of Computer Science, Princeton University, 1987.

[Bak90] H. Bakoglu. *Circuits, Interconnections and Packaging for VLSI.* Addison-Wesley, Reading, MA, 1990.

[BBB89] S. Boon, S. Butler, R. Byrne, B. Setering, M. Casalanda, and Al Scherf. "High Performance Clock Distribution For CMOS ASICS." In *Proc. IEEE Custom Integrated Circuits Conf.*, pp. 15.4.1–15.4.4, 1989.

[BD86] M. W. Bern and M. De Carvalho. "A Greedy Heuristic for the Rectilinear Steiner Tree Problem." Technical Report UCB/CSD 87/306, Computer Science Division (EECS), UCB, 1986.

[Ber88] M. W. Bern. "Two Probabilistic Results on Rectilinear Steiner Trees." *Algorithmica*, **3**:191–204, 1988.

[BGS91] R. W. Bassett, P. S. Gillis, and J. J. Shushereba. "Testing and Diagnosis of High-Density CMOS Multichip Modules." In *Proc. IEEE Workshop on Multichip Modules*, pp. 108–113, Santa Cruz, March 1991.

[BHH59] J. Beardwood, H. J. Halton, and J. M. Hammersley. "The Shortest Path Through Many Points." *Proc. Cambridge Philos. Soc.*, **55**:299–327, 1959.

[BK92a] K. D. Boese and A. B. Kahng. "Zero Skew Clock Routing With Minimum Wirelength." Technical Report CSD-TR-920012, Computer Science Department, UCLA, March 1992.

[BK92b] K. D. Boese and A. B. Kahng. "Zero Skew Clock Routing With Minimum Wirelength." In *Proc. IEEE Intl. ASIC Conf. (to appear)*, Rochester, NY, September 1992.

[BL91] J. Barraquand and J. C. Latombe. "Nonholonomic Multibody Mobile Robots: Controllability and Motion Planning in the Presence of Obstacles." *Proc. IEEE Intl. Conf. on Robotics and Automation*, pp. 2328–2335, April 1991.

[BMH89] R. H. Bruce, W. P. Meuli, and J. Ho. "Multi Chip Modules." In *Proc. ACM/IEEE Design Automation Conf.*, pp. 389–393, Las Vegas, June 1989.

[BP83] 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**:177–180, 1983.

[BR92] P. Berman and V. Ramaiyer. "Improved Approximations for the Steiner Tree Problem." In *Proc. ACM/SIAM Simposium on Discrete Algorithms*, pp. 325–334, San Francisco, CA, January 1992.

[Bur91] J. Burkis. "Clock Tree Synthesis for High Performance ASICs." In *Proc. IEEE Intl. ASIC Conf.*, pp. 9.8.1–9.8.4, Rochester, NY, September 1991.

[BWM86] 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. on Computer Design*, pp. 118–122, Port Chester, NY, October 1986.

[Can88] J. Canny. *The Complexity of Robot Motion Planning.* MIT Press, 1988.

[CCR92] N.-C. Chou, C.-K. Cheng, and T. C. Russell. "High-Performance Microelectronic Substrate Verification Using Multiprobe Testers." In *Proc. IEEE Intl. ASIC Conf. (to appear)*, Rochester, NY, September 1992.

[CE92] B. Chazelle and H. Edelsbrunner. "An Optimal Algorithm for Intersecting Line Segments." *J. ACM*, **39**(1):177–180, January 1992.

[CH90] T. H. Chao and Y. C. Hsu. "Rectilinear Steiner Tree Construction by Local and Global Refinement." In *Proc. IEEE Intl. Conf. on Computer-Aided Design*, pp. 432–435, Santa Clara, CA, November 1990.

[CK90] P. K. Chan and K. Karplus. "Computing Signal Delay in General RC Networks by Tree/Link Partitioning." *IEEE Trans. on Computer-Aided Design*, **9**(8):898–902, August 1990.

[CKC84] J. C. Crowell, R.J. Keogh, and J.A. Conti. "Moving Probe Bare Board Tester Offers Unlimited Testing Flexibility." *Industrial Electronics Equipment Design*, September 1984.

[CKR90] J. Cong, A. B. Kahng, and G. Robins. "Performance-Driven Global Routing for Cell Based IC's." Technical Report CSD-TR-900052, Computer Science Department, UCLA, December 1990.

