Approvals

This dissertation is submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Computer Science

Joseph L. Ganley

Approved:

James P. Cohoon (Advisor)

Jeffrey S. Salowe

Gabriel Robins

James M. Ortega (Chair)

James H. Aylor (Minor Representative)

Accepted by the School of Engineering and Applied Science:

Richard W. Miksad (Dean)

May 1995

Geometric Interconnection and Placement Algorithms

A Dissertation

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia

In Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

Computer Science

by

Joseph Lavinus Ganley

May 1995

For my wife, Cathy

Without her love, patience, and support, this dissertation would never have existed.

Acknowledgments

The first person to thank, of course, is my advisor, Jim Cohoon. His guidance, advice, and support have been invaluable throughout my doctoral studies. I also thank Jeff Salowe, with whom I had many discussions on much of the material presented here. His input is reflected in many places in this research. Thanks also to Gabe Robins, Jim Ortega, and Jim Aylor for serving on my dissertation committee and for many helpful comments on this work.

In addition, I am grateful to the following people for helpful comments on various bits and pieces of this research: Mike Alexander, Shekhar Bapat, Hans Bodlaender, Mike DeLong, David Eppstein, Mike Fellows, Shaodi Gao, Andrew Grimshaw, Lenny Heath, Alexander Hulpke, Andrew Kahng, John McKay, R. Ravi, Dana Richards, Éva Tardos, Clark Thomborson, Jim Varanelli, and the anonymous referees of our papers. My apologies to the others who I have doubtless forgotten to mention.

Special thanks to Lenny Heath, my M.S. advisor, who taught me much about the fundamentals of good research and good writing.

Thanks to Michel Berkelaar, Steve Brown, Matthew Saltzman, Jeff Salowe, Clark Thomborson, and Dave Warme for providing me with various pieces of code.

During my doctoral studies, I have received financial support from the National Science Foundation under grants MIP–9107717, CCR–9224789, and CDA–8922545, from the Virginia Center for Innovative Technology under award 5–30971, and from a University of

Virginia Dean's Fellowship and a Virginia Space Grant Fellowship. Their support is greatly appreciated.

This document was prepared using LaTeX version 2_ε. Some figures were prepared using Examine [9], FrameMaker^{TM}, and GNUPLOT.

Contents

List of Figures

List of Tables

Abstract

This dissertation examines a number of geometric interconnection, partitioning, and placement problems arising in the field of VLSI physical design automation. In particular, many of the results concern the geometric Steiner tree problem: given a set of terminals in the plane, find a minimum-length interconnection of those terminals according to some geometric distance metric.

Two new algorithms are introduced that compute optimal rectilinear Steiner trees. Both are provably faster than any previous algorithm for instances small enough to solve in practice, and both are also fast in practice. The first algorithm is a dynamic programming algorithm based on decomposing a rectilinear Steiner tree into full trees. A full tree is a Steiner tree in which every terminal is a leaf. Its time complexity is $O(n3^n)$, where n is the number of terminals. The second algorithm modifies the first by the use of full-set screening, which is a process by which some candidate full trees are eliminated from consideration. Its time complexity is approximately $O(n^2 2.62^n)$. We demonstrate that for instances small enough to reasonably solve in practice, these algorithms are faster than has been proven for any previous algorithm. Empirical evidence is also given indicating that both algorithms are faster in practice than several other popular algorithms for computing optimal rectilinear Steiner trees.

A new theorem is proven that allows efficient computation of rectilinear Steiner trees in the presence of obstacles. This theorem leads to the first algorithms that compute optimal obstacle-avoiding rectilinear Steiner trees in time corresponding to the size of the

instance (the number of terminals and obstacle border segments) rather than the size of the routing area. Two types of algorithms are presented. The first computes optimal rectilinear Steiner trees in the presence of obstacles for small instances. The second computes heuristic solutions to arbitrary instances. In addition, two special cases are examined: the case when all terminals lie on obstacle perimeters and the case when all terminals lie on the perimeter of the routing region.

The power-p Steiner tree problem is introduced, in which the weight of an edge is its geometric length raised to the p power. A special case of the power-p Steiner tree problem is also considered: the bottleneck Steiner tree problem, which is to find a geometric Steiner tree with the length of the longest edge minimized. Algorithms are described for computing optimal optimal Euclidean power-2 Steiner trees, rectilinear bottleneck Steiner trees, and rectilinear Steiner trees that minimize a combination of bottleneck weight and total length. The power-p Steiner tree problem for $p > 2$ is also considered, and evidence given that the problem is essentially unsolvable for large p. In addition, bounds are given on the power-p Steiner ratio, which measures the quality of a minimum spanning tree as an approximation of a power-p Steiner tree. The exact value of the bottleneck Steiner ratio is also derived, and a fully polynomial-time approximation scheme is given for computing a bottleneck Steiner tree with a given topology in any distance metric.

Finally, a system called MONDRIAN is presented that performs simultaneous placement and global routing in field-programmable gate arrays (FPGAs). MONDRIAN is a recursive geometric partitioning strategy using optimal rectilinear Steiner trees of terminals that lie in a small grid. Experimental results show that MONDRIAN produces results significantly superior to those of previous FPGA layout algorithms. In addition, we present a modified version of MONDRIAN that computes performance-driven placement and routing solutions, in which more accurate estimates of electrical delay are optimized. The development of MONDRIAN includes several results of independent interest, such as minimum-congestion

routing in a cycle and computing optimal rectilinear Steiner trees for terminals in a small grid.

1

Introduction

VLSI physical design automation is the process of translating an electrical description of a circuit into a physical layout on the surface of an integrated circuit [123, 135]. From the physical design automation domain arises a number of difficult algorithmic and combinatorial problems. The ability to compute good solutions to these problems is crucial to the production of high-performance integrated circuits at a reasonable cost.

The overall domain of VLSI design is a *circuit*, which is an electronic implementation of a Boolean function. Prior to the physical design phase of the design process, the description of a circuit consists of *cells* and *nets*. The cells are electronic devices that implement the operators (e.g., AND, OR, and NOT) in the Boolean function being implemented. A cell is typically a pre-packaged unit consisting of a small number of electrical components (for example, a single cell might be an AND gate or a flip-flop). For purposes of physical design automation, a cell can be viewed simply as a polygon (typically a rectangle) on whose perimeter lies a number of *pins*. The pins are the inputs and outputs to the cell. A special type of cell is an *input/output pad*, which typically must lie on the perimeter of the circuit and connects it to the outside, e.g. to other integrated circuits.

Cell pins are connected to one another by *wires*, which carry digital *signals* between the cells (Boolean operators) to implement the semantics of the Boolean expression. Prior

to the physical design phase of the design process, these interconnections are represented as *nets*. Each net specifies a set of cell pins that must be electrically interconnected. The wires that implement the nets cannot, of course, intersect the interiors of the cells nor the wires in other nets.

The portions of the physical design process with which we are concerned are the tasks of placement and routing. The *placement* process assigns the cells to physical locations on the surface of the integrated circuit. The *routing* process realizes the electrical interconnections specified by the nets as wires on the integrated-circuit surface.

The input to the placement and routing process consists of a number of cells (typically hundreds or thousands) and a number of nets specifying the electrical interconnections among them.

The flexibility allowed in the physical design process varies according to the *design style* of the circuit. At one end of the spectrum—the *full custom* design style—the placement and routing of the cells may be completely unrestricted. This design style allows great flexibility and very efficient use of circuit area, but increases the difficulty of the physical design process and the cost of producing these integrated circuits.

A slightly more restricted model is the *standard cell* design style, in which all cells are rectangular and must be placed such that they abut one another in rows, with the routing performed within the *channels* between the rows of cells.

A still more restricted model is the *gate array* design style, where identical prefabricated logic cell templates are laid out on a circuit surface prior to physical design. The cell templates may be laid out in rows like a standard cell design, or in a grid. The gate array design style simplifies the physical design process still further; now the placement process consists simply of assigning the unplaced cells to the cell templates.

The gate array concept is taken yet another step further in the *field-programmable gate array (FPGA)* design style [48]. In an FPGA, even the routing resources are prefabricated

on the circuit surface. A given design is implemented by electrically programming the wires between the cell templates to realize the desired electrical interconnections.

Most of the results in this dissertation focus on the routing phase of the physical design process. In the routing phase, the electrical interconnections specified by the nets must be realized as to minimize a number of objectives. These objectives typically optimize two factors: the cost of producing the integrated circuit and the performance of the circuit. The objective toward the minimization of production cost is typically to minimize circuit area. For custom and standard-cell design styles, this amounts to minimizing the overall area of the circuit. For prefabricated design styles like gate arrays, minimizing cost amounts to minimizing the amount of prefabricated circuit resources that are required to implement the circuit. Minimizing the amount of logic resources required is typically handled prior to the physical design process, so the primary cost objective in physical design of gate arrays is to minimize the amount of routing resources required to implement the circuit. The cost of a gate array varies according to the amount of logic and routing resources it contains, so minimizing these resource requirements minimizes the cost of implementing the integrated circuits.

The second objective to be optimized in physical design automation is circuit performance. Here, *electrical delay*—the time it takes signals to propagate through the circuit—is of prime importance. Electrical delay in a circuit is contributed from two sources: delay within the logic cells and interconnect delay. For all but the most fully custom design styles, delay within the logic cells is not within the control of the physical design process. On the other hand, interconnect delay is crucially related to the quality of placement and routing solutions. Furthermore—one of several reasons why physical design automation is more critical than it once was—as integrated-circuit feature sizes decrease, overall circuit delay is increasingly governed by interconnect delay rather than delay within the logic cells [7].

Typically, integrated-circuit technology requires the use of only horizontal and vertical wires. This restriction prescribes the use of the rectilinear distance metric that measures

distance in this manner [71]. (The rectilinear distance between two points is the sum of the x-distance and the y-distance between the points.) The precise routing objectives that minimize electrical delay vary with the particulars of the VLSI technology being used. In most technology to date, electrical delay through a net is directly proportional to the total length of the wires in the interconnection of that net, and thus a common routing objective is for the total length of each net to be minimized. This objective prescribes the use of *Steiner trees* for routing. Steiner trees are described further in Chapter 2, but for the present, suffice it to say that a Steiner tree is a minimum-length geometric interconnection of a set of points (logic cell pins). A close relative of the Steiner tree is a *minimum spanning tree*, which is a minimum-length interconnection in which the pins can only be connected to one another. For example, Figure 1.1 shows four pins and a rectilinear minimum spanning tree and rectilinear Steiner tree of these pins. In the figure, as is often true, the Steiner tree is a shorter interconnection than the minimum spanning tree, due to the use of additional points called *Steiner points* (the open circle in Figure 1.1(a)).

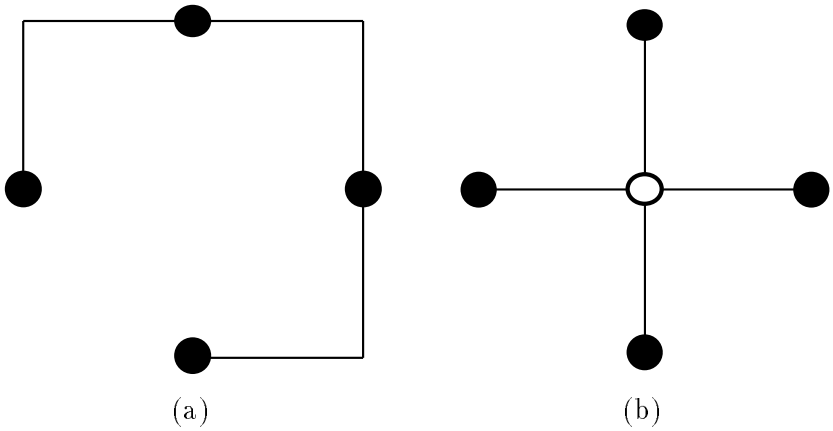


Figure 1.1: (a) A rectilinear minimum spanning tree and (b) a rectilinear Steiner tree of an example set of pins.

In some design styles, multiple layers are available for routing. Often, nets whose electrical delay is critical to the overall timing of the circuit will be routed first in their own layer. Thus, the routing problem is exactly the Steiner tree problem: we wish to find a minimum-length interconnection of the set of logic block pins, and there are no obstacles that the interconnection must avoid. In Chapter 3, we present two new algorithms for computing optimal rectilinear Steiner trees. As suggested by the fact that the rectilinear Steiner tree problem is NP-complete [63], the algorithms we present require time that is exponential in the number of pins to be interconnected. However, the instances that arise in VLSI routing typically contain few pins (this claim is substantiated in Section 4.3.2), so an exponential-time algorithm that is fast in practice for small instances can be practically applied to VLSI routing problems. Our algorithms are based on a dynamic programming approach to finding an optimal decomposition of the set of terminals into *full sets*, or subsets of terminals whose corresponding subtree in an optimal Steiner tree contains no terminals of degree 2 or greater. The algorithms we present are faster than any other practically applicable algorithm has been proven to be. Furthermore, we give experimental results indicating that the algorithms are fast in practice for the small instances typical of those arising in VLSI routing.

Nets whose electrical delay is not so critical to overall circuit performance often must be routed in the presence of obstacles such as logic cells or wires in previously routed nets. It is still appropriate to compute a minimum-length Steiner-tree interconnection, but the presence of obstacles leads to a new constraint: we must compute Steiner trees that do not intersect the obstacles. For example, Figure 1.2 illustrates a rectilinear Steiner tree in the presence of obstacles. This problem is addressed in Chapter 4. Specifically, we present results that allow, for the first time, computation of optimal Steiner trees in the presence of obstacles in time that is a function of the size of the instance (the number of pins and obstacle border segments). All previous algorithms require time that is a function of the routing area, which is prohibitively large for small feature sizes and large circuit area. We

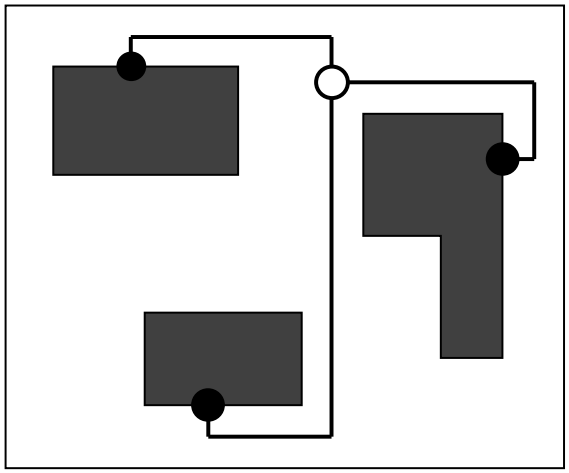


Figure 1.2: A rectilinear Steiner tree in the presence of obstacles.

present fast algorithms for computing optimal and near-optimal Steiner trees in the presence of obstacles. We also examine two special cases that have practical significance: when the pins all lie on the borders of obstacles (as is often the case in practice) and when the pins all lie on the perimeter of the routing region.

Advances in VLSI technology, particularly decreasing feature size, change the character of the physical design process, in the sense that the relationship between electrical delay and interconnect length becomes more tenuous [91]. As a result, new measures of interconnect quality have been devised that more accurately reflect their electrical delay in such technologies.

One such measure is accomplished by minimizing a nonlinear function of the lengths of the wires in the interconnect. For example, a first-order approximation of electrical delay under many technologies is the product of the resistance and capacitance of a wire [34]. If the width of the wire is fixed, then this product is a quadratic function in the length of the wire. Consideration of these types of objective inspired us to introduce the power-p

Steiner tree problem, which is the subject of Chapter 5. The power-p Steiner tree problem is to find a Steiner tree that, rather than minimizing the sum of the lengths of the wires in the interconnect, minimizes a nonlinear function of their lengths. A special case with practical significance is also examined, where the objective is to minimize the length of the longest wire in the interconnect [23, 133]. In Chapter 5, we present a number of results concerning computation of optimal and approximate power-p Steiner trees. Specifically, we present algorithms for computing optimal Euclidean power-2 Steiner trees, optimal rectilinear bottleneck Steiner trees, and rectilinear Steiner trees that minimize a combination of bottleneck weight and total length. We also present results concerning the quality of a minimum spanning tree as an approximation of a power-p Steiner tree.

Field-programmable gate array (FPGA) technology [48] allows inexpensive prototyping and implementation of custom integrated-circuit designs. An FPGA is a prefabricated integrated circuit consisting of electrically reprogrammable logic and routing resources. Once the physical design of an FPGA has been computed, it can be directly programmed into the FPGA without the use of manufacturing or similarly expensive and time-consuming processes. Since FPGAs can be produced quickly and inexpensively, this technology is well on the way to revolutionizing the production of integrated circuits for custom and rapid-prototyping applications [17]. Furthermore, the reprogrammable nature of FPGAs shortens the design cycle considerable, making it more similar to software design than to more traditional VLSI design processes.