[CKR91a] J. Cong, A. B. Kahng, and G. Robins. "Matching-Based Methods for High-Performance Clock Routing." *IEEE Trans. on Computer-Aided Design (submitted)*, 1991.

[CKR91b] J. Cong, A. B. Kahng, and G. Robins. "On Clock Routing For General Cell Layouts." In *Proc. IEEE Intl. ASIC Conf.*, pp. P14:5.1–P14:5.4, Rochester, NY, September 1991.

[CKR91c] 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. on Computer Design*, pp. 170–173, Cambridge, MA, October 1991.

[CKR92a] 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. on Circuits and Systems*, pp. 2240–2243, San Diego, CA, May 1992.

[CKR92b] J. Cong, A. B. Kahng, G. Robins, C. K. Wong, and M. Sarrafzadeh. "Provably Good Performance-Driven Global Routing." *IEEE Trans. on Computer-Aided Design*, **11**(6):739–752, 1992.

[CLR90] T. H. Cormen, C. E. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[Con67] P. Concus. "Numerical Solution of the Minimal Surface Equation." *Mathematics of Computation*, **21**:340–350, 1967.

[Con90] J. H. Connell. *Minimalist Mobile Robotics.* Academic Press, 1990.

[Cou50] R. Courant. *Dirichlet's Principle, Conformal Mapping, and Minimal Surfaces.* Interscience Publishers, Inc., New York, 1950.

[CP88] J. Cong and B. Preas. "A New Algorithm for Standard Cell Global Routing." In *Proc. IEEE Intl. Conf. on Computer-Aided Design*, pp. 176–179, Santa Clara, CA, November 1988.

[CR41] Courant and Robbins. *What is Mathematics? An Elementary Approach to Ideas and Methods.* Oxford University Press, London, England, 1941.

[CR91] J. Cohoon and J. Randall. "Critical Net Routing." In *Proc. IEEE Intl. Conf. on Computer Design*, pp. 174–177, Cambridge, MA, October 1991.

[DAD84] A. E. Dunlop, V. D. Agrawal, D.N. Deutsch, M. F. Jukl, P. Kozak, and M. Wiesel. "Chip Layout Optimization Using Critical Path Weighting." In *Proc. ACM/IEEE Design Automation Conf.*, pp. 133–136, 1984.

[Dai91] W. W. Dai. "Performance Driven Layout of Thin-film Substrates for Multichip Modules." In *Proc. IEEE Workshop on Multichip Modules*, pp. 114–121, Santa Cruz, March 1991.

[DAK85] W. M. Dai, T. Asano, and E. S. Kuh. "Routing Region Definition and Ordering Scheme for Building-Block Layout." *IEEE Trans. on Computer-Aided Design*, 4(7):189–197, July 1985.

[DFW84] S. Dhar, M. A. Franklin, and D. F. Wann. "Reduction of Clock Delays in VLSI Structures." In *Proc. IEEE Intl. Conf. on Computer Design*, pp. 778–783, Port Chester, NY, October 1984.

[DH90] D. Z. Du and F. K. Hwang. "A Proof of Gilbert-Pollak's Conjecture on the Steiner Ratio." In *Proc. IEEE Symp. on Foundations of Computer Science*, 1990.

[DNA90] W. E. Donath, R. J. Norman, B. K. Agrawal, and S. E. Bello. "Timing Driven Placement Using Complete Path Delays." In *IEEE Trans. on Computer-Aided Design*, pp. 84–89, 1990.

[Dou31] J. Douglas. "Solution of the Problem of Plateau." *Trans. Amer. Math. Soc.*, **33**:263–321, 1931.

[Dou38] J. Douglas. "The Most General Form of the Problem of Plateau." *Proc. Nat. Acad. Sci. USA*, **24**:360–364, 1938.

[Eda90] M. Edahiro. "A Clock Net Reassignment Algorithm Using Voronoi Diagrams." In *Proc. IEEE Intl. Conf. on Computer-Aided Design*, pp. 420–423, Santa Clara, CA, November 1990.

[EIR90] D. Eppstein, G. Italiano, R. E. Tarjan R. Tamassia, J. Westbrook, and M. Yung. "Maintenance of a Minimum Spanning Forest in a Dynamic Planar Graph." In *Proc. ACM/SIAM Simposium on Discrete Algorithms*, pp. 1–11, San Francisco, CA, January 1990.