Continuing the software design analogy, design automation software is analogous to compilers for high-level programming languages. Good physical design automation software for FPGAs can be even more crucial than for other design styles, for two reasons. The first is that the high degree of flexibility offered by FPGAs comes at the cost of decreased performance [3]. As FPGAs are increasingly used to actually implement integrated circuits (rather than, for example, simply prototyping them), it becomes increasingly important to optimize their performance as well as possible at the physical design level. The second reason

physical design automation software is so crucial is that the low cost and high ease of use of FPGAs makes them accessible to a far larger user community than more custom integrated-circuit technologies. Most users of more custom integrated-circuit technology have, or have access to, professional layout designers who can manually compensate for the shortcomings of their physical design automation software. FPGAs, however, are increasingly used by people who do not have access to such resources, and thus it is vital that software exists that can automate the entire physical design process, producing high-quality physical designs without requiring any manual intervention.

In Chapter 6 we present a tool that simultanously computes high-quality placement and routing solutions for FPGAs. The tool is based on a recursive partitioning strategy. In this strategy, the FPGA is overlaid with a grid. The subproblems within each region of the grid are solved recursively, and the solutions to these subproblems are then merged to form a solution for the entire FPGA. We show that our tool produces higher-quality FPGA physical designs than those produced by previous tools. Furthermore, the development of the tool includes many algorithmic results that are of independent interest.

2

The Steiner Tree Problem

In this chapter the Steiner tree problem, to which all the results in this dissertation are related, is formally defined. Some notation and terminology is presented and a number of background results on the Steiner tree problem are given, which are used throughout the dissertation.

2.1 Notation

If a is a point in $\mathcal{R}^2$, then its x and y coordinates are denoted x_a and y_a, respectively. The distance in some L_p metric between points a and b is denoted $\|a-b\|_p$. The subscript indicating the distance metric is omitted when it is clear from context. Note that in the rectilinear (L_1) metric,

$$\|a-b\|_1 = |x_a - x_b| + |y_a - y_b|,$$

and in the Euclidean (L_2) metric,

$$\|a-b\|_2 = \sqrt{(x_a - x_b)^2 + (y_a - y_b)^2}.$$

The length of an edge $e = (a,b)$ is $\|e\| = \|a-b\|$, and the length of a tree τ is $\|\tau\| = \sum_{e \in \tau} \|e\|$.

2.2 Steiner Trees

All of the problems examined in this dissertation are related to the *Steiner tree* problem. This section describes some concepts and terminology related to the Steiner tree problem.

As was mentioned above, a typical objective in routing is to route each net using a minimum total length of wire. Such a routing problem can be abstracted to the following: given a set of points in the plane, find a short interconnection of those points. It is easy to see that such an interconnection will be a tree, since if it contained a cycle then some wire could be removed and the length reduced while still maintaining electrical connectivity.

One type of short interconnection is a *minimum spanning tree (MST)*. An MST is a shortest interconnection that uses only wires between the given points, which are called *terminals*. The MST problem in an undirected, weighted graph G is defined as follows: find a minimum-weight subtree of G that spans all the vertices. MSTs are efficiently computed by algorithms such as those of Kruskal [101] and Prim [124].

An MST of a selected set of vertices in a graph G is modeled by a complete graph on these vertices, in which each edge corresponds to a shortest path between its endpoints in G.

While an MST is a short interconnection—typically within a constant factor of the best possible—generally a shorter interconnection can be realized using a Steiner tree. The basic version of the Steiner tree problem is the *graph Steiner tree (GST)* problem. An instance of the GST problem consists of an undirected, weighted graph $G = (V, E)$ and a subset of the vertices $T \subseteq V$ that are identified as *terminals* (the vertices in $V - T$ are called *nonterminals*). The GST problem is to find a minimum-weight subtree of G that contains all the terminals[1]. Equivalently, the problem is to find a set S of *Steiner points*, where $S \subseteq V - T$, such that the length of an MST of $T \cup S$ is minimized. For example, Figure 2.1 illustrates a graph and an MST and optimal Steiner tree for that graph. As will be our convention

[1] If edges can have negative weight, then the word "tree" is something of a misnomer, as the minimum-weight subgraph containing the terminals may not be a tree. However, in all the applications studied here, edges have nonnegative weight, so we retain this nomenclature.

(c)

) A graph, (b) an MST with weight 4, and (c) a Steiner tree with

---

is the *shortest path* problem. Both, of course, are easily solvable in However, the general GST problem is NP-complete [93], indicating that a gorithm to solve exactly it is unlikely to exist. Furthermore, it remains if G is planar or if all edge weights are equal.

the basic Steiner tree problem is the *geometric Steiner tree* problem

variations have been proven to be NP-complete as well (by Garey, Graham, and Johnson [62] and Garey and Johnson [63], respectively).

2.2.1 The Steiner ratio

Much attention has been given to the quality of an MST as an approximation of an optimal Steiner tree for various versions of the Steiner tree problem. The *Steiner ratio*, denoted ρ, is defined for a particular version of the Steiner tree problem to be the maximum, over all instances, of the ratio of the length of an MST to the length of an optimal Steiner tree. That is, if I is a Steiner tree instance, $M(I)$ is an MST of I, and $\tau(I)$ is an optimal Steiner tree of I, then

$$\rho = \max_I \frac{\|M(I)\|}{\|\tau(I)\|}.$$

For most Steiner tree problems, ρ is a small constant, indicating that an MST is a good approximation of an optimal Steiner tree.

For example, for the GST problem, $\rho = 2$. The lower bound is witnessed by a "star" graph consisting of a single nonterminal vertex of degree n connected by unit-weight edges to n terminal vertices of degree 1. The length of an MST of this graph is $2n - 2$, and the length of an optimal Steiner tree is n. Thus, this graph witnesses $\rho \geq 2 - 2/n$. If n is large, then ρ is arbitrarily close to 2. For the upper bound, consider an optimal Steiner tree τ in any graph G. Replace each edge with a pair of parallel edges, and construct an Euler tour of τ. An Euler tour of a graph G is a tour that visits every edge in G exactly once, returning to the vertex from which it started. An Euler tour of the augmented version of τ has length $2\|\tau\|$. Removing the longest edge gives us a spanning tree, which can be no shorter than the MST. This longest edge has length at least $2\|\tau\|/n$, so the length of the MST is at most $2\|\tau\| - 2\|\tau\|/n$. Thus, $\rho \leq 2 - 2/n$; again, ρ is arbitrarily close to 2 if n is large.

For geometric Steiner tree problems, ρ is often smaller than 2. For the EST problem, $\rho = 2/\sqrt{3}$ (this fact was conjectured by Gilbert and Pollak in 1966 [65] and finally

proven in 1992 by Du and Hwang [45]). For the RST tree problem, Hwang [82] proved that $\rho = 3/2$.

2.2.2 Rectilinear Steiner trees

Most of the results in this dissertation concern the RST problem. The RST problem naturally arises in VLSI routing applications because typically VLSI fabrication technology requires all wires to be either horizontal or vertical. Figure 2.2 illustrates an optimal RST for a set of 27 terminals (terminals are depicted as filled circles and Steiner points as open circles). The severely constrained nature of the rectilinear distance metric leads to several

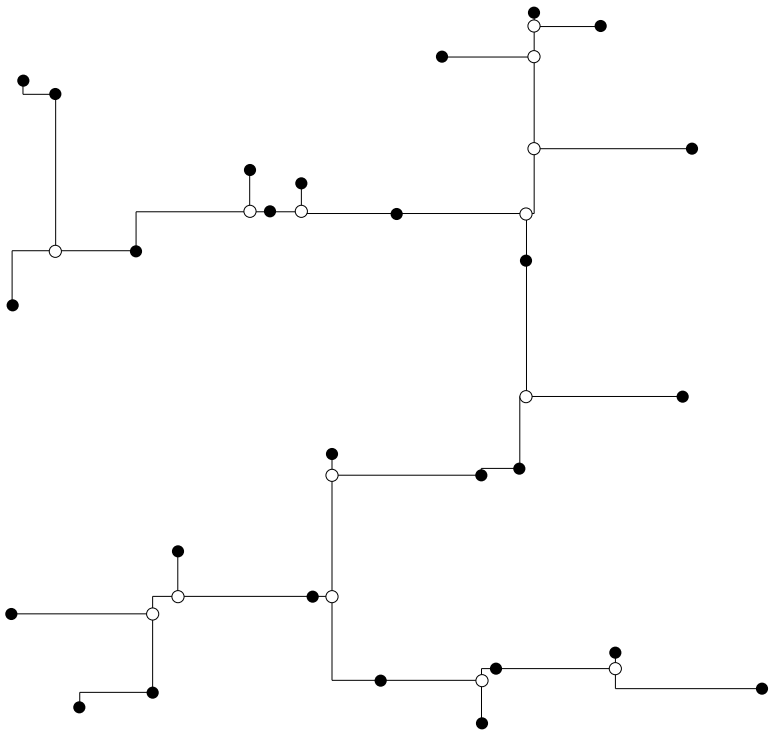


Figure 2.2: An optimal RST for a set of 27 terminals.

important geometric results.

The first (and earliest) is *Hanan's theorem.*

Theorem 2.1 (Hanan [71]) *For every set of terminals, there exists an optimal RST in which every Steiner point shares both its x- and y-coordinates with some pair of terminals.*

Hanan's theorem implies that the following graph, called the *Hanan grid graph*, always contains an optimal RST. Draw horizontal and vertical *grid lines* through each terminal. Construct a graph $G = (V, E)$ in which each vertex in V corresponds to the intersection of two grid lines (note that the intersections include the terminals). There is an edge between two vertices if they are adjacent along a grid line, and the weight of each edge is the rectilinear distance between its endpoints. Hanan's theorem implies that an optimal solution to the GST problem in G is an optimal solution to the RST problem from which it was constructed. Figure 2.3 illustrates the Hanan grid graph for the RST instance of Figure 2.2 with the optimal RST darkened. Hanan's theorem allows computation of an optimal RST by applying an algorithm for the GST problem to G, though as shown in Chapter 3, more efficient solutions are obtained by considering the problem in a more geometric fashion.

A set T of terminals is a *full set* if in every optimal Steiner tree of T, every terminal in T is a leaf (i.e., has degree 1). A Steiner tree of a full set is called a *full tree.* An important result concerning full sets in the rectilinear metric is *Hwang's theorem.*

Theorem 2.2 (Hwang [82]) *An optimal full tree can have only one of the following two topologies. A* type I *topology consists of a* backbone *segment adjacent to one of the extreme terminals, with segments connecting the other terminals to the backbone. (Assume without loss of generality that the backbone is horizontal.) From left to right, these terminals must appear on alternating sides of the backbone. A* type II *topology is similar to a type I topology, but with the leftmost (or rightmost) terminal connected to the segment that connects the second terminal from the left (or right) to the backbone.*

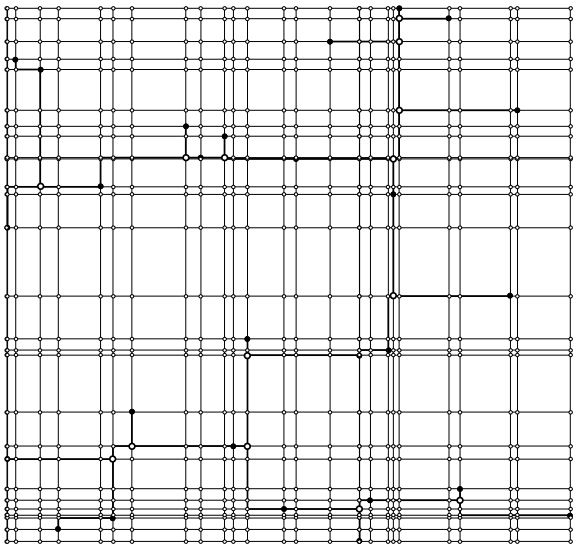


Figure 2.3: The Hanan grid graph and optimal RST for a set of 27 terminals.

The two topologies are illustrated in Figure 2.4. Using Hwang's theorem, an optimal RST of a full set is computed in linear time by examining these two topologies[2].

Finally, a well-known decomposition theorem on Steiner trees is the following.

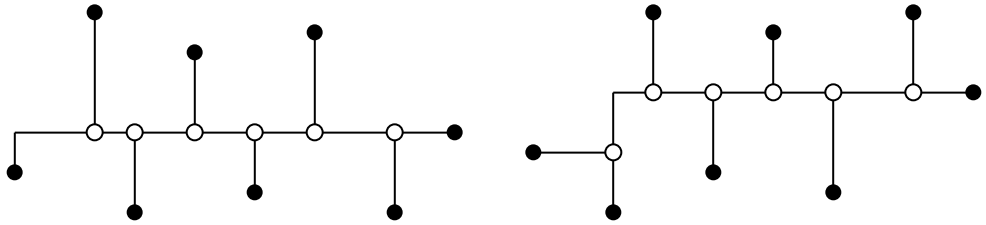
Theorem 2.3 *Every Steiner tree is composed of a number of full trees that intersect at terminals of degree* 2 *or greater.*

(See Hwang, Richards, and Winter [84].)

2.3 Overview

The remainder of this dissertation is organized in four chapters, each of which examines a different geometric problem related to VLSI design automation.

[2]As an aside, we know of no other class of graph or distance metric in which an optimal full tree is computable in polynomial time but an optimal Steiner tree is not. This is an interesting topic for future research.



Type I Type II

Figure 2.4: Possible full tree topologies according to Hwang's theorem.

Chapter 3 presents two new algorithms for computing optimal RSTs. The first, called Full-set Dynamic Programming, involves a bottom-up construction of an optimal RST from its component full sets. Its time complexity is $O(n3^n)$ for n terminals. The second algorithm, called Screened Full-set Dynamic Programming, augments the first algorithm with the addition of full set screening, which eliminates many candidate full sets from consideration. This modification, in combination with a proof of an upper bound on the number of candidate full sets, results in a time complexity of approximately $O(n^2 2.62^n)$. We show that these algorithms have lower time complexity than has been proven for any practically applicable algorithm and that in practice they compare favorably with several other popular algorithms for computing optimal RSTs.

Chapter 4 is concerned with computing optimal and heuristic RSTs in the presence of obstacles that the RST must not intersect. Specifically, we prove a result analogous to Hanan's theorem (Theorem 2.1 on page 14) that enables computation of optimal obstacle-avoiding RSTs in time corresponding to the instance size. We then present algorithms for computing optimal and heuristic RSTs in the presence of obstacles, and examine two special cases of the problem.

Chapter 5 introduces the power-p Steiner tree problem, which is to compute a geometric Steiner tree that minimizes the sum of the edge weights each raised to the p power. We

also examine a special case of the power-p Steiner tree problem: the bottleneck Steiner tree problem, which is to find a geometric Steiner tree that minimizes the length of the longest edge. We give algorithms for computing optimal Euclidean power-2 Steiner trees, optimal rectilinear bottleneck Steiner trees, and rectilinear Steiner trees that minimize a combination of bottleneck weight and total length. We also examine the power-p analogue of the Steiner ratio, which measures the quality of an MST as an approximation of a power-p Steiner tree. We give bounds on the general power-p Steiner ratio and we derive the exact value of the bottleneck Steiner ratio. We also consider some other aspects of the problem, such as computing power-p Steiner trees for p larger than 2 and computing good approximate bottleneck Steiner trees.

Chapter 6 describes a system called MONDRIAN that performs simultaneous placement and global routing of field-programmable gate arrays. MONDRIAN uses a recursive geometric partitioning strategy based on optimal rectilinear Steiner trees of terminals that lie in a small grid. The use of this strategy, implemented using a number of algorithmic techniques, allows minimization of channel width, total wire length, and maximum source-sink path lengths. MONDRIAN is shown to produce results superior to those of previous algorithms. The development of MONDRIAN also includes some algorithmic results of independent interest, such as minimum-congestion routing in a cycle and computing optimal rectilinear Steiner trees for terminals in a small grid.

3

Computing Optimal Rectilinear Steiner Trees

As mentioned in Section 2.2, the *rectilinear Steiner tree (RST)* problem is NP-complete [63]. This suggests that a polynomial-time algorithm that computes an optimal RST is unlikely to exist. As with most NP-complete problems, one avenue of research is to devise exponential-time algorithms that compute optimal solutions to small instances. Such algorithms are particularly pertinent in VLSI routing, since a typical net contains only a few terminals (see Section 4.3.2). One objective of such research is to devise algorithms that have good asymptotic time complexity and that can solve as large an instance as possible in practice.