[Elm48] W. C. Elmore. "The Transient Response of Damped Linear Networks with Particular Regard to Wide-Band Amplifiers." *J. Appl. Phys.*, **19**(1):55–63, 1948.

[Eve79] S. Even. *Graph Algorithms.* Computer Science Press, Inc., Potomac, MD, 1979.

[FF61] L. R. Ford and D. R. Fulkerson. *Flows in Networks.* Princeton University Press, Princeton, NJ, 1961.

[Fis90] J. Fishburn. "Clock Skew Optimization." *IEEE Trans. on Computers*, **39**(7):945–951, 1990.

[FK82] A. L. Fisher and H. T. Kung. "Synchronizing Large Systolic Arrays." In *Proc. of SPIE*, pp. 44–52, May 1982.

[Fom90a] A. T. Fomenko. *The Plateau Problem, Part I: Historical Survey.* Gordon and Breach Science Publishers, Amsterdam, 1990.

[Fom90b] A. T. Fomenko. *The Plateau Problem, Part II: The Present State of the Theory.* Gordon and Breach Science Publishers, Amsterdam, 1990.

[Fou84] L. R. Foulds. "Maximum Savings in the Steiner Problem in Phylogeny." *J. Theoretical Biology*, **107**:471–474, 1984.

[Gab76] H. Gabow. "An Efficient Implementation of Edmond's Algorithm for Maximum Matching on Graphs." *J. Assoc. Comput. Mach.*, **23**:221–234, 1976.

[GG88] D. Goldfarb and M. D. Grigoriadis. "A Computational Comparison of the Dinic and Network Simplex Methods for Maximum Flow." *Annals of Operation Research*, **13**(1-4):83–123, June 1988.

[GHY74] R. E. Gomory, T. C. Hu, and J. M. Yohe. "R-Separating Sets." *Can. J. Math.*, **XXVI**(6):1418–1429, 1974.

[GJ77] M. Garey and D. S. Johnson. "The Rectilinear Steiner Problem is NP-Complete." *SIAM J. Applied Math.*, **32**(4):826–834, 1977.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: a Guide to the Theory of NP Completeness.* W. H. Freeman, San Francisco, 1979.

[GP68] E. N. Gilbert and H. O. Pollak. "Steiner Minimal Trees." *SIAM J. Applied Math.*, **16**:1–29, 1968.

[GP87] G. Georgakopoulos and C. H. Papadimitriou. "The 1-Steiner Tree Problem." *J. Algorithms*, **8**:122–130, 1987.

[Gre65] D. Greenspan. "On Approximating Extremals of Functions. I: The Method and Examples for Boundary Value Problems." *ICC Bull.*, **4**:99–120, 1965.

[Gre67] D. Greenspan. "On Approximating Extremals of Functions. II: Theory and Generalization Related to Boundary Value Problems for Nonlinear Differential Equations." *Intl. J. Engineering Sci.*, **5**:571–588, 1967.

[Gro88] Joint Test Action Group. *JTAG Boundary-Scan Architecture Standard Proposal*, version 2.0 edition, March 1988.

[GTT89] A. V. Goldberg, E. Tardos, and R. E. Tarjan. "Network Flow Algorithms." manuscript, March 1989.

[GWR90] S.D. Golladay, N.A. Wagner, J.R. Rudert, and R.N. Schmidt. "Electron-Beam Technology for Open/Short Testing of Multi-Chip Substrates." *IBM J. Res. Develop.*, **34**(2/3):250–259, March/May 1990.

[Han66] M. Hanan. "On Steiner's Problem With Rectilinear Distance." *SIAM J. Applied Math.*, **14**:255–265, 1966.

[Her90] D. Herrell. "Multichip Module Technology at MCC." In *Proc. IEEE Intl. Symp. on Circuits and Systems*, pp. 2099–2103, June 1990.

[HHK92] T. C. Hu, J. H. Huang, and A. B. Kahng, 1992. draft, submitted for publication.

[HKR91] T. C. Hu, A. B. Kahng, and G. Robins. "Optimal Robust Path Planning in General Environments." Technical Report CSD-910082, Computer Science Department, UCLA, December 1991.

[HKR92a] T. C. Hu, A. B. Kahng, and G. Robins. "Optimal Robust Path Planning in General Environments." *IEEE Trans. on Robotics and Automation*, 1992.