3.1 Overview

This chapter presents two new algorithms for computing optimal RSTs. The first runs in $O(n3^n)$ time, where n is the number of terminals. This algorithm is not only faster than commonly used algorithms, but is very simple and easy to implement. The second algorithm improves on the first by using the concept of full-set screening. This second algorithm runs in at most $O(n^2(1+\phi)^n)$ time, where $\phi = (1+\sqrt{5})/2 \approx 1.62$. For instances

Earlier versions of portions of this chapter appear in Ganley and Cohoon [54, 56].

that can reasonably be solved in practice, we show that these bounds are better than those of any previous algorithm.

The remainder of this chapter is organized as follows. Section 3.2 surveys algorithms for computing optimal RSTs, including a few graph-based algorithms and many geometric algorithms. Section 3.3 then describes the first of our algorithms, which is called Full-set Dynamic Programming and runs in $O(n3^n)$ time. Sections 3.4 and 3.5 introduce the concept of full-set screening and prove upper bounds on the number of possible full sets. Section 3.6 then describes the second, $O(n^2 2.62^n)$ algorithm, called Screened Full-set Dynamic Programming. Section 3.7 discusses the relevance of asymptotic time complexity to instances that can be solved in practice. In particular, we show that for instances small enough to solve in practice, the running times of our algorithms are faster than the best known time bounds of any algorithm. Section 3.8 gives experimental results comparing our algorithms with some other popular algorithms for computing optimal RSTs. Finally, Section 3.9 summarizes and describes directions for current and future work.

3.2 Previous Work

A number of algorithms for computing optimal RSTs have appeared in the literature. The RST problem can be solved either by reducing it to the *graph Steiner tree (GST)* problem as described in Section 2.2.2 or by solving it more directly in a geometric fashion. Here we discuss a few algorithms for the GST problem and their performance when used to solve the RST problem. We then survey geometric algorithms for finding optimal RSTs.

As discussed in Section 2.2.2, one can use Hanan's theorem to construct a graph G from an instance of the RST problem such that an optimal solution to the GST problem in G is an optimal solution to the RST instance from which it was constructed.

An early algorithm for the GST problem is Hakimi's spanning tree enumeration algorithm [70], which has time complexity $O(n2^{n^2-n}\log n)$ when applied to G. Hakimi's algorithm solves 8-terminal problems in less than a day on a workstation.

The most provably efficient algorithm for solving the GST problem is the dynamic programming algorithm of Dreyfus and Wagner [43], which has time complexity $O(n^2 3^n)$ when applied to G. Its time and space requirements restrict it to solving instances with 16 or fewer terminals on a workstation. Thomborson, Alpern, and Carter [145] present some improvements to the Dreyfus-Wagner algorithm that do not change the algorithm's time complexity, but do improve its efficiency in practice—their algorithm solves 20-terminal problems in a day on a workstation.

A large number of other algorithms exist for the GST problem, but they are either provably less efficient than the Dreyfus-Wagner algorithm, or else good bounds on their time complexity are elusive, and there exist no investigations of their performance when applied to G. The reader is referred to Hwang, Richards, and Winter [84] for further discussion of algorithms for the GST problem.

A number of other algorithms have been devised that solve the RST problem in a geometric fashion, without explicit use of the Hanan grid graph.

Yang and Wing [158] present a branch-and-bound algorithm with worst-case complexity $O(2^{n^2})$; the largest instance on which they test their algorithm contains 9 terminals.

Wong and Pecht [154] describe an exact RST algorithm that is essentially an exhaustive version of the edge-embedding heuristic of Ho, Vijayan, and Wong [75]. The time complexity of the algorithm is a summation for which no closed form is given, but it grows slightly faster than $O(4^n)$. Their algorithm solves roughly 15-terminal instances [153].

Lewis, Pong, and Van Cleave [109] use some geometric properties of optimal RSTs to devise an algorithm with time complexity $O(4^{n \log n}/n^4)$, which solves 10-terminal instances in a day.

Sidorenko [137] describes an exact RST algorithm. He proves some geometric and topological properties of optimal RSTs, and uses them to devise an $O(n!)$ algorithm. He states that the algorithm is is applicable to instances with up to 11 terminals[1].

[1] Thanks to Masha Sosonkina for translating the paper.

Smith [139] presents an algorithm with worst-case time complexity $n^{O(\sqrt{n})}$, which is asymptotically faster than any $\Omega(c^n)$ algorithm for constant $c > 1$. However, the algorithm is not practically applicable due to tremendous constant factors in its time complexity. Smith's algorithm is discussed further in Section 3.7.

Salowe and Warme [131] present an algorithm that works well in practice—it is applicable to 35-terminal instances—but the only previously known bound on its worst-case time complexity is $O(2^{2^n})$. Our results in Section 3.5 improve this bound slightly to $O(2^{n1.62^n})$.

Authors	Time	n_{day}
Dreyfus and Wagner [43]	$O(n^2 3^n)$	16
Hakimi [70]	$O(n2^{n^2-n}\log n)$	8
Lewis et al. [109]	$O(4^{n\log n}/n^4)$	10
Salowe and Warme [131]	$O(2^{n1.62^n})$	35
Sidorenko [137]	$O(n!)$	11
Smith [139]	$n^{O(\sqrt{n})}$	—
Thomborson et al. [145]	$O(n^2 3^n)$	20
Wong and Pecht [154]	$O(a^n), a \approx 4$	15
Yang and Wing [158]	$O(2^{n^2})$	9
FDP	$O(n3^n)$	22
SFDP	$O(n^2 2.62^n)$	27

Table 3.1: Summary of algorithms for computing optimal RSTs. The column labeled n_{day} indicates the problem size that each algorithm solves in 24 hours on a workstation. FDP and SFDP are the algorithms described in this chapter.

3.3 Full-set Dynamic Programming

The first of our algorithms is called *Full-set Dynamic Programming (FDP)*. The FDP algorithm is very simple and easily implemented and runs in $O(n3^n)$ time. This improves,

both theoretically and practically, on the Dreyfus-Wagner algorithm, which is often used in practice to compute optimal RSTs [41, 42, 98, 145].

Theorem 2.3 implies that an optimal RST for every set of terminals is either a full tree satisfying Hwang's theorem (Theorem 2.2 on page 14), or else it can be divided into two optimal subtrees joined at a terminal. This observation leads to the FDP algorithm. Subsets of the input set of terminals are enumerated in order of increasing cardinality. For each subset S, the algorithm compares the length of the full tree produced by applying Hwang's theorem to the lengths of the trees produced by joining the optimal RSTs of every pair of subsets A and B such that $A \cup B = S$ and $|A \cap B| = 1$. (Henceforth we write $A \bowtie B = S$ if $A \cup B = S$ and $|A \cap B| = 1$.) The decomposition with minimum length is an optimal RST for the set S of terminals. Since the subsets are enumerated in order of increasing cardinality, at each step the optimal RSTs for the smaller subsets A and B have already been computed and stored.

Figure 3.1 formally describes the FDP algorithm. As shown, the algorithm computes

```
(1)  For m = 2 to |T|
(2)      For all S ⊆ T such that |S| = m
(3)          ℓ[S] = ‖H(S)‖
(4)          For all A, B such that A ⋈ B = S
(5)              ℓ[S] = min{ℓ[S], ℓ[A] + ℓ[B]}
```

Figure 3.1: The FDP algorithm. T is the set of input terminals, and $H(S)$ is an optimal full tree of a set S of terminals.

only the length of the optimal tree; a similar top-down pass computes the actual tree given the lengths computed by the first pass.

The time complexity of the FDP algorithm is derived in a manner similar to that of Dreyfus and Wagner [43]. For each value of m in loop (1) in Figure 3.1,

- Loop (2) iterates $I_2 = \binom{n}{m}$ times.

◇ Loop (4) iterates $I_4 = m2^{m-1}I_2$ times.

Thus, the time complexity of the FDP algorithm is

$$\sum_{m=2}^{n} \binom{n}{m} m2^{m-1} \leq n3^{n-1}.$$

This analysis does not include line (3) of Figure 3.1, in which a full tree is computed for every subset of T, each in linear time, resulting in an additional $O(n2^n)$ time. However, this term is dominated by the decomposition term, so the overall asymptotic time complexity is $O(n3^n)$. This improves upon the Dreyfus-Wagner algorithm, which has time complexity $O(n^2 3^n)$.

It is also noteworthy that the space complexity of the FDP algorithm is $O(2^n)$, whereas the space complexity of the Dreyfus-Wagner algorithm is $O(n^2 2^n)$ for the RST problem. In practice, the space requirements of the Dreyfus-Wagner algorithm are more restrictive than its time complexity [13], and it cannot solve problems much above 16 terminals within the main memory of a workstation. The Thomborson, Alpern, and Carter [145] optimization to the Dreyfus-Wagner algorithm manages memory explicitly on disk, but even so, a 23-terminal problem requires about 400 megabytes of storage, so problems larger than this are currently infeasible on a typical workstation.

In practice, one might wish to store the actual decompositions of each subset along with its length. This modification does not change the time complexity of the algorithm, but speeds it up in practice by eliminating the need for computing the optimal lengths and the actual RST in two separate passes. This modification increases the space requirements to $O(n2^n)$, which is still a significant improvement over Dreyfus-Wagner.

3.4 Full-set Screening

The key concept in the Euclidean Steiner tree algorithms of Cockayne and Hewgill [28, 29] and Winter [149] and the RST algorithm of Salowe and Warme [131] is that of *full-set screening*. The idea is that relatively few subsets of the set of terminals can be full sets. Thus, several tests are applied to each subset to potentially eliminate it from candidacy as

a possible full set. For example, the subset must be connectable according to one of the topologies specified by Hwang's theorem.

Full-set screening can be used to improve the running time of the FDP algorithm. In the FDP algorithm, the innermost loop enumerates, for each subset S of terminals, all pairs of subsets A and B such that $A \bowtie B = S$. Having identified a number of candidate full sets, one can also require that A be a candidate full set while still satisfying Theorem 2.3. Thus, if the number of candidate full sets that are subsets of S is asymptotically smaller than the total number of subsets of S, and if the subsets of S that are candidate full sets can be efficiently enumerated, then the time complexity of the FDP algorithm is improved.

Before proceeding further, we define some additional notation. For every set S of terminals, let $F(S)$ denote the set of candidate full sets that are subsets of S. Note that if S is itself a candidate full set, then $F(S)$ includes S. The set of input terminals is T, so the complete set of candidate full sets is $F(T)$. Let $f(n) = \max_{|T|=n} |F(T)|$ denote the maximum number of candidate full sets over all sets of n terminals.

3.5 Bounding the Number of Full Sets

An important component of our analysis is proving an upper bound on $f(n)$ that is asymptotically smaller than $O(2^n)$. By using Hwang's theorem, we prove that $f(n) \leq O(n\phi^n)$, where $\phi = (1+\sqrt{5})/2 \approx 1.62$.

We first prove a pair of results regarding binary strings, and then use these results to derive the bound. These strings are described by regular expressions; readers not familiar with regular expression notation may refer to Hopcroft and Ullman [80]. Define a *string* S to be a sequence $s_1 s_2 \cdots s_m$, where $s_i \in \{\mathsf{a}, \mathsf{b}\}$, i.e. S is a string in the language $(\mathsf{a}+\mathsf{b})^*$. Define a *substring* of a string S to be $s_{i_1} s_{i_2} \cdots s_{i_k}$ where $1 \leq i_1 < i_2 < \cdots < i_k \leq m$. Each s_i in a string S is considered distinct; e.g., given a string $S = \mathsf{aaa}$, the substrings $s_1 s_2$ and $s_1 s_3$ are considered distinct even though they both have value aa. An *alternating string* is one in the

language $\mathsf{b}(\mathsf{ab})^*(\mathsf{a}+\epsilon)$, and an *anti-alternating string* is one in the language $\mathsf{a}(\mathsf{ba})^*(\mathsf{b}+\epsilon)$. (Note that the empty string ϵ is both alternating and anti-alternating.)

Lemma 3.1 *There are at most ϕ^m alternating (or anti-alternating) substrings of an alternating string of length m, where $\phi = (1+\sqrt{5})/2$.*

Proof: Let $A(m)$ denote the number of alternating substrings of an alternating string of length m. Similarly let $B(m)$ be the number of anti-alternating substrings of an anti-alternating string of length m. Let $S = s_1 s_2 \cdots s_m$ be an alternating string of length m. An alternating substring of S is defined recursively as either

1. s_1 concatenated with an anti-alternating substring of $s_2 s_3 \cdots s_m$, or
2. An alternating substring of $s_3 s_4 \cdots s_m$.

This observation yields the following recurrence:

$$\begin{aligned} A(m) &= B(m-1) + A(m-2), \\ B(m) &= A(m-1) + B(m-2), \\ A(1) &= 1, \quad \text{and} \\ B(1) &= 1. \end{aligned}$$

This recurrence solves to $A(m) = F_m < \phi^m$ (F_m is the m^{th} Fibonacci number). The number of anti-alternating substrings in a string S is equal to number of alternating substrings in the complement of S, so the same applies for anti-alternating substrings. □

We now show that the number of alternating substrings of a string S is maximized if S is itself alternating.

Lemma 3.2 *The number $A(m)$ of alternating substrings of an alternating string of length m is at least as large as the number of alternating substrings of any string of length m.*

Proof: The proof is by induction on m. The base case is $m = 1$. The alternating string b contains one alternating substring while the string a contains none. For the inductive

hypothesis, assume that an alternating string of length less than m contains at least as many alternating substrings as every string of equal length.

Let S be a string of length m, and assume that S is neither an alternating nor an anti-alternating string, so that it must contain a contiguous sequence C of the form aaa^* or bbb^* (without loss of generality assume that C is in bbb^*). Let W be the portion of S to the left of C, and let E be the portion of S to the right of C, like so:

$$S = \underbrace{((\mathsf{a}+\mathsf{b})^*\mathsf{a}+\epsilon)}_{W}\underbrace{\mathsf{bbb}\cdots\mathsf{bbb}}_{C}\underbrace{(\mathsf{a}(\mathsf{a}+\mathsf{b})^*+\epsilon)}_{E}.$$

Let $m_W = |W|$, let $m_C = |C|$, and let $m_E = |E|$, and assert that C is maximal in the sense that if W is nonempty then $w_{m_W} = \mathsf{a}$, and if E is nonempty then $e_1 = \mathsf{a}$. Assume that the number $A(S)$ of alternating substrings of S is greater than the number $A(m)$ of alternating substrings of an alternating string of length m. We will show that this assumption leads to a contradiction.

In all cases, at most one of the elements of C is present in an alternating substring. If W is empty, then an alternating substring of S is formed by any element of C concatenated with an anti-alternating substring of E, or by no element of C nor e_1, but an alternating substring of $e_2e_3\cdots e_{m_E}$. By the inductive hypothesis and Lemma 3.1, the number of alternating substrings in every string of length i for $i < m$ is at most ϕ^i. Thus,

$$A(S) \leq c\phi^{m-c} + \phi^{m-c-1}.$$

We have assumed that $A(S) > A(m)$, implying that

$$\begin{aligned} \phi^{m-c} + \phi^{m-c-1} &> \phi^m, \text{ i.e.,} \\ c + \frac{1}{\phi} &> \phi^c, \end{aligned}$$

which has no solution, contradicting our assumption. An analogous argument applies if instead E is empty.

If neither W nor E is empty, then an alternating substring of S is formed by concatenating an alternating substring of W, an element of C, and an anti-alternating substring of E,

or by concatenating an alternating substring of W with an alternating substring of E. In the latter case, the number of strings in which w_{m_W} and e_1 both appear must be subtracted, since $w_{m_W} = e_1 = \mathsf{a}$. Again using the inductive hypothesis and Lemma 3.1,

$$A(S) \leq c\phi^{m-c} + \phi^{m-c} - \phi^{m-c-2}.$$

Invoking the assumption that $A(S) > A(m)$ yields

$$c + 1 - \frac{1}{\phi^2} > \phi^c,$$

which again has no solution, providing the desired contradiction. □

We now use Lemmas 3.1 and 3.2 to bound the number of candidate full sets for a given backbone. Assume that no two terminals have the same x or y value; if necessary, this property can be ensured by perturbation (see Edelsbrunner [47]). Label the terminals $t_1, t_2, \ldots, t_n$ such that $x_{t_i} < x_{t_{i+1}}$ for all $1 \leq i < n$. We now show that if t_i and t_j define a backbone β, then the number of full sets inducing a type I topology with β as a backbone is at most $O(\phi^{j-i})$.

Lemma 3.3 *The number of full sets inducing a type I topology with a backbone defined by terminals t_i and t_j is at most $O(\phi^{j-i})$.*

Proof: Assume without loss of generality that the backbone β is horizontal, its right endpoint is t_j, its left endpoint is (x_{t_i}, y_{t_j}), and $y_{t_i} < y_{t_j}$. Let $y_\beta = y_{t_j}$.

A type I topology with β as a backbone can contain only terminals t_k such that $i \leq k \leq j$. Build a string S of length $j - i - 1$ based on the positions of the terminals relative to β; specifically, $s_{k-i-1} = \mathsf{a}$ if $y_{t_k} > y_\beta$, and $s_{k-i-1} = \mathsf{b}$ if $y_{t_k} < y_\beta$. Since $y_{t_i} < y_\beta$, terminal t_i corresponds to a b, and there is a one-to-one correspondence between alternating substrings of S and sets of terminals that form a valid type I topology with β as their backbone.

From Lemmas 3.1 and 3.2, the maximum number of alternating substrings of a string of length m is ϕ^m, so the maximum number of candidate full sets with β as a backbone is ϕ^{j-i-1}. This value must be doubled to account for the case where β is vertical rather

than horizontal, and doubled again for the case when β is adjacent to t_i rather than t_j, giving us $4\phi^{j-i-1} = O(\phi^{j-i})$, as desired. □

We now use Lemma 3.3 to show that the total number $f(n)$ of candidate full sets on n terminals is at most $O(n\phi^n)$.

Theorem 3.1 *The number $f(n)$ of candidate full sets on n terminals is at most $O(n\phi^n)$.*

Proof: A candidate full set must satisfy Hwang's theorem, so the quantity $f(n)$ is bounded by the total number of candidate full sets with type I or type II topology for every possible backbone. Lemma 3.3 gives a bound on the number of candidate full sets with type I topology for a given backbone. Given a backbone defined by terminals t_i and t_j, a type II topology is formed by joining a terminal to the left of t_i or to the right of t_j to the full tree. In the worst case, for every type I topology, any of these terminals forms a valid type II topology in this manner. Thus, from Lemma 3.3, the maximum number of candidate full sets with type II topology for a backbone defined by t_i and $t_j = t_{i+m}$ is $O((i+n-j)\phi^{j-i})$. Summing over all choices of t_i and t_j yields

$$f(n) \leq \sum_{i=1}^{n} \sum_{m=0}^{n-i+1} (n-m+1)\phi^m = O(n\phi^n),$$

as desired. □

3.6 Screened Full-set Dynamic Programming

Given the results above, it remains only to show that for each subset S of terminals, the set of candidate full sets that are subsets of S, i.e. the set $F(S) = \{S' \in F(T) : S' \subseteq S\}$, can be efficiently computed for each S. We now show that this computation can be accomplished in $O(|F(S)| + |S|)$ time.

The set $F(S)$ consists of S itself if $S \in F(T)$, as well as $F(S')$ for each $S' = S - \{s_i\}$ such that $s_i \in S$. This observation dovetails with the dynamic programming algorithm: as the

algorithm enumerates the sets S of cardinality m, it retains $F(S)$ for each. Each new $F(S)$ is then computed as follows:

$$F(S) = (\{S\} \cap F(T)) \cup \bigcup_{S'=S-\{s_i\}} F(S').$$

The algorithm stores each $F(S)$; since the subsets are enumerated in order of increasing cardinality, each $F(S')$ has been stored from the previous iteration, and this computation is performed in $O(|F(S)| + |S|)$ time.

We now describe the algorithm, which we call *Screened Full-set Dynamic Programming (SFDP)*. The subsets of the input set T of terminals are enumerated in order of increasing cardinality, starting with subsets of cardinality three (subsets of cardinality two cannot be decomposed, though they are considered in the decomposition of larger subsets). For each subset S, compute $F(S)$ as described in the previous paragraph, and examine each subset A in $F(S)$. For every such A, consider each set B such that $A \bowtie B = S$. The length of an optimal RST of S is the minimum of the length of a full tree on S (if S is in $F(T)$) or the minimum combined length of optimal RSTs of each pair of subsets A and B such that $A \in F(S)$ and $A \bowtie B = S$. Since the subsets are enumerated in order of increasing cardinality, each required B has already been computed and stored. Figure 3.2 describes the SFDP algorithm in detail.

As described above, $F(S)$ is computed for every set S in $O(|F(S)| + |S|)$ time. Thus, the time complexity of the SFDP algorithm is at most

$$\sum_{m=3}^{n} \binom{n}{m} m(f(m) + m).$$

From Theorem 3.1, $f(m) \leq m\phi^m$, so the time complexity of the SFDP algorithm is at most

$$\begin{aligned} \sum_{m=3}^{n} \binom{n}{m} m(m\phi^m + m) &\leq n^2(1+\phi)^n + n^2 2^n \\ &= O(n^2(1+\phi)^n) \\ &\approx O(n^2 2.62^n). \end{aligned}$$

```
(1)  For m = 3 to |T|
(2)      For all S ⊆ T such that |S| = m
(3)          If (S ∈ F(T)) then ℓ[S] = ‖H(S)‖
             Else ℓ[S] = ∞
(4)          Compute F(S)
(5)          For all A, B such that A ∈ F(S) and A ⋈ B = S
(6)              ℓ[S] = min{ℓ[S], ℓ[A] + ℓ[B]}
```

Figure 3.2: The SFDP algorithm. T is the set of input terminals and $F(T)$ is the set of candidate full sets. $H(S)$ is an optimal full tree of the set S of terminals.

The time required to compute the set $F(T)$ of candidate full sets is $O(h(n))$, where $h(n)$ is the number of subsets that satisfy Hwang's theorem [131]; by Theorem 3.1, this time complexity does not exceed $O(n\phi^n)$ and is dominated by the decomposition term.

Like the FDP algorithm, the space complexity of the SFDP algorithm is $O(2^n)$. Again, one may wish to store the optimal decomposition of each subset along with its length, increasing the space requirements to $O(n2^n)$ but eliminating the need for two passes.

3.7 Time Complexity

As mentioned previously, Smith's $n^{O(\sqrt{n})}$ algorithm [139] is asymptotically faster than either of our algorithms, or indeed than any $\Omega(c^n)$ algorithm with $c > 1$. However, for instances small enough to solve in practice, our algorithms are provably faster than Smith's.

The time complexity of Smith's algorithm is given by the following recurrence:

$$T_W(n) = n^{\sqrt{n}} \cdot 4^{\sqrt{n}} \cdot 2 \cdot T_W(n/2).$$

This recurrence solves to $n^{O(\sqrt{n})}$ for asymptotically large n, but a more accurate value for smaller n is derived by actually expanding this recurrence. Expanding once we get

$$T_W(n) = (4n)^{\sqrt{n}} \cdot 4 \cdot (2n)^{\sqrt{n/2}} \cdot T_W(n/4).$$

Similarly, we consider the more exact bounds for the FDP algorithm:

$$T_F(n) = n3^n + n2^n,$$

and for the SFDP algorithm:

$$T_S(n) = n^2 2.62^n + n^2 2^n + n^2 1.62^n.$$

Even if we only expand Smith's recurrence once (assume that $T_W(n/4) = 1$), the value of $T_W(n)$ is greater than $T_F(n)$ for n less than 61, and it is greater than $T_S(n)$ for n less than 77. Figure 3.3 depicts the behavior of these three functions with respect to n. Note that currently the fastest algorithms only solve 35-terminal instances in a day on a

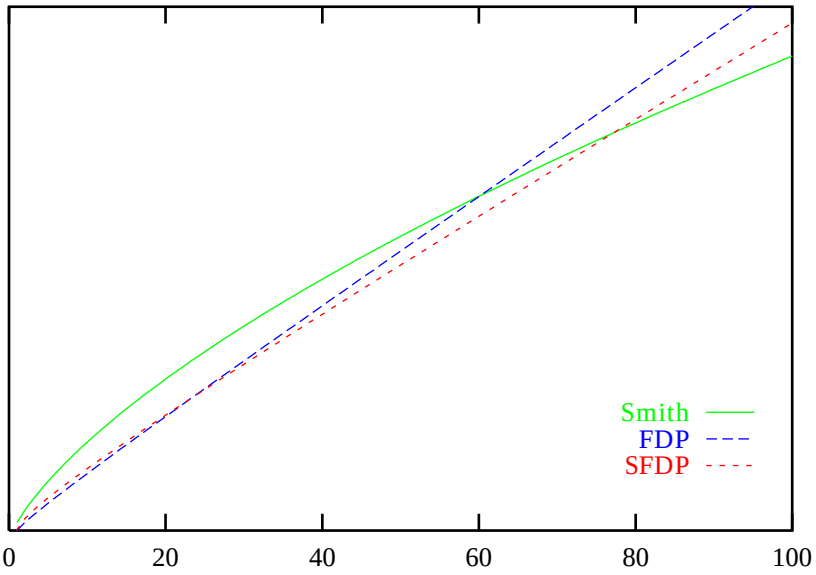


Figure 3.3: Running times of Smith's algorithm vs. the FDP and SFDP algorithms.

workstation [131], so computational technology will have to improve tremendously before Smith's algorithm is faster than ours on practically solvable instances.

Above we obtained a loose lower bound on the running time of Smith's algorithm by expanding the recurrence only once; if one were to expand the recurrence further, then one would discover that Smith's algorithm is slower than ours for even larger instances than described above. In fact, Smith states that one would want to use a different algorithm for instances with less than 300 terminals [139].

Hwang, Chang, and Lee [85] use essentially the same techniques as Smith to solve a number of other problems; readers interested in the technique itself are referred there as well as to Smith [139].

3.8 Empirical Results

We have implemented the FDP and SFDP algorithms in order to compare them empirically with Hakimi's algorithm [70], the Dreyfus-Wagner algorithm [43], and the algorithm of Thomborson, Alpern, and Carter [145].

Figure 3.4 plots the running time of each algorithm as a function of the number n of input terminals. Each data point results from 10 runs on each of 10 different sets of terminals generated uniformly at random from a 10000 by 10000 grid. As can be seen, the FDP algorithm is faster than the Hakimi, Dreyfus-Wagner, and Thomborson, Alpern, and Carter algorithms.

While the SFDP algorithm is a bit slower than the FDP algorithm for $n < 10$, after this point it is faster than all four of the other algorithms, and becomes more so as n grows, due to the asymptotic improvement in the exponential. Note that $O(n^2 2.62^n)$ is an extremely pessimistic bound. In practice, we apply a number of other tests for full set candidacy [131], and $f(n)$ is *much* smaller than $O(n\phi^n)$ in practice. In fact, we conjecture that the upper bound on $f(n)$ is at most $O(n^2)$ and is quite possibly $O(n \log n)$. Indeed, the slope of the running time curve for the SFDP algorithm suggests that its time complexity in practice is $O(p(n)2^n)$, where $p(n)$ is a polynomial function of n. This suggests that the number of full sets, at least in practice, is indeed polynomial, since the 2^n term is incurred by the

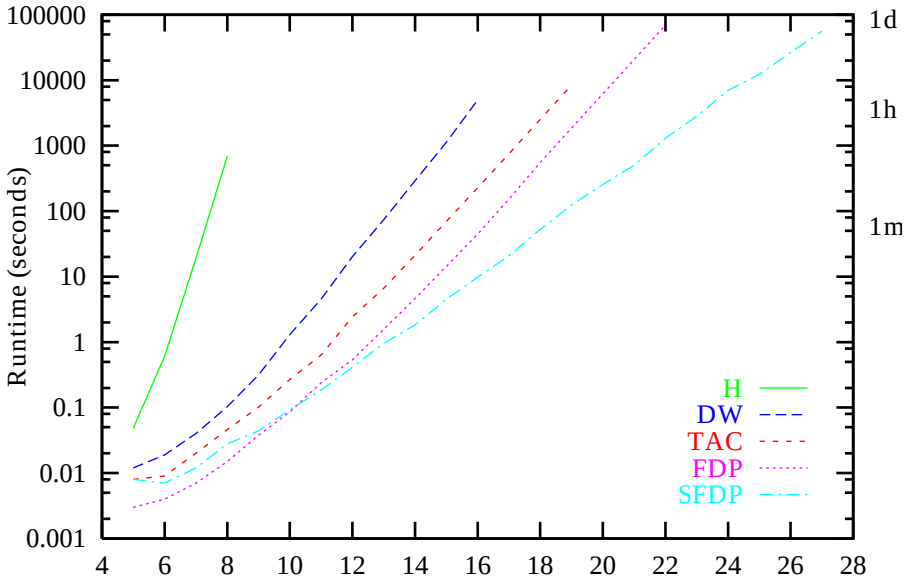


Figure 3.4: Running times of Hakimi (H), Dreyfus and Wagner (DW), Thomborson, Alpern, and Carter (TAC), FDP, and SFDP algorithms.

outer two loops of the algorithm alone. Figure 3.5 depicts, as a function of n, our upper bound of $n\phi^n$, the observed maximum, minimum, and average numbers of candidate full sets over $10,000$ randomly generated instances for each value of n, and the conjectured upper bound of n^2.

The Salowe and Warme algorithm [131] is still faster than the SFDP algorithm in practice. However, since they use a branch-and-bound search to examine the various full-set decompositions, the running time of the algorithm is unpredictable, and it is difficult to derive a bound on the time complexity of their algorithm that is better than $O(2^{f(n)})$, where $f(n)$ is the maximum number of candidate full sets on n terminals. The bound on the number of full sets in Theorem 3.1 gives an upper bound of $O(2^{n\phi^n})$. Based on the performance of the algorithm in practice, this bound is pessimistic, at least for randomly

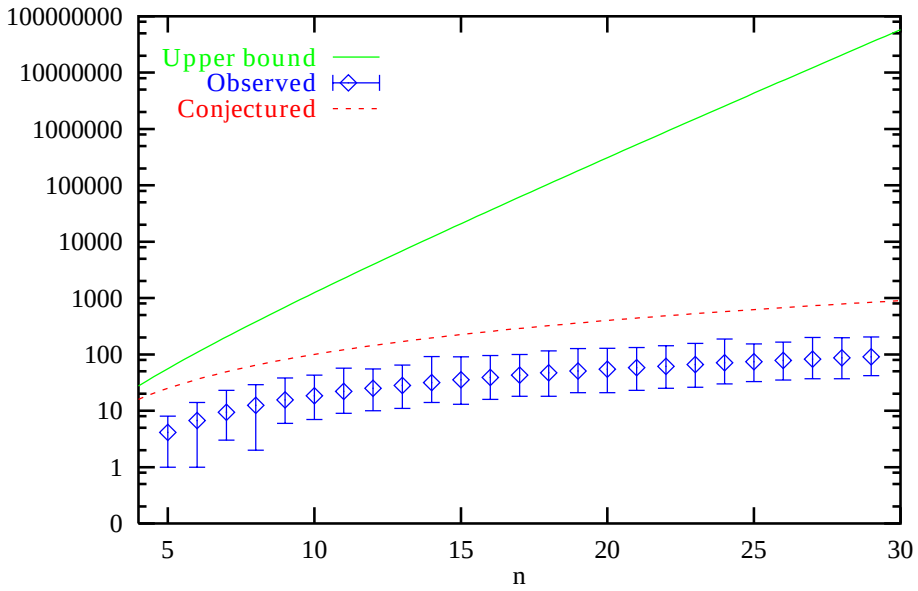


Figure 3.5: Number of candidate full sets as a function of n: the proven upper bound, empirical values for randomly generated data, and the conjectured upper bound of n^2.

generated sets of terminals. However, in the absence of good worst-case bounds, it is impossible to know whether there exist pathological instances for which the algorithm is much slower than it is for randomly generated data.

3.9 Conclusions and Future Work

We have presented two new dynamic programming algorithms for computing optimal rectilinear Steiner trees. The first is called Full-set Dynamic Programming (FDP), and runs in $O(n3^n)$ time. The FDP algorithm is very simple and easily implemented. It is particularly well-suited to applications in which the Dreyfus-Wagner algorithm was previously used, such as in the basis of recursive decomposition algorithms like those of Komlós and

Shing [98] and Deneen, Shute, and Thomborson [42]. It is at least as easily implemented as the Dreyfus-Wagner algorithm and is faster and uses less space.

Our second algorithm is called Screened Full-set Dynamic Programming (SFDP), and runs in at most $O(n^2(1+\phi)^n)$ time, where $\phi = (1+\sqrt{5})/2 \approx 1.62$. The analysis of the SFDP algorithm includes an upper bound of $O(n\phi^n)$ on the maximum number of full sets on n terminals. We have proven that both algorithms, while asymptotically slower than that the algorithm of Smith [139], are provably faster than Smith's algorithm for instances that can be solved in practice. Thus, for instances that can be solved in practice, the SFDP algorithm has faster proven time complexity than that of any previous algorithm for computing optimal RSTs.

Finally, we have demonstrated empirically that our algorithms are faster in practice than three popular previous algorithms for computing optimal RSTs: Hakimi's spanning tree enumeration algorithm [70], the dynamic programming algorithm of Dreyfus and Wagner [43], and the improvement to the Dreyfus-Wagner algorithm devised by Thomborson, Alpern, and Carter [145].