[HKR92b] T. C. Hu, A. B. Kahng, and G. Robins. "Optimal Solution of the Discrete Plateau Problem." Technical Report CSD-920006, Computer Science Department, UCLA, January 1992.

[HKR92c] T. C. Hu, A. B. Kahng, and G. Robins. "Optimal Solution of the Discrete Plateau Problem." *Mathematics of Computation (submitted)*, 1992.

[HLC89] J. Ho, D. T. Lee, C. H. Chang, and C. K. Wong. "Bounded-Diameter Spanning Trees and Related Problems." In *Proc. ACM Symp. on Computational Geometry*, pp. 276–282, 1989.

[HNY87] P. S. Hauge, R. Nair, and E. J. Yoffa. "Circuit Placement for Predictable Performance." In *Proc. IEEE Intl. Conf. on Computer-Aided Design*, pp. 88–91, Santa Clara, CA, November 1987.

[HSK74] M. Hinata, M. Shimasaki, and T. Kiyono. "Numerical Solution of Plateau's Problem by a Finite Element Method." *Mathematics of Computation*, **28**(125):45–60, January 1974.

[Hu69] T. C. Hu. *Integer Programming and Network Flows.* Addison-Wesley, Reading, MA, 1969.

[HVW90a] N. Hasan, G. Vijayan, and C. K. Wong. "A Neighborhood Improvement Algorithm for Rectilinear Steiner Trees." In *Proc. IEEE Intl. Symp. on Circuits and Systems*, New Orleans, LA, 1990.

[HVW90b] J.-M. Ho, G. Vijayan, and C. K. Wong. "New Algorithms for the Rectilinear Steiner Tree Problem." *IEEE Trans. on Computer-Aided Design*, **9**(2):185–193, 1990.

[Hwa76] F. K. Hwang. "On Steiner Minimal Trees with Rectilinear Distance." *SIAM J. Applied Math.*, **30**(1):104–114, 1976.

[Hwa78] F. K. Hwang. "The Rectilinear Steiner Problem." *J. Design Automation and Fault-Tolerant Computing*, **2**:303–310, 1978.

[Hwa79a] F. K. Hwang. "An O(n log n) Algorithm for Rectilinear Minimal Spanning Trees." *J. ACM*, **26**(2):177–182, 1979.

[Hwa79b] F. K. Hwang. "An O(n log n) Algorithm for Suboptimal Rectilinear Steiner Trees." *IEEE Trans. on Circuits and Systems*, **26**(1):75–77, 1979.

[JAM89] D. S. Johnson, C. R. Aragon, L. A. McGeogh, and C. Schevon. "Optimization by Simulated Annealing: An Experimental Evaluation (part 1)." *Operations Research*, **37**(6):865–892, March/May 1989.

[JK89] M. A. B. Jackson and E. S. Kuh. "Performance-Driven Placement of Cell-Based IC's." In *Proc. ACM/IEEE Design Automation Conf.*, pp. 370–375, 1989.

[JKM87] M. A. B. Jackson, E. S. Kuh, and M. Marek-Sadowska. "Timing-Driven Routing for Building Block Layout." In *Proc. IEEE Intl. Symp. on Circuits and Systems*, pp. 518–519, 1987.

[JSK90] M. A. B. Jackson, A. Srinivasan, and E. S. Kuh. "Clock Routing for High-Performance IC's." In *Proc. ACM/IEEE Design Automation Conf.*, pp. 573–579, 1990.

[KCR90] A. B. Kahng, J. Cong, and G. Robins. "High-Performance Clock Routing Based on Recursive Geometric Matching." Technical Report CSD-TR-900046, Computer Science Department, UCLA, December 1990.

[KCR91] A. B. Kahng, J. Cong, and G. Robins. "High-Performance Clock Routing Based on Recursive Geometric Matching." In *Proc. ACM/IEEE Design Automation Conf.*, pp. 322–327, June 1991.

[KMB81] L. Kou, G. Markowsky, and L. Berman. "A Fast Algorithm for Steiner Trees." *Acta Informatica*, **15**:141–145, 1981.

[KR90] 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. on Computer-Aided Design*, pp. 428–431, Santa Clara, CA, November 1990.

[KR91a] A. B. Kahng and G. Robins. "New Family of Steiner Tree Heuristics with Good Performance." Technical Report CSD-TR-900014, Computer Science Department, UCLA, April 1991.