We note that the improvement of the SFDP algorithm over the FDP algorithm comes at the expense of a significant increase in implementation complexity. In our implementation, in addition to Hwang's theorem, a number of other tests are applied to eliminate subsets from full-set candidacy [131]. We are investigating how each test for full-set candidacy affects the resulting number of candidate full sets. In particular, applying only Hwang's theorem still guarantees the worst-case runtime of the SFDP algorithm, but would reduce the code considerably. We are investigating how this change affects the runtime of the SFDP algorithm in practice.

Our work continues toward further improving the running time of these algorithms. The main source of inefficiency in the SFDP algorithm is examining every subset of the set of terminals. In practice, it appears that a very small portion of these subsets admits *any* valid decomposition into candidate full sets—in our tests, the ratio of the number

of subsets with valid decompositions to the total number of subsets is roughly 0.04 for 20-terminal instances, and this ratio decreases as n grows. We are currently working on a "lazy" implementation that works in a top-down fashion, computing the optimal RSTs of subsets of terminals only on demand. In this way, we can avoid examining many subsets that do not admit any valid decomposition. We believe that with such a modification, the time complexity of the algorithm can be improved to $O(p(n)c^n)$ for $c < 2$ and that it would become competitive with the Salowe-Warme algorithm in practice. Note that this modification requires a different way to handle enumerating candidate full sets, as the current technique is dependent on enumerating every subset of the set of terminals.

Perfection is the child of Time.

— Bishop Joseph Hall, *Works*

4

Obstacle-Avoiding Rectilinear Steiner Trees

In VLSI physical design automation, a fundamental task is routing a net, i.e. interconecting a set of terminals. Often this routing is performed in the presence of obstacles that the wires of the net must not intersect, such as logic cells and wires in previously routed nets.

A special case that has received significant attention is the case in which only two terminals need to be connected. This problem is equivalent to finding a rectilinear shortest path in the presence of obstacles. A more general formulation, in which more than two terminals must be interconnected, forms a generalization of the rectilinear Steiner tree problem. The *rectilinear Steiner tree (RST)* problem is described in Section 2.2.2. The *obstacle-avoiding rectilinear Steiner tree (OARST)* problem is identical to the RST problem except for the presence of rectilinear obstacles that the segments in the Steiner tree must not intersect. The RST problem is NP-complete [63], and therefore the OARST problem is as well. However, the two-terminal problem is efficiently solvable.

The remainder of this chapter consists of four sections. Section 4.2 surveys some of the literature on the special case of the OARST problem in which there are only two terminals. Section 4.3 presents results from the literature as well as new results on the general OARST problem. Section 4.4 discusses several special cases of the OARST problem, such as when

Earlier versions of portions of this chapter appear in Ganley and Cohoon [30, 57].

all terminals lie on obstacle perimeters or when all terminals lie on the perimeter of the routing region. Finally, Section 4.5 gives conclusions and some open problems.

4.1 Terminology

An instance of the OARST problem consists of a set T of terminals and a set S of obstacle perimeter segments. We denote an instance as a pair (T, S). We let $n = |T|$ and $s = |S|$, and denote the instance size $c = n + s$.

4.2 Two-Terminal Interconnections

The problem of two-terminal rectilinear interconnection in the presence of obstacles has been well studied. We refer to this special case of the OARST problem as the *obstacle-avoiding shortest path (OASP)* problem. The literature on the OASP and similar problems is quite vast; since other surveys have been written [74, 104, 141] and since our focus is on Steiner-tree routing of multi-terminal nets, we only overview some of the principal results for the two-terminal case.

4.2.1 Grid-based algorithms

The earliest techniques for solving the OASP problem are so-called *grid-based* algorithms. Grid-based routing techniques find their genesis in the *maze routing* algorithms of Lee [103] and Moore [120]. In a grid-based algorithm, the routing surface is typically divided into a grid in which each square is the size of the smallest feature that can be fabricated in the given technology (this distance is denoted λ). This grid divides the routing area into a number of *grid cells*. The two terminals, a and b, are each associated with a grid cell. An integer value v_{xy} is associated with each grid cell (x, y). Initially $v_{xy} = \infty$ for all x and y. Then v_a is assigned the value 0, and adjacent grid cells are visited in breadth-first search order. Each time a grid cell (x, y) is visited, for each of its neighbors (x', y'), $v_{x'y'}$ is set to

Figure 4.1: The result of running a maze routing algorithm.

---

with value $d-1$, $d-2$, etc., until the point a with value 0 is reached. One such path
n in bold in Figure 4.1.

major drawback of grid-based techniques is that they require time and space corre-

for rectilinear interconnection in the presence of obstacles began with the independent works of Hightower [73] and Mikami and Tabuchi [116]. Their algorithms were innovative in that they were the first line-based routing algorithms, but they suffer several disadvantages. Primary among them is that they do not necessarily find a shortest path between the two terminals, and furthermore they sometimes does not find any solution although one exists.

Soon after these works, other researchers devised algorithms that solve these problems. Many of these approaches, rather than describing an algorithm per se, describe the construction of a graph guaranteed to contain shortest paths between all pairs of terminals, to which shortest-path algorithms are then applied. One such graph is the *escape graph* of Cohoon and Richards [31].

4.2.2.1 The escape graph

Figure 4.2(a) depicts a set of obstacles and terminals. The escape graph is constructed as follows. Draw a "beltway" around each obstacle at a distance of λ from the obstacle, and similarly inscribe a beltway around the interior of the routing region, as shown in Figure 4.2(b). Then extend each segment maximally, i.e. until it hits another beltway segment, as shown in Figure 4.2(c). (Some researchers assume that $\lambda = 0$, in which case step (b) is skipped, and step (c) amounts to simply extending each obstacle perimeter segment maximally.) Finally, extend segments from the terminals in all unobstructed directions, again maximally, as shown in Figure 4.2(d). The segments described by this construction are called *escape segments*. From the escape segments, a graph called the *escape graph* is computed in a straightforward manner: the vertices correspond to the intersections of escape segments, and there is an edge between every pair of vertices that are adjacent along an escape segment. The weight of an edge is the rectilinear distance between its endpoints. We let $G_e(T, S)$ denote the escape graph for an instance (T, S), and let m denote the number of vertices in $G_e(T, S)$. Note that $m = O(c^2)$ in the worst case and also that $G_e(T, S)$ is

(d)

Figure 4.2: Constructing the escape graph.

4.2.2.2 Other algorithms

Wu, Widmayer, Schlag, and Wong [155] describe a graph they call a *track graph*, which is identical to the escape graph defined in Section 4.2.2.1 except that the segments adjacent to terminals extend only until they hit *some* escape segment (in the escape graph they are extended maximally). Thus, the track graph contains $O(n + s^2)$ vertices. Note that the track graph itself does not necessarily contain a shortest path between every pair of terminals. However, they describe an algorithm that, using the track graph, finds a set of single-source shortest paths in $O(n \log n + s^2 \log s)$ time. Depending on the values of n and s, this may or may not improve on the $O(m)$ time required by the technique of applying Kanchanasut's algorithm to the escape graph.

Several researchers [6, 31, 112] have devised techniques to find a shortest path between two vertices in a rectilinear graph such as the escape graph in $O(c \log c)$ time. (Note that this can be *sublinear* in the size of the escape graph!) These techniques would seem to improve upon both the straightforward algorithm and the algorithm of Wu, Widmayer, Schlag, and Wong, but one must keep in mind that it takes at least $O(m)$ time to *generate* the escape graph, so these algorithms require at least $O(m)$ total time.

Mitchell [119] describes a more direct plane-sweep algorithm that does not explicitly use any graph structures. The algorithm requires $O(e \log s)$ time, where e is the number of "events" encountered during the plane-sweep algorithm. He proves that $e \leq O(s \log s)$, making the overall time complexity of the algorithm $O(s \log^2 s)$, which improves upon the times required by the algorithms described above. Furthermore, Mitchell conjectures that in fact $e = O(s)$, in which case the algorithm would have the optimal running time of $O(s \log s)$.

An elegant approach to the OASP problem is due to Widmayer [148], who describes the construction of a graph similar to the escape graph that contains $O(s \log s)$ vertices while still containing a shortest path between every pair of terminals. Widmayer calls this graph the *shortest-paths preserving graph*, or *spp graph*.

The spp graph is defined recursively in the following way. Let P denote the set of terminals and obstacle corner points. Choose a vertical line located at $x = x_d$ that divides P into two roughly equal-sized subsets P_1 and P_2. Let $G_1 = (V_1, E_1)$ be the spp graph for the set of points P_1, and let $G_2 = (V_2, E_2)$ be the spp graph for the set of points P_2. Initialize V' and E' to $\emptyset$. For each $p \in P_1 \cup P_2$, let v_p be a new vertex located at (x_d, y_p). For each v_p such that the straight line from p to v_p does not intersect any obstacle, add v_p to V' and add the edge (p, v_p) to E'. Also, add to E' the edge (v_{p_1}, v_{p_2}) for each v_{p_1} and v_{p_2} such that no obstacle and no other v_{p_i} lies between v_{p_1} and v_{p_2}. Now, the spp graph for P is $G = (V, E)$, where $V = V_1 \cup V_2 \cup V'$ and $E = E_1 \cup E_2 \cup E'$.

It can be shown that the spp graph contains a shortest path between every pair of terminals, and that it contains $O(s \log s)$ vertices and edges. Furthermore, the spp graph can be generated in $O(s \log s)$ time. The reader is referred to Widmayer [148] for details.

Widmayer also describes how to use the spp graph to compute a shortest path between two terminals in $O(s \log s \log \log s)$ time. However, we can apply Kanchanasut's algorithm [92] to the spp graph to compute a set of single-source shortest paths in $O(s \log s)$ total time, which is optimal [37].

Recent work by Lee, Yang, and Wong [105] gives an optimal-time plane-sweep algorithm that also solves the OASP problem directly in $O(s \log s)$ time.

4.2.3 Multiple-net routing

The problem of routing multiple nets has received substantially less attention than the single-net problem. The technique often used in practice is *sequential* routing, where the nets are ordered in some fashion, and then they are each routed in sequence, with each net becoming a set of obstacles for subsequently routed nets. If the instance is not entirely routed using this technique, then often *rip-up and reroute* techniques are applied, where one or more of the previously routed nets are removed and rerouted differently to make room for other nets [39, 40, 136]. Clearly these techniques are heuristic in nature, but

the more elegant solution—simultaneously routing all the nets—is NP-complete even in a planar graph if all nets contain only two terminals [126, 128].

Jájá and Wu [88] describe a technique for doing just that: simultaneously routing a number r of two-terminal nets. Their approach is to modify the basic escape graph by replacing each escape segment with $O(r)$ parallel segments at distance λ from each other. A r-net escape graph G_e^r is constructed from these segments in the same manner as for the single-net escape graph; it contains $O(r^2s^2)$ vertices and edges. The main result proven by Jájá and Wu is that the G_e^r correctly generalizes the single-net escape graph to the r-net case. Let R denote the set of nets; each net R_i in R consists of a pair of terminals. Thus, a problem instance is denoted (R, S), where S is the set of obstacle perimeter segments.

Theorem 4.1 (Jájá and Wu [88]) *A r-net routing instance (R, S) has a solution if and only if there are r vertex-disjoint paths in $G_e^r(R, S)$.*

They use algorithms by Robertson and Seymour [130] to find these vertex-disjoint paths in polynomial time for any fixed r. This approach is of largely theoretical interest, as the running time is exponential in r. However, Theorem 4.1 might instead be used in a heuristic fashion for routing multiple nets; to our knowledge, this has not yet been tried.

4.3 Multi-Terminal Interconnections

The general OARST problem has received substantially less attention than the special case in which there are only two terminals. Several authors [108, 155] have pointed out that if one can solve the OASP problem, then one can construct a *minimum spanning tree (MST)* of a multi-terminal instance. The MST has length not exceeding twice the length of an optimal OARST; thus, an obstacle-avoiding MST algorithm is a 2-approximation algorithm for the OARST problem [108].

For the standard RST problem, Hanan's theorem (Theorem 2.1 on page 14) states that an optimal RST always exists that is a subgraph of the grid graph formed by passing a

horizontal and vertical line through each terminal. Hanan's theorem enables solution of the RST problem by a reduction to the *graph Steiner tree (GST)* problem. We generalize this result to the OARST problem by showing that the escape graph is guaranteed to contain an *optimal* OARST (note that if there are no obstacles, then the Hanan grid graph and the escape graph are identical).

Theorem 4.2 *If an OARST exists for an instance* (T, S)*, then there exists an optimal OARST that is a subgraph of the escape graph* $G_e(T, S)$*.*

Proof: Suppose there exists a problem instance (T, S) with $|T| > 2$, such that all optimal Steiner trees for (T, S) contain at least one segment that is not an escape segment. We show that this supposition leads to a contradiction.

Let τ be an optimal Steiner tree for (T, S) that contains a minimum number of non-escape segments among optimal Steiner trees for (T, S). Let segment ℓ be a segment in τ that is not an escape segment. Without loss of generality, assume that ℓ is horizontal.

Obviously, ℓ has two endpoints a and b beyond which no further colinear segment is incident. There may be segments incident and orthogonal to ℓ. In fact, there must be orthogonal segments incident to a and b. If either a or b did not have an orthogonal segment incident to it, then it would either be a terminal, contradicting the assumption that ℓ is not an escape segment, or else a portion of s could be removed, contradicting the assumption that τ is optimal.

Let u be the number of orthogonal segments incident to ℓ from above, and let d be the number of orthogonal segments incident from below. Colinear segments that are incident to ℓ both from above and below are considered two distinct segments separated by ℓ.

If u is equal to d, then slide ℓ up until it is colinear with some escape segment. There is room to slide since ℓ is not an escape segment. An escape segment above ℓ must exist, since the routing region perimeter is itself inscribed by escape segments. Since the length of the segments above ℓ decreases by exactly the amount that the length of the segments below ℓ increases, the tree resulting from this maneuver has the same length as τ; hence,

it is optimal. In addition, every vertical segment incident to ℓ that was an escape segment remains an escape segment. In fact, all segments that were escape segments before the slide remain so. Thus, the tree resulting from this sliding maneuver contradicts our assumption that τ contains a minimum number of non-escape segments.

If instead, u is greater than (less than) d, then we may slide ℓ up (down), decreasing the length of the tree and contradicting its optimality. We again know there is room to slide, since ℓ is not an escape segment.

This completes the case analysis. We have shown that every solvable instance of the OARST problem has an optimal solution composed only of escape segments, and thus an optimal solution to the GST problem in $G_e(T, S)$ is an optimal solution to the OARST instance (T, S). □

Theorem 4.2 is the first to allow computation of optimal OARSTs in time corresponding to the instance size rather than the size of the routing area. In addition, since the escape graph is guaranteed to contain an optimal OARST, applying approximation algorithms for the graph Steiner tree problem produces equivalent approximations for the OARST problem.

4.3.1 Escape graph reduction

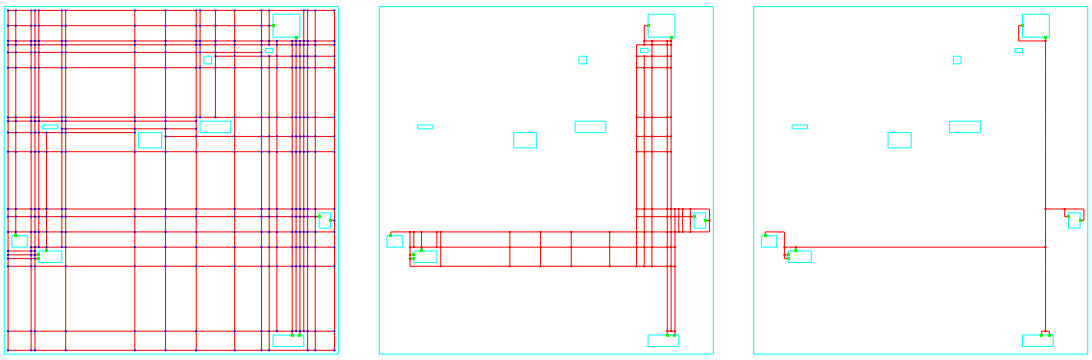
Often, many of the vertices in the escape graph can be deleted, along with their adjacent edges, while retaining the guarantee that an optimal solution exists that is constrained to the escape graph. We now describe a few straightforward tests whose application typically eliminates many vertices from the escape graph. We call the resulting graph the *reduced escape graph.*

The first test is the *dimension reduction test* of Yang and Wing [158] for the standard RST problem. Yang and Wing prove that if a vertex v is a corner vertex, i.e. it is incident to exactly two orthogonal edges e_1 and e_2, and v is not a terminal, and edges exist that

holds for escape graphs as well: every path that includes v
length that instead passes through the sides of the rectangle

est can be implemented to run in $O(m)$ time. Start by storing
ueue. Then repeat the following process until the queue is
m the front of the queue, and if it can be deleted according
delete v and its adjacent edges from the graph, and add its
ueue.

v of degree 2 that remains after the dimension reduction test



(a) (b) (c)

Figure 4.3: (a) The escape graph, (b) the reduced escape graph, and (c) an optimal OARST for an example instance.

problem, the escape graph model enables computation of optimal Steiner trees for three- or four-terminal nets as efficiently as a typical heuristic solution.