[KR91b] A. B. Kahng and G. Robins. "On Performance Bounds for a Class of Rectilinear Steiner Tree Heuristics in Arbitrary Dimension." Technical Report CSD-TR-900015, Computer Science Department, UCLA, April 1991.

[KR92a] A. B. Kahng and G. Robins. "A New Class of Iterative Steiner Tree Heuristics With Good Performance." *IEEE Trans. on Computer-Aided Design (to appear)*, 1992.

[KR92b] A. B. Kahng and G. Robins. "On Performance Bounds for a Class of Rectilinear Steiner Tree Heuristics in Arbitrary Dimension." *IEEE Trans. on Computer-Aided Design (to appear)*, 1992.

[Kru56] M. Kruskal. "On the Shortest Spanning Subtree of a Graph, and the Traveling Salesman Problem." *Proc. Amer. Math Soc.*, **7**:48–50, 1956.

[KRW91a] A. B. Kahng, G. Robins, and E. A. Walkup. "On Connectivity Verification in Multi-Chip Module Substrates." Technical Report CSD-TR-910074, Computer Science Department, UCLA, 1991.

[KRW91b] A. B. Kahng, G. Robins, and E. A. Walkup. "On Connectivity Verification in Multi-Chip Module Substrates." *IEEE Trans. on Computers (submitted)*, 1991.

[KRW92] A. B. Kahng, G. Robins, and E. A. Walkup. "New Results and Algorithms for MCM Substrate Testing." In *Proc. IEEE Intl. Symp. on Circuits and Systems*, pp. 1113–1116, San Diego, CA, May 1992.

[Lat91] J. C. Latombe. *Robot Motion Planning.* Kluwer Academic Publishers, 1991.

[Law76] E. Lawler. *Combinatorial Optimization: Networks and Matroids.* Holt Rinehart and Winston, New York, 1976.

[Law85] E. L. Lawler. *The Traveling Salesman Problem: a Guided Tour of Combinatorial Optimization.* Wiley, Chichester, New York, 1985.

[LBH76] J. H. Lee, N. K. Bose, and F. K. Hwang. "Use of Steiner's Problem in Sub-Optimal Routing in Rectilinear Metric." *IEEE Trans. on Circuits and Systems*, **23**:470–476, 1976.

[LD90] I. Lin and D. H. C. Du. "Performance-Driven Constructive Placement." In *Proc. ACM/IEEE Design Automation Conf.*, pp. 103–106, 1990.

[Len90] T. Lengaur. *Combinatorial Algorithms for Integrated Circuit Layout.* John Wiley & Sons Ltd, West Sussex, England, 1990.

[Lin65] S. Lin. "Computer Solutions of the Traveling Salesman Problem." *Bell System Technical Journal*, **44**:2245–2269, 1965.

[LS90] K. W. Lee and C. Sechen. "A New Global Router for Row-Based Layout." In *Proc. IEEE Intl. Conf. on Computer-Aided Design*, pp. 180–183, Santa Clara, CA, November 1990.

[MC80] C. Mead and L. Conway. *Introduction to VLSI Systems.* Addison-Wesley, Reading, MA, 1980.

[McL90] J. McLeod. "In PC-board CAD, Backs Are Against the Wall." *Electronics*, **63**(3):52–54, March 1990.

[McW91] B. McWilliams. "private communication." (invited talk at CANDE meeting), San Marcos, CA, April 1991.

[ML89] M. Marek-Sadowska and S. P. Lin. "Timing Driven Placement." In *Proc. IEEE Intl. Conf. on Computer-Aided Design*, pp. 94–97, Santa Clara, CA, November 1989.

[MMP87] J. S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou. "The Discrete Geodesic Problem." *SIAM J. Computing*, **16**(4):647–668, 1987.

[OPK90] Y. Ogawa, M. Pedram, and E. S. Kuh. "Timing-Driven Placement for General Cell Layout." In *Proc. IEEE Intl. Symp. on Circuits and Systems*, pp. 872–876, 1990.

[Oss69] R. Osserman. *A Survey of Minimal Surfaces.* Van Nostrand Reinhold Company, New York, 1969.

[OY83] C. O'Dunlaing and C. K. Yap. "A Retraction Method for Planning the Motion of a Disc." *J. Algorithms*, **6**:187–192, 1983.