A well-known folk theorem of VLSI routing is that most nets contain four or fewer terminals. In an effort to verify this claim, we examine the SIGDA standard-cell benchmark suite [122]. Figure 4.4 illustrates the distribution of terminals per net in the SIGDA benchmarks. As can be seen in the figure, in these benchmarks three- and four-terminal nets comprise the vast majority of the nets with more than two terminals.

For a three-terminal net, an optimal Steiner tree has one of two topologies. It is either a simple path between the terminals, or else all three terminals are connected to a single Steiner point. Thus, an optimal OARST for a three-terminal net is computed as follows. The length of each of the three possible simple paths is checked, as well as the length of every tree formed by connecting the three terminals to each candidate Steiner point, and the tree with minimum length is optimal. The latter topology in which the tree contains a Steiner point dominates the computation time. This topology is examined in $O(m)$ time, where m is the number of candidate Steiner points, if all-pairs shortest paths information is

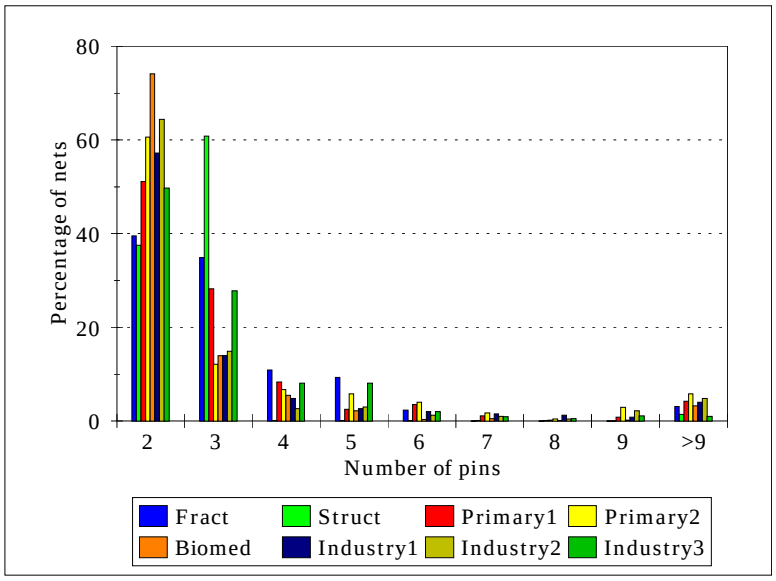


Figure 4.4: Distribution of terminals per net in the SIGDA benchmarks.

available. All-pairs shortest paths can be computed in $O(m^2)$ time by applying the $O(m)$ single-source shortest paths algorithm of Kanchanasut [92] to each vertex in the escape graph.

A similar observation can be made for four-terminal nets. For four terminals, the following topologies are possible:

- ◇ A simple path through the four terminals.
- ◇ A *star* in which three terminals are directly connected to the fourth.
- ◇ A *cross* in which all four terminals are directly connected to a single Steiner point.
- ◇ A T in which three terminals are directly connected to a single Steiner point, and the fourth terminal is connected to one of those three terminals.
- ◇ An H in which two terminals are directly connected to each of two Steiner points, which are directly connected to each other.

These topologies are illustrated in Figure 4.5. There are twelve possible mappings of the

Path Star Cross

Figure 4.5: The possible topolo

terminals onto the simple path topology, four ma
cross, twelve mappings for the T, and six mapping
instances—a sufficiently small number to examin
a four-terminal net is efficiently computed by en
star topologies are examined, the cross and T
every candidate Steiner point, and the H topolog
of candidate Steiner points. The shortest tree se
computation incurs a time complexity of $O(m^2)$,

These *explicit enumeration* algorithms are si
ation algorithm [70]; however, identifying and ex
has significant dividends. A straightforward im
enumeration algorithm that uses an MST algorit
enumeration algorithms. For example, for random

$n = 3$		$n = 4$	
EE	STE	EE	STE
0.04	1.45	0.08	9.06

Table 4.1: Average running times (in seconds) of explicit enumeration (EE) and spanning tree enumeration (STE) algorithms.

4.3.2.1 More than four terminals

It is possible to perform the case analyses and construct similar explicit enumeration algorithms for exact solution of problem instances with more than four terminals. However, the number of possible topologies increases superexponentially, and examining them all rapidly becomes too expensive.

As noted previously, a full topology is one in which every terminal is a leaf. Let $f(n,k)$ denote the number of full topologies with n terminals and k Steiner points. Since no vertex in the escape graph has degree exceeding 4, a Steiner point has degree 3 or 4. A full topology with n terminals and k Steiner points is formed by any pair of terminals connected to a vertex in a full topology with $n-1$ terminals and $k-1$ Steiner points, or by any three terminals connected to a vertex in a full topology with $n-2$ terminals and $k-1$ Steiner points. These facts lead to the following recurrence for $f(n,k)$:

$$\begin{aligned} f(n,1) &= 1 \\ f(n,k) &= \binom{n}{2} f(n-1,k-1) + \binom{n}{3} f(n-2,k-1) \end{aligned}$$

A full topology has anywhere from 1 to $n-2$ Steiner points, so the total number $f(n)$ of full topologies on n terminals is $\sum_{i=1}^{n-2} f(n,i)$.

A general topology on n terminals has one of the following forms:

- A full topology on n terminals,

- ◇ A topology on $n-1$ terminals, with the n^{th} terminal connected by an edge to one of the other $n-1$ terminals, or
- ◇ A full topology on i terminals, $3 \leq i \leq n-1$, connected at any of its terminals to a topology on the remaining $n-i+1$ terminals.

Thus, the total number $F(n)$ of topologies on n terminals is given by the following recurrence:

$$\begin{aligned} F(2) &= 1 \\ F(n) &= f(n) + nF(n-1) + \sum_{i=3}^{n-1} f(i) \binom{n}{i} iF(n-i+1) \end{aligned}$$

The values of $f(n)$ and $F(n)$ for $3 \leq n \leq 8$ are given in Table 4.2. As the table suggests,

n	3	4	5	6	7	8
$f(n)$	1	7	81	1356	31312	952673
$F(n)$	4	35	516	10662	285398	9496937

Table 4.2: The number $f(n)$ of full topologies and the total number $F(n)$ of topologies on n terminals.

the explicit enumeration approach might be practically applicable for five terminals, but almost certainly not for six. The time complexity of explicit enumeration can be improved by examining only full topologies and allowing degenerate full topologies (i.e., allowing more than one Steiner point to map to the same vertex and allowing Steiner points to map to terminal vertices). However, this still requires the examination of $f(n)$ different topologies, which rapidly becomes prohibitively large.

4.3.3 Graph-based heuristics

Since explicit enumeration is probably impractical for nets with more than four terminals, heuristics for the graph Steiner tree problem can instead be used to quickly find good

solutions for such nets. Given the exact three- and four-terminal algorithms in Section 4.3.2, a natural approach is *Steinerization*. In a Steinerization heuristic, portions of an MST that contain a few adjacent terminals are replaced with an optimal Steiner subtree for those terminals. Typically, such heuristics examine subsets of a fixed size, i.e. subsets of size N for some small N. In light of the results in Section 4.3.2, we examine heuristics with $N = 3$ and $N = 4$.

The first heuristic, *greedy Steinerization*, starts with an MST of the terminals. It then repeatedly examines vertex subsets of size N that are adjacent in the MST, Steinerizing the one that improves the MST the most. The Steiner points introduced by the Steinerization are candidates for further Steinerization in later iterations. For $N = 3$, greedy Steinerization is an oft-repeated idea whose genesis is unclear. For the standard RST problem, Richards (see Hwang, Richards, and Winter [84]) first investigated 3-Steinerization in this greedy form, and more complex variants appear in Chao and Hsu [20], Lee, Bose, and Hwang [106] and Smith, Lee, and Liebman [138]; these and others are summarized in Hwang, Richards, and Winter [84]. For the OARST problem, greedy 3-Steinerization (henceforth called *G3S*) has time complexity $O(n^2m)$, since $O(nm)$ time is required to find each locally optimal 3-Steinerization and at most $O(n)$ of these operations are performed. For $N = 4$, greedy Steinerization is similar to the algorithm of Beasley [10], though Beasley's algorithm computes a new MST at each iteration rather than locally modifying the current MST. For the OARST problem, greedy 4-Steinerization (henceforth, *G4S*) has time complexity $O(n^3m^2)$, since $O(n^2m^2)$ time is required to find each locally optimal 4-Steinerization and at most $O(n)$ of these operations are performed.

The running time of Steinerization heuristics can be improved using a batching technique similar to the heuristic of Hasan, Vijayan, and Wong [72] for the standard RST problem. In their *neighborhood Steinerization* heuristic, each vertex v is assigned a weight that is the amount of improvement that is gained by Steinerizing v and its neighbors. Since every vertex in a rectilinear MST has at most 8 neighbors, each Steinerization is performed in

constant time. Since large neighborhoods cannot be efficiently Steinerized in the OARST problem, the heuristic instead sets the weight of a vertex v to the maximum improvement resulting from 3-Steinerizing v and any two of its neighbors.

The heuristic then finds a *maximum-weight independent set (MWIS)* of the tree. An MWIS of a tree is computed in $O(n)$ time by the following dynamic programming algorithm. Choose an arbitrary vertex r to be the root of the tree. Associate with each vertex v in the tree two independent sets in the subtree rooted at v. The independent set $M^+(v)$ is an MWIS that includes v, and $M^-(v)$ is an MWIS that does not include v. Computing $M^+(v)$ and $M^-(v)$ is trivial when v is a leaf. When v is an internal vertex, $M^+(v)$ is the union of $M^-(u)$ for all children u of v. Similarly, $M^-(v)$ is the union of $M^+(u)$ for all children u of v. The sets $M^+(v)$ and $M^-(v)$ are computed for each vertex v in bottom-up fashion, starting with the leaves. When the algorithm terminates, whichever of $M^+(r)$ and $M^-(r)$ has maximum weight is the MWIS of the tree.

The best 3-neighborhood of each vertex in the MWIS is then Steinerized, and the process is repeated for the new tree resulting from replacing each neighborhood with its Steiner subtree. The time complexity of this algorithm, which we call *batched* 3*-Steinerization (B3S)* is $O(bnm)$, where b is the number of iterations required, which is a function of n. In the worst case, b is $O(n)$, so the worst-case time complexity is the same as for G3S; however, this bound is quite pessimistic. Table 4.3 shows the average value of b for various values

n	3	5	7	9	11	13	15
b	1.0	1.5	1.8	2.0	2.1	2.2	2.3

Table 4.3: Average iteration count for B3S.

of n, for randomly generated instances containing 10 rectangular obstacles. Empirically it appears that b is $O(\log n)$, giving B3S a time complexity of $O(mn \log n)$ in practice.

Another optimization can further reduce the running time of Steinerization heuristics. Before computing the Steiner subtree for each subset T' of the terminals, perform the reductions described in Section 4.3.1 on the escape graph, considering only members of T' to be terminals. Since the size N of the subsets considered is small, the number of vertices eliminated from the escape graph is typically substantial. Since the number N of terminals is a fixed constant, the reductions have time complexity $O(m)$ for each subset. For 3-Steinerization, this is equal to the time complexity of actually Steinerizing the subset, so performing the reductions is not productive. However, for 4-Steinerization, the cost of Steinerizing each subset is $O(m^2)$, so linear-time preprocessing is effective if it substantially reduces m. For $N = 4$, many of the vertices are typically eliminated, and experimentally this approach does indeed dramatically improve the running time of G4S—for the 20-terminal instances tested, this reduction optimization improves the average running time by a factor of almost 2.5.

Table 4.4 shows the result quality (percent improvement over the MST) and runtime for G3S and B3S, and for G4S with the reduction optimization described above, for randomly generated instances containing 10 rectangular obstacles and the indicated numbers of terminals. For the standard RST problem, the average improvement of optimal RSTs over the MST is roughly 12% (see Hwang, Richards, and Winter [84]). For the instances of the OARST problem tested here, this value is somewhat lower. An optimal Steiner tree in the escape graph can be computed using the algorithm of Dreyfus and Wagner [43], which has time complexity $O(m3^n)$. Table 4.5 gives the average improvement of the optimal OARST over the MST for the instances in Table 4.4 with 10 or fewer terminals. Thus, the improvement values in Table 4.4 should not be compared with those reported in the literature for standard RST heuristics.

The reader should note that B3S produces trees roughly as good as, and sometimes better than, those produced by G3S. In addition, note that G4S consistently produces better trees than G3S, but that the difference in running times is not nearly as pronounced

	G3S		B3S		G4S	
n	Qual.	Time	Qual.	Time	Qual.	Time
4	7.83	0.25	7.83	0.25	8.19	0.40
5	8.58	0.48	8.59	0.46	9.21	0.97
6	8.55	0.81	8.61	0.75	9.12	1.81
7	8.81	1.26	8.82	1.10	9.35	2.94
8	8.44	1.92	8.40	1.56	9.02	4.55
9	8.74	2.75	8.72	2.10	9.33	6.61
10	9.02	4.17	9.03	2.96	9.53	9.82
12	8.72	7.59	8.69	4.78	9.14	17.64
14	8.93	13.63	8.90	7.48	9.40	31.58
16	8.99	22.66	8.95	11.51	9.48	51.33
18	9.03	34.63	9.04	16.15	9.46	78.79
20	9.02	50.21	8.99	21.53	9.43	112.9

Table 4.4: Average result quality (percent improvement over MST) and running time (in seconds) for the heuristics.

n	4	5	6	7	8	9	10
Qual.	8.19	9.48	9.46	9.95	9.87	10.10	10.32

Table 4.5: Improvement of optimal OARST over MST for test instances.

as one would expect from their relative time complexities—in particular, the running time of G4S *decreases* relative to the running time of G3S as n increases. We attribute this phenomenon to the reduction technique described above.

Note that the worst-case ratio of the length of an MST to the length of an optimal Steiner tree (called the *Steiner ratio*; see also Section 2.2.1) is 2 for the OARST problem [108]. All three of these heuristics always produce trees at least as short as the MST, and thus produce

trees that are no more than twice the length of an optimal tree. In practice, of course, their performance is rarely that bad.

Aside from allowing computation of optimal OARSTs, Theorem 4.2 has important implications with respect to approximate OARSTs. Since the escape graph is guaranteed to contain an optimal OARST, any approximation algorithm for the graph Steiner tree problem with an approximation bound of $\tilde{\rho}$ is an approximation algorithm with the same bound for the OARST problem. As mentioned above, an MST is an approximate solution to the graph Steiner tree problem, and thus the OARST problem, with $\tilde{\rho} = 2$. Recently researchers have devised approximations with better bounds for the graph Steiner tree problem: Zelikovsky [159] describes an algorithm with $\tilde{\rho} = 11/6 \approx 1.83$, and Berman and Ramaiyer [12] improve Zelikovsky's bound to $\tilde{\rho} = 16/9 \approx 1.78$, with better approximations possible at the expense of increased running time. Theorem 4.2 implies that these results provide equivalent approximations for the OARST problem.

4.3.4 Other heuristics

It is possible to compute heuristic OARSTs without using a graph model. For example, the standard routing techniques of global followed by detailed routing do exactly that. However, such algorithms are designed to find a feasible routing of a large number of nets, rather than a good routing of a single net. A detailed discussion of these techniques is beyond the scope of this chapter; interested readers are referred to Preas and Lorenzetti [123].

Also, as mentioned previously, any of the two-terminal interconnection techniques described in Section 4.2 can be used to compute heuristic OARSTs.

Chen [21] describes a somewhat different technique. He describes an algorithm that, given any heuristic OARST, performs iterative improvements that possibly reduce the length of the OARST without changing its topological structure. If certain conditions are met, then the algorithm computes an *optimal* OARST with the given topology.

4.4 Special Cases

One common solution technique when examining NP-complete problems is to find special cases that might be solvable in polynomial time. In this section we examine two special cases. In the case where all terminals lie on the perimeters of obstacles—which is often true in practice—we observe that the problem remains NP-complete, though in some cases such problems can be solved more efficiently than general OARST problems. In the case where all terminals lie on the perimeter of the routing region, forming a switchbox with obstacles, we show that the OARST problem is solvable in polynomial time.