[Pap85] C. H. Papadimitriou. "An Algorithm for Shortest-Path Motion in Three Dimensions." *Information Processing Letters*, **20**:259–263. 1985.

[PL88] B. T. Preas and M. J. Lorenzetti. *Physical Design Automation of VLSI Systems.* Benjamin/Cummings, Menlo Park, CA, 1988.

[Pla73] J. Plateau. *Statique expérimentale et théorique des liquides soumis aux seules forces molécularies.* Gathier - Villars, Paris, 1873.

[Pri57] A. Prim. "Shortest Connecting Networks and Some Generalizations." *Bell Syst. Tech J.*, **36**:1389–1401, 1957.

[PS82] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization.* Prentice-Hall, 1982.

[PS85] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Springer-Verlag, New York, 1985.

[Rad33] T. Radó. *On the Problem of Plateau.* Springer-Verlag, Berlin, 1933.

[Ric89] D. Richards. "Fast Heuristic Algorithms for Rectilinear Steiner Trees." *Algorithmica*, **4**:191–207, 1989.

[RPH83] J. Rubinstein, P. Penfield, and M. A. Horowitz. "Signal Delay in RC Tree Networks." *IEEE Trans. on Computer-Aided Design*, **2**(3):202–211, 1983.

[RS89] P. Ramanathan and K. G. Shin. "A Clock Distribution Scheme for Non-Symmetric VLSI Circuits." In *Proc. IEEE Intl. Conf. on Computer-Aided Design*, pp. 398–401, Santa Clara, CA, November 1989.

[Rus91] T. Russell. "private communication.", August 1991. ALCOA Corp.

[Sak83] T. Sakurai. "Approximation of Wiring Delay in MOSFET LSI." *IEEE Journal of Solid-State Circuits*, **18**(4):418–426, August 1983.

[SCK91a] A. Srinivasan, K. Chaudhary, and E. S. Kuh. "RITUAL: A Performance Driven Placement Algorithm." Technical Report UCB/ERL M91/103, College of Engineering, UCB, November 1991.

[SCK91b] A. Srinivasan, K. Chaudhary, and E. S. Kuh. "RITUAL: A Performance Driven Placement Algorithm." In *Proc. IEEE Intl. Conf. on Computer-Aided Design*, pp. 48–51, Santa Clara, CA, November 1991.

[Sec88] C. Sechen. *VLSI Placement and Global Routing Using Simulated Annealing.* Kluwer Academic Publishers, 1988.

[Sed88] R. Sedgewick. *Algorithms, 2nd ed.* Addison-Wesley, Reading, MA, 1988.

[Ser81] M. Servit. "Heuristic Algorithms for Rectilinear Steiner Trees." *Digital Process.*, **7**(1):21–31, 1981.

[Sha91] K. P. Shambrook. "An Overview of Multichip Module Technologies." In *Proc. IEEE Workshop on Multichip Modules*, pp. 1–6, Santa Cruz, CA, March 1991.

[SL79] J. M. Smith and J. S. Liebman. "Steiner Trees, Steiner Circuits and the Interference Problem in Building Design." *Engineering Optimization*, **4**:15–36, 1979.

[SLL80] 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**:179–192, 1980.

[Sny90] T. L. Snyder. "A Simple and Faster Algorithm for the Rectilinear Steiner Problem in General Dimension." In *Proc. ACM Symp. on Computational Geometry*, 1990.

[SR83] K. J. Supowit and E. M. Reingold. "Divide and Conquer Heuristics for Minimum Weighted Euclidean Matching." *SIAM J. Computing*, **12**(1):118–143, 1983.

[SR90] C. C. De Souza 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**:109–111, 1990.

[Sri91] A. Srinivasan, October 1991. private communication.

[SRP83] 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**(1):144–156, 1983.

[SS90a] T. L. Snyder and J. M. Steele. "Worst-Case Greedy Matchings in the Unit d-Cube." *Networks*, **20**:779–800, 1990.

[SS90b] S. Sutanthavibul and E. Shragowitz. "An Adaptive Timing-Driven Layout for High Speed VLSI." In *Proc. ACM/IEEE Design Automation Conf.*, pp. 90–95, 1990.