4.4.1 Terminals on obstacle perimeters

It is easily shown that the OARST problem remains NP-complete if all terminals lie on obstacle perimeters. The proof is by reduction from the standard RST problem. Given an RST instance, simply attach to each terminal a square of side length ϵ, where ϵ is very small. The escape graph for such an OARST instance is equivalent to the Hanan grid graph (see Section 4.3) with each segment replaced by two or three parallel segments at distance ϵ from one another. For sufficiently small ϵ (or by scaling the original terminals), an optimal solution to this OARST problem provides an optimal solution to the original RST instance. Thus, the OARST problem remains NP-complete if all terminals lie on obstacle perimeters.

However, if the number of obstacles is small relative to the number of terminals, then a result due to Bern [13] allows computation of an optimal OARST more efficiently than by applying the Dreyfus-Wagner algorithm to the escape graph (see Section 4.3.3). Among the results in Bern's paper is an algorithm for computing optimal Steiner trees in a planar graph in which all the terminals lie on the boundaries of a small number of faces. Each obstacle forms a face in the escape graph, so if all terminals lie on obstacle perimeters, then they all lie on the boundaries of the corresponding faces. If there are f obstacles (i.e. the terminals lie on the boundaries of f faces), then Bern's algorithm finds an optimal OARST in $O(mn^{2f+1} + (m \log m)n^{2f})$ time. This improves upon the straightforward application of

the Dreyfus-Wagner algorithm if f is less than $O(n/\log n)$, i.e., if there are, on average, more than $O(\log n)$ terminals on the perimeter of each obstacle.

4.4.2 Switchboxes with obstacles

Another special case of the OARST problem is when the terminals lie on the perimeter of the routing region. This special case is roughly analogous to the standard RST problem when the terminals lie on the perimeter of a convex polygon. In the standard RST problem, this restriction renders the problem solvable in polynomial time. For example, Cohoon, Richards, and Salowe [32] describe an algorithm to compute an optimal RST in linear time for a set of terminals that lie on the perimeter of a rectangle. More generally, Richards and Salowe [129] present an algorithm that computes, in $O(k^4 n)$ time, an optimal RST for terminals on the perimeter of a k-sided convex polygon. Cheng, Lim, and Wu [22] describe an algorithm that computes an optimal RST of such an instance in $O(n^3)$ time regardless of k (note that this is an improvement if $k = \Omega(\sqrt{n})$).

Mirayala, Hashmi, and Sherwani [118] present a linear-time algorithm that computes an optimal OARST when the terminals lie on the perimeter of a rectangle and there is one rectangular obstacle. They also present an approximation algorithm for the more general case where there is any number of rectangular obstacles. If τ is an OARST computed by their approximation algorithm and τ^* is an optimal OARST, then they prove that $\|\tau\| \leq \|\tau^*\| + w$, where w is the maximum length of any obstacle side.

Chiang, Sarrafzadeh, and Wong present algorithms that compute an optimal OARST if the terminals lie on two parallel lines (i.e. on opposite sides of a channel) [24] and if the terminals lie on the perimeter of a rectangle [25]. These algorithms run in time linear in the number of terminals but exponential in the number of obstacles.

Despite these exponential-time results, the OARST problem is solvable in polynomial time if the terminals lie on the perimeter of the routing region, regardless of the shape of the routing region. Recall from Section 4.2.2.1 that the escape graph contains an optimal

: The escape graph for an instance with terminals on the perimeter of
g region.

time. However, their algorithm is not applicable to a grid graphs with "holes," such as most escape graphs.

4.5 Conclusions and Future Work

We have proven a theorem (Theorem 4.2) that extends Hanan's theorem to the presence of obstacles. The theorem states that a graph called the escape graph, which is constructed from the terminals and obstacle border segments, is guaranteed to contain an optimal OARST. This theorem allows, for the first time, computation of *optimal* OARSTs in time that is a function of the input size rather than the routing area.

We have also presented algorithms that quickly compute optimal OARSTs for three- and four-terminal nets and algorithms that compute good heuristic solutions for larger nets.

We have also examined a number of special cases of the OARST problem. We prove that the OARST problem remains NP-complete when all terminals lie on obstacle perimeters, though in some cases such problems can be solved more efficiently than the general OARST problem. We also prove that the OARST problem is solvable in polynomial time if all terminals lie on the perimeter of the routing region. This result is particularly interesting since exponential-time algorithms have appeared in the literature.

Of course, many interesting avenues of further research remain.

4.5.1 Graph reductions

One interesting open problem is whether one can devise a graph with fewer than $O(c^2)$ vertices that is guaranteed to contain an optimal OARST (or equivalently, a graph reduction that is proven to remove $\Omega(f(c))$ vertices). Such a result would have significant impact on the standard rectilinear Steiner tree problem as well, since to our knowledge it has not been proven that a graph with fewer than $O(n^2)$ vertices exists that is guaranteed to contain an optimal RST.

Winter [151] has devised a number of graph reductions intended for the standard RST problem. These reductions appear empirically to be much more effective than the standard reductions such as the dimension reduction test described in Section 4.3.1. Furthermore, the reductions are applicable to other rectilinear graphs such as the escape graph, and Winter [150] conjectures that they will be at least as effective in this domain as for the standard RST problem. The application of Winter's reductions to escape graphs should be empirically examined.

4.5.2 Computing optimal OARSTs

A *full Steiner tree* is one in which every terminal is a leaf. A *full set* of terminals is one for which every optimal Steiner tree is a full Steiner tree. For the standard RST problem, Hwang's theorem (Theorem 2.2 on page 14) indicates that an RST of a full set has only one of two simple topologies, enabling computation of an optimal full Steiner tree in linear time. This characterization enables the construction of algorithms that compute optimal rectilinear Steiner trees more efficiently than by using an algorithm for the graph Steiner problem ([131] and Chapter 3).

Unfortunately, no such characterization is known for the OARST problem. In particular, Hwang's theorem does not hold in the presence of obstacles. For example, Hwang's theorem implies that the Steiner points in a full Steiner tree induce a chain. This is not the case for the OARST problem; Figure 4.7 illustrates an instance for which the optimal OARST is a full Steiner tree but in which the Steiner points do not induce a chain.

In Section 4.4.1 we observed that the OARST problem remains NP-complete if all terminals lie on obstacle perimeters, in which case every optimal Steiner tree is a full Steiner tree. This would seem to indicate that the problem of computing a full OARST is NP-complete.

Even in the absence of characterizations as strong as Hwang's theorem, it might be possible to use the geometric structure of the escape graph to compute optimal OARSTs more

Figure 4.7: A counterexample to Hwang's theorem.

by simply applying the Dreyfus-Wagner algorithm for the graph Steiner
o the escape graph. For example, Hwang's theorem does not apply in a
grid graph with a non-convex boundary. However, Kaufman, Gao, and Thu-
use the geometric structure of the grid graph to implement tie-breaking rules
from consideration many Steiner trees of equal length. The resulting algo-
than the 1-outerplanar version of the Dreyfus-Wagner algorithm. We believe

5

The Power-p Steiner Tree Problem

Many VLSI routing applications, such as routing to minimize estimates of electrical delay [14, 34, 77, 91], involve the computation of geometric Steiner trees in which the weight of an edge is a nonlinear function of its length. Consideration of this type of application inspired us to introduce the *power-p Steiner tree* problem. The power-p Steiner tree problem is stated as follows: given a set of terminals in the plane, find a geometric Steiner tree that minimizes the sum of the edge lengths each raised to the p power. In addition to the VLSI applications mentioned above, nonlinear Steiner tree problems are often studied in the operations research community under the name *facility location* [52, 78, 147].

A special case of the power-p Steiner tree problem is the *bottleneck Steiner tree* problem, which is to find a geometric Steiner tree that minimizes the length of the longest edge. The bottleneck Steiner tree problem is the limiting case of the power-p problem as p approaches infinity. Again, bottleneck Steiner trees find application in VLSI routing [23, 76] as well as in facility location [38, 50, 113].

Our first set of results concerns computation of optimal power-p Steiner trees. We give an algorithm for computing optimal Euclidean power-2 Steiner trees and describe the difficulties in applying it to the rectilinear case. We give an algorithm for computing optimal

Earlier versions of portions of this chapter appear in Ganley and Salowe [61].

rectilinear bottleneck Steiner trees, and describe the difficulties encountered when trying to generalize this algorithm to the Euclidean case. We also give an algorithm that computes a rectilinear Steiner tree with minimum bottleneck weight and that among all trees with minimum bottleneck weight has minimum total length. We also consider computation of power-p Steiner trees for $p > 2$, giving evidence that the problem is essentially unsolvable for large p.

Our second set of results concerns approximate power-p Steiner trees. We first consider the *power-p Steiner ratio*. The power-p Steiner ratio is the maximum ratio of the weight of a minimum power-p spanning tree to the weight of an optimal power-p Steiner tree. The Steiner ratio is important in analyzing the performance of approximation algorithms for Steiner tree problems. We prove bounds on the power-p Steiner ratio for general p, and we give the exact value of the bottleneck Steiner ratio. We also provide a fully polynomial-time approximation scheme for the bottleneck Steiner tree problem for a given topology.

5.1 Basics

We assert that every Steiner point has degree at least 3, since otherwise one could add a large number of degree-2 Steiner points to reduce the weight of the tree to an arbitrarily small value. Soukup [140] shows that given this restriction, for a class of edge weight functions that includes the power-p weight function, the degree of every Steiner point in an optimal power-p Steiner tree is exactly 3.

We consider power-p Steiner trees with respect to both the Euclidean and rectilinear distance metrics. We call these problems the *Euclidean power-p Steiner tree (EpST)* problem and the *rectilinear power-p Steiner tree (RpST)* problem. Note that E1ST and R1ST, respectively, are the standard Euclidean and rectilinear Steiner tree problems. By this notation, E∞ST and R∞ST would denote the Euclidean and rectilinear bottleneck problems, but for clarity we instead refer to these as the EBST and RBST problems, respectively.

The *topology* of a Steiner tree is a graph $\Upsilon = (V, E)$ (for our purposes, Υ is always a tree) that specifies the graph structure of the Steiner tree. That is, the vertex set V is the set of terminals and Steiner points, and the edge set E specifies the interconnections of the terminals and Steiner points. However, a topology does not specify the geometric locations of the Steiner points nor the edge lengths.

The power-p weight $\omega_p(\tau)$ of a tree τ is given by

$$\omega_p(\tau) = \sum_{(a,b)\in\tau} \|a-b\|^p.$$

The weight $\omega_B(\tau)$ of a bottleneck Steiner tree τ is

$$\omega_B(\tau) = \lim_{p\to\infty} \omega_p(\tau) = \max_{(a,b)\in\tau} \|a-b\|.$$

Recall that $\|\tau\|$ is the geometric length of the tree τ, i.e. $\|\tau\| = \omega_1(\tau)$. The weight of a Steiner tree τ is abbreviated $\omega(\tau)$ when the subscript is clear from context.

5.2 Computing Optimal Power-p Steiner Trees

This section presents a number of results concerning computation of optimal power-p Steiner trees. Specifically, we present an algorithm for computing optimal E2STs and describe the difficulties in applying it to the rectilinear metric. We present evidence that the EpST problem is essentially unsolvable for $p \geq 5$. We then present an algorithm for computing optimal RBSTs and RBSTs that minimize both bottleneck weight and total weight, and we briefly discuss the difficulties in applying these algorithms to the EBST problem. Finally, we present a fully polynomial-time approximation scheme for the bottleneck Steiner tree problem in any distance metric.

5.2.1 Euclidean power-2 Steiner trees

A problem similar to the E2ST problem has been considered in the operations research community under the name *quadratic facility location*. In that context, Eyster and White [52]

give an iterative procedure that approximates an optimal solution to a quadratic facility location problem, and White [147] proves a result similar to our Theorem 5.1. However, ours is the first known algorithm for computing optimal E2STs.

Soukup [140] shows that for three terminals, the unique Steiner point is located at the arithmetic mean of the coordinates of the terminals.

Lemma 5.1 (Soukup [140]) *For three terminals t_1, t_2, and t_3, the Steiner point s that minimizes the length of the E2ST is given by*

$$s = \left(\frac{x_{t_1} + x_{t_2} + x_{t_3}}{3}, \frac{y_{t_1} + y_{t_2} + y_{t_3}}{3} \right).$$

We use Lemma 5.1 to devise a linear-time algorithm that produces an optimal E2ST with a given topology.

Theorem 5.1 *For n terminals in the plane and a topology Υ, an E2ST optimal with respect to Υ can be computed in $O(n)$ time.*

Proof: For each Steiner point in the topology Υ, set up an equation that expresses its optimal position relative to its neighbors. That is, if the Steiner point s has neighbors a, b, and c, then assemble the equations

$$x_s = \frac{x_a + x_b + x_c}{3} \quad \text{and} \tag{5.1}$$

$$y_s = \frac{y_a + y_b + y_c}{3}. \tag{5.2}$$

For neighbors that are terminals, the appropriate x and y values are substituted into the equations; for Steiner points, they are left as variables. These equations are linear, so an optimal solution to the system of linear equations can be computed using, e.g., Gaussian elimination. The graph structure of the system is a tree, so the system can be solved in $O(n)$ time by Gaussian elimination on the leaves of the tree [139]. In addition, the x and y equations are independent, so each of the two systems can be solved separately.

To prove that this construction produces an optimal tree, consider a single quadratic equation expressing the weight of the E2ST as a function of the Steiner point locations:

$$\omega(\tau) = \sum_{(a,b)\in\tau} \|a - b\|^2.$$

The first partial derivative of $\omega(\tau)$ with respect each x_s is identical to Equation (5.1), and similarly the first partial derivative with respect to each y_s is identical to Equation (5.2). Thus, the system of linear equations defined above has the same roots as $\omega(\tau)$.

Furthermore, since each equation has exactly one root, and the second partial derivatives are all positive, the solution to the system of equations is the unique global minimum. Therefore, the algorithm above minimizes the weight of the E2ST, and thus computes an E2ST that is optimal with respect to Υ. □

Theorem 5.1 allows computation of an optimal E2ST for any set of terminals in the plane. The topology of every Steiner tree is a (possibly degenerate) *full topology*, i.e. a topology in which every terminal is a leaf (see Hwang, Richards, and Winter [84]). Thus, an optimal E2ST is computed by applying the construction from Theorem 5.1 to every full topology and taking the shortest resulting tree as the optimum. There are $O(n!)$ full topologies on n terminals [139], so this algorithm computes an optimal E2ST of n terminals in $O(n \cdot n!)$ time. In practice, this algorithm is too inefficient to be applied to more than a few terminals. However, the algorithm of Theorem 5.1 can also be used to compute good heuristic solutions by considering a small number of likely topologies and using the algorithm to compute an optimal E2ST for each.

5.2.1.1 Stability of the E2ST algorithm

One concern that must be addressed is the possibility that the system of linear equations defined in Theorem 5.1 is ill-conditioned. Theorem 5.1 defines a system of linear equations of the form $Ax = b$, where A is an $n-2$ by $n-2$ matrix and x and b are vectors of length $n-2$. Label the Steiner points in the topology Υ as $s_1, s_2, \ldots, s_{n-2}$, and label the

$$A = \begin{vmatrix} 3 & -1 & 0 & 0 \\ -1 & 3 & -1 & -1 \\ 0 & -1 & 3 & 0 \\ 0 & -1 & 0 & 3 \end{vmatrix}$$

Figure 5.1: A topology and its A matrix.

---

s follows:

$$b_i = \sum_{(s_i,t_j)\in\Upsilon} x_{t_j}$$

x coordinates of the Steiner points, and

Proof: The matrix A is clearly symmetric. Gershgorin's theorem indicates that the eigenvalues of A lie in the interval $[0,6]$. Since A is nonsingular, none of its eigenvalues is 0. Therefore, A is symmetric and positive definite, and so its condition number $\kappa(A) = |\lambda_n(A)/\lambda_1(A)|$, where $\lambda_n(A)$ and $\lambda_1(A)$ are, respectively, the largest and smallest eigenvalues of A.

The matrix A has the form $A = 3I - A_\Upsilon$, where A_Υ is the adjacency matrix of the topology graph Υ and I is the identity matrix. Since Υ is a tree, and all trees are bipartite, $\lambda_1(-A_\Upsilon) = \lambda_1(A_\Upsilon) = -\lambda_n(-A_\Upsilon) = -\lambda_n(A_\Upsilon)$ (the entire spectrum of a bipartite graph, considered as a set of points in $\mathcal{R}^1$, is symmetric with respect to reflection about the origin) [36]. Furthermore, since $A = 3I - A_\Upsilon$, each $\lambda_i(A) = \lambda_i(A_\Upsilon) + 3$. Thus, $\kappa(A)$ is given by the following expression:

$$\begin{aligned} \kappa(A) &= \left| \frac{\lambda_n(A_\Upsilon) + 3}{\lambda_1(A_\Upsilon) + 3} \right| \\ &= \left| \frac{\lambda_n(A_\Upsilon) + 3}{3 - \lambda_n(A_\Upsilon)} \right| \end{aligned} \tag{5.3}$$

If A_F is a the adjacency matrix of a forest (a collection of one or more trees) with maximum degree d, then $\sqrt{d} \leq \lambda_n(A_F) \leq 2\sqrt{d-1}$ [66]. The maximum degree of Υ is 3, so $\sqrt{3} \leq \lambda_n(A_\Upsilon) \leq 2\sqrt{2}$. The value of $\lambda_n(A_\Upsilon)$ that maximizes Equation (5.3) subject to these bounds is the upper bound, $\lambda_n(A_\Upsilon) = 2\sqrt{2}$.