[SSB91] R. G. Sartore, N. Shastry, U. Brahme, K. Jefferson, and R. Halaviati. "Tutorial for Computer Aided Diagnostic E-Beam Testing of ASICs." In *Proc. IEEE Intl. ASIC Conf.*, pp. T8:1.1 - T8:1.7, Rochester, NY, September 1991.

[SSH87] J. T. Schwartz, M. Sharir, and H. Hopcroft. *Planning, Geometry and Complexity of Robot Motion.* Ablex Publishing Corp., 1987.

[Ste88] J. M. Steele. "Growth Rates of Euclidean Minimal Spanning Trees With Power Weighted Edges." *Annals of Probability*, **16**(4):1767–1787, 1988.

[Str89] M. Struwe. *Plateau's Problem and the Calculus of Variations.* Princeton University Press, NJ, 1989.

[Sup90] K. J. Supowit. "New Techniques for Some Dynamic Closest-Point and Farthest-Point Problems." In *Proc. ACM/SIAM Simposium on Discrete Algorithms*, pp. 84–90, 1990.

[SW92] M. Sarrafzadeh and C. K. Wong. "Hierarchical Steiner Tree Construction in Uniform Orientations." *IEEE Trans. on Computer-Aided Design (to appear)*, 1992.

[TD91] M. Taylor and W. W. Dai. "TinyMCM." In *Proc. IEEE Workshop on Multichip Modules*, pp. 143–147, Santa Cruz, CA, March 1991.

[TF91] D. T. Thi and A. T. Fomenko. *Minimal Surfaces, Stratified Multivarifolds, and the Plateau Problem.* American Mathematical Society, Providence, RI, 1991.

[Tsa91] R. S. Tsay. "Exact Zero Skew." In *Proc. IEEE Intl. Conf. on Computer-Aided Design*, pp. 336–339, Santa Clara, CA, November 1991.

[Tsu86] T. Tsuchiya. "On Two Methods for Approximating Minimal Surfaces in Parametric Form." *Mathematics of Computation*, **46**(174):517 529, April 1986.

[Tsu87] T. Tsuchiya. "Discrete Solution of the Plateau Problem and its Convergence." *Mathematics of Computation*, **49**(179):157–165, July 1987.

[Tsu90] T. Tsuchiya. "A Note on Discrete Solutions of the Plateau Problem" *Mathematics of Computation*, **54**(189):131–138, January 1990.

[Vai88] P. Vaidya. "Geometry Helps in Matching." In *Proc. ACM Symp. on the Theory of Computing*, pp. 422–425, 1988.

[Web89] S. Weber. "For VLSI, Multichip Modules May Become the Packages of Choice." *Electronics*, pp. 106–112, April 1989.

[WF83] D. F. Wann and M. A. Franklin. "Asynchronous and Clocked Control Structure for VLSI Based Interconnection Networks." *IEEE Trans. on Computers*, **21**(3):284–293, 1983.

[Wil61] W. L. Wilson. "On Discrete Dirichlet and Plateau Problems." *Numerische Mathematik*, **3**:359–373, 1961.

[Win87] P. Winter. "Steiner Problem in Networks: A Survey." *Networks*, **17**:129–167, 1987.

[WMM89] L.T. Wang, M. Marhoefer, and E.J. McCluskey. "A Self-Test and Self-Diagnosis Architecture for Boards Using Boundary Scans." In *Proc. European Test Conf.*, pp. 119–126, Paris, April 1989.

[Wol91] S. Wolfram. *Mathematica: A System for Doing Mathematics by Computer, Second Edition.* Addison-Wesley, Reading, MA, 1991.

[WWW86] Y. F. Wu, P. Widmayer, and C. K. Wong. "A Faster Approximation Algorithm for the Steiner Problem in Graphs." *Acta Informatica*, **23**(2):223–229, 1986.

[YCC91] S.-Z. Yao, N.-C. Chou, C.-K. Cheng, and T. C. Hu. "A Multi-Chip Module Substrate Testing Algorithm." In *Proc. IEEE Intl. ASIC Conf.*, pp. P9:4.1 – P9:4.4, Rochester, NY, September 1991.

[YW72] Y. Y. Yang and O. Wing. "Suboptimal Algorithm for a Wire Routing Problem." *IEEE Trans. on Circuit Theory*, **19**:508–511, 1972.

[Zel92] A. Z. Zelikovsky. "The 11/6 Approximation Algorithm for the Steiner Problem on Networks." *Information and Computation (to appear)*, 1992.