Therefore, $\kappa(A) \leq (2\sqrt{2}+3)/(3-2\sqrt{2}) \approx 34$, as desired. □

A rule of thumb is that solving a linear system of equations with a given condition number κ results in a worst-case loss of approximately $\log_{10}\kappa$ digits of accuracy. Thus, the E2ST algorithm of Theorem 5.1 can be used with a loss of at most 2 digits of accuracy.

Figure 5.2 illustrates an optimal E2ST on eight randomly generated terminals.

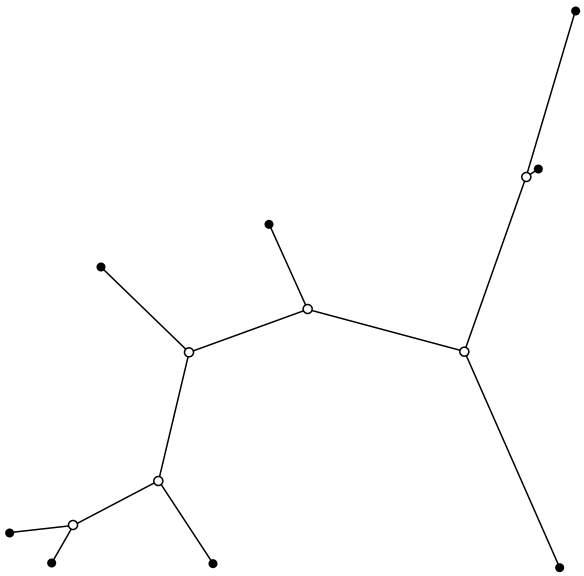


Figure 5.2: An optimal E2ST on eight terminals.

5.2.2 Rectilinear power-2 Steiner trees

The algorithm of Theorem 5.1 cannot be easily modified to compute optimal R2STs, as the formula for the square of rectilinear distance is not continuously differentiable. The formula for the weight ω of an R2ST with three terminals t_1, t_2, and t_3 and a single Steiner point s is:

$$\begin{aligned}\omega &= \sum_{i=1}^{3}(|x_{t_i} - x_s| + |y_{t_i} - y_s|)^2 \\ &= \sum_{i=1}^{3}(x_{t_i} - x_s)^2 + 2 \cdot |x_{t_i} - x_s| \cdot |y_{t_i} - y_s| + (y_{t_i} - y_s)^2.\end{aligned}$$

The middle term of this equation, involving absolute values, is not continuously differentiable, and thus its minimum cannot be computed by finding the roots of its first partial derivatives as in the Euclidean case.

One might attempt to derive a finite (but very inefficient) algorithm by trying each possible value of the derivatives of the absolute values (the derivative of an absolute value is either 1 or -1). If a specific value is substituted for the derivative of each absolute value, then the remaining equations form a system of linear equations. However, the resulting systems can be numerically unstable; for some topologies and choices of derivatives of the absolute values, the A matrix forming the system is singular.

More direct (i.e. non-algebraic) algorithms for computing optimal R2STs might be devised by considering geometric properties of the R2ST problem. We leave R2ST algorithms as a topic for future research.

5.2.3 Power-p Steiner trees for larger p

We now consider the EpST problem for $p > 2$ (for the duration of this section, assume that p is an integer). First, we note that the approach of Theorem 5.1 cannot be applied for $p > 2$, as it would involve solving a general system of polynomials of degree $p - 1$.

We will make a stronger conjecture of unsolvability for $p \geq 5$; however, we must first give some background about solvability of polynomials[2]. A linear equation (i.e., a polynomial of degree 1) can be solved using only basic arithmetic operations (addition, subtraction, multiplication, and division). Most polynomials of degree d with $2 \leq d \leq 4$ cannot be solved using basic arithmetic operations alone but can be solved using basic arithmetic operations and radicals (k^{th} roots). While roots often cannot be computed exactly (since they can be irrational), the expression of a solution using radicals allows the solution to be easily computed to any desired finite precision and allows further symbolic manipulation.

Most polynomials of degree $d \geq 5$ cannot be solved using basic arithmetic operations and radicals. A degree-d polynomial is unsolvable in this sense if its Galois group is an unsolvable group. The best approach to approximating solutions to such polynomials is iterative numerical techniques that converge to the solutions.

[2]Unless cited otherwise, algebra results can be found in standard algebra texts such as Hungerford [81] or van der Waerden [146].

We conjecture that, in this sense, the EpST problem cannot be solved exactly if $p \geq 5$.

Conjecture 5.1 *Solving the EpST problem requires solving an irreducible polynomial of degree $d = p - 1$ if p is even or an irreducible polynomial of degree $d = 2(p-1)$ if p is odd. The Galois group of this polynomial is an unsolvable group. Therefore, the solution to an EpST problem cannot be expressed using basic arithmetic operations and radicals if $p \geq 5$.*

Evidence: We consider a restricted 3-terminal EpST instance; clearly if the conjecture is true for this special case, then it is true for the general EpST problem. Locate the three terminals at $(0,0)$, $(0,1)$, and $(1,0)$. In an EpST, these three terminals will be connected to a single Steiner point located at (x,x), where x minimizes the weight $\omega_p(x)$ of the EpST. The function $\omega_p(x)$ is given by

$$\omega_p(x) = (2x^2)^{p/2} + 2[x^2 + (1-x)^2]^{p/2}.$$

Let $d\omega_p(x)$ be the derivative of $\omega_p(x)$ with respect to x.

If p is even, then $d\omega_p(x)$ is a polynomial of degree $d = p - 1$. If p is odd, then $d\omega_p(x)$ is not, strictly speaking, a polynomial at all, since it includes fractional exponents. However, collecting appropriate terms on either side of the equation and squaring both sides results in a true polynomial of degree $d = 2(p-1)$. We conjecture that this degree-d polynomial is irreducible, and that its Galois group is an unsolvable group. If this is true, then the polynomials are unsolvable using basic arithmetic operations and radicals.

Proving the conjecture requires proving that the general equation for $d\omega_p(x)$ is irreducible and that its Galois group is an unsolvable group. While we have not proved this, we provide some evidence that it is true.

We use the GAP software package [134] to verify that each polynomial is irreducible and that its Galois group is the symmetric group S_d with $d \geq 5$, which is an unsolvable group. These computations indicate that, as suspected, the Galois group of $d\omega_p(x)$ for $p \in \{6, 8, 10, 12, 14, 16\}$ is the symmetric group S_{p-1}, and the Galois group of $d\omega_p(x)$ for $p \in \{5, 7\}$ is $S_{2(p-1)}$. Thus, for $p \in \{5, 6, 7, 8, 10, 12, 14, 16\}$, the E$p$ST problem is not

solvable using basic arithmetic operations and radicals. We conjecture that this is the case for all $p \geq 5$. □

Note that the feasibility of computating of EpSTs for $p \in \{3, 4\}$ remains an open problem.

The RpST problem is difficult to examine algebraically, as discussed in Section 5.2.2, so it is unclear whether these results apply in the rectilinear case.

5.2.4 Rectilinear bottleneck Steiner trees

As described previously, a *bottleneck Steiner tree (BST)* is a geometric Steiner tree in which the length of the longest edge is minimized. The BST problem is the limiting case of the power-p Steiner tree problem as p approaches infinity.

In this section, we consider computation of optimal BSTs under the rectilinear distance metric, i.e., the *rectilinear bottleneck Steiner tree (RBST)* problem.

Drezner and Wesolowsky [44] examine a generalization of the RBST problem where the topology Υ is not necessarily a tree. They give two iterative techniques that asymptotically converge to optimal solutions, but they do not describe the rate of convergence and the algorithms do not produce exact solutions.

For a given topology Υ, a solution to the following linear program is an optimal RBST.

$$
\begin{array}{rcl}
\text{Minimize } \omega & & \text{subject to} \\
\omega & \geq & \Delta x_e + \Delta y_e \text{ for all } e \in \Upsilon \\
\Delta x_e & \geq & x_u - x_v \text{ for all } e = (u, v) \in \Upsilon \\
\Delta x_e & \geq & x_v - x_u \text{ for all } e = (u, v) \in \Upsilon \\
\Delta y_e & \geq & y_u - y_v \text{ for all } e = (u, v) \in \Upsilon \\
\Delta y_e & \geq & y_v - y_u \text{ for all } e = (u, v) \in \Upsilon
\end{array}
$$

This linear program contains $5n - 7$ variables and $10n - 15$ constraints. It is difficult

to determine the time required to solve a given linear program, but applying Khachian's ellipsoid algorithm [95] results in an $O(n^6)$ algorithm (see Ignizio and Cavalier [87]).

Dearing and Francis [38] describe a more involved linear programming formulation that involves solving two linear programs, each of which contains $n-1$ variables and $3n^2-5n$ constraints. They state that their linear programming formulation leads to an $O(n^3 \log n)$ algorithm that computes an optimal RBST with a given topology.

Ichimori [86] describes a parametric search algorithm that computes an optimal RBST for a given topology. However, the algorithm relies on an involved shortest-paths formulation of the problem and has time complexity $O(n^4)$.

We now describe a geometric algorithm that computes an optimal RBST with a given topology in $O(n^2)$ time without the use of linear programming.

Sarrafzadeh and Wong [133] describe a simple algorithm that, given a set of terminals, a topology, and a real value δ, either computes a bottleneck Steiner tree τ with $\omega(\tau) \leq \delta$ or determines that such a tree does not exist. Since our algorithm generalizes theirs, we briefly describe their algorithm.

Arbitrarily choose an internal vertex r to be the root of the topology Υ and direct the edges of Υ away from r. Note that r has three children, and every other internal vertex has two children. Associate a region $R_\delta(s)$ with each vertex s in Υ. For each $R_\delta(s)$, define $R_\delta^+(s)$ to be the set of points that are within distance δ of some point in $R_\delta(s)$.

If s is a terminal, then $R_\delta(s)$ is simply s itself. Otherwise, s is a Steiner point, and $R_\delta(s)$ is computed as follows:

$$R_\delta(s) = \bigcap_{(s,t)\in\Upsilon} R_\delta^+(t). \tag{5.4}$$

The regions are computed in a bottom-up fashion, beginning with the leaves. If $R_\delta(s) = \emptyset$ for any vertex s, then a bottleneck Steiner tree τ with $\omega(\tau) \leq \delta$ does not exist. Otherwise, each Steiner point s is chosen to be any point within the region $R_\delta(s)$, and the resulting bottleneck Steiner tree τ has $\omega(\tau) \leq \delta$.

We now describe an algorithm for the more general rectilinear (L_1) and L_∞ bottleneck problems—namely, given a set of terminals and a topology, compute a bottleneck Steiner tree τ with minimum $\omega(\tau)$ for the given topology. Without loss of generality, consider the L_∞ metric, where the unit disk is a square (the L_1 and L_∞ metrics are equivalent under a 45-degree rotation)[3]. An *isothetic rectangle* is a rectangle each of whose sides is parallel to either the x-axis or the y-axis. The first result is that the region $R(s)$ for every Steiner point is either empty or is an isothetic rectangle, and the second result characterizes the descriptions of these rectangles.

Lemma 5.2 *In the L_∞ metric, every $R(s)$ is either empty or an isothetic rectangle.*

Proof: The proof is by induction on the height of s in Υ. As a basis, if s has height 1, then it is either empty or it is the intersection of two squares. In the latter case, this intersection is a (possibly degenerate) isothetic rectangle. The inductive step holds because the intersection of two isothetic rectangles is either empty or it is an isothetic rectangle. □

The *link distance* between two vertices a and b in a tree is the number of edges in the unique path from a to b. Let $d_\Upsilon(a,b)$ be the link distance between vertices a and b in the topology Υ. Let $T_\Upsilon(a)$ be the set of terminals that are leaves of the subtree rooted at a in Υ. Also, let $S(a,r)$ be the square centered at a with radius r.

Lemma 5.3 *In the L_∞ metric,*

$$R_\delta(s) = \bigcap_{t \in T_\Upsilon(s)} S(t, \delta \cdot d_\Upsilon(s,t)).$$

Proof: The proof is by induction on the height of s in the rooted topology Υ. If s has height 1, then the lemma is clearly true. Now suppose that s has height greater than 1. Let s_1, s_2, and s_3 be the children of s (note that s has three children only if it

[3]The L_∞ distance from a to b is $\|a-b\|_\infty = \max\{|x_a - x_b|, |y_a - y_b|\}$

is the root). By the inductive hypothesis, $R_\delta(s_i) = \bigcap_{t \in T_\Upsilon(s_i)} S(t, \delta \cdot d_\Upsilon(s_i, t))$. By Equation (5.4), $R_\delta(s) = \bigcap_{i=1}^{3} R_\delta^+(s_i)$. We now show that $R_\delta^+(s_i) = \bigcap_{t \in T_\Upsilon(s_i)} S(t, \delta + \delta \cdot d_\Upsilon(s_i, t))$. Since $d_\Upsilon(s, x) = 1 + d_\Upsilon(s_i, t)$ for all terminals in the subtree rooted at s_i, this implies the lemma.

Clearly $R_\delta^+(s_i)$ is contained within $\bigcap_{t \in T_\Upsilon(s_i)} S(t, \delta + \delta \cdot d_\Upsilon(s_i, t))$. Conversely, for every point $\alpha \in \bigcap_{t \in T_\Upsilon(s_i)} S(t, \delta + \delta \cdot d_\Upsilon(s_i, t))$, there must be a point $\beta \in R_\delta(s_i)$ such that $\|\alpha - \beta\| \leq \delta$. These two facts are sufficient to prove the lemma. □

Using Lemma 5.3, we devise a linear-time algorithm that decides whether there exists an RBST with bottleneck weight δ. Starting with the leaves (terminals), compute the region $R_\delta(s)$ for each Steiner point s according to Lemma 5.3. The construction of each $R_\delta(s)$ requires constant time, so the entire procedure requires $O(n)$ time. If any region $R_\delta(s)$ is empty, then no RBST exists with bottleneck weight δ; otherwise, placing each Steiner point s anywhere within region $R_\delta(s)$ results in an RBST with bottleneck weight δ.

We now incorporate this algorithm for computing an RBST with a given bottleneck weight into the *parametric search* framework devised by Megiddo [115]. The resulting algorithm computes the bottleneck weight δ^* of an optimal RBST in $O(n^2)$ time, from which the actual RBST is easily computed by the above algorithm.

Imagine running the modified Sarrafzadeh-Wong algorithm for the (unknown) bottleneck value δ^*. Each step of the algorithm involves computing a region $R_{\delta^*}(s)$ by intersecting two isothetic rectangles (or three, if s is the root of Υ). The key concept that enables the use of the parametric search paradigm is that the actual value of δ^* is not needed to perform these computations! Intersecting two isothetic rectangles involves examining each pair of top, bottom, left, and right border segments to determine on which side of one another they lie. These decisions can be made without knowing the value of δ^*.

Consider a Steiner point s and two of its children s_1 and s_2 in Υ. The descriptions of the regions $R_\delta(s_1)$ and $R_\delta(s_2)$ have already been computed; they are squares $S(t_1, \delta \cdot d_\Upsilon(s_1, t_1))$ and $S(t_2, \delta \cdot d_\Upsilon(s_2, t_2))$ for two particular terminals t_1 and t_2. The top border of $R_\delta(s)$

is colinear with whichever of the top borders of these two squares is above the other for bottleneck weight δ^*. If the top border of $R_\delta(s_1)$ is above the top border of $R_\delta(s_2)$, then the top border of $R_\delta(s)$ is a segment of the line

$$y = y_{t_1} + \delta \cdot d_\Upsilon(s, t_1). \tag{5.5}$$

Similarly, if the top border of $R_\delta(s_2)$ is above the top border of $R_\delta(s_1)$, then the top border of $R_\delta(s)$ is a segment of the line

$$y = y_{t_2} + \delta \cdot d_\Upsilon(s, t_2). \tag{5.6}$$

Assume without loss of generality that the top border of $R_\delta(s_1)$ is above the top border of $R_\delta(s_2)$. Now consider the value δ', which is the bottleneck weight value at which the top borders of $R_{\delta'}(s_1)$ and $R_{\delta'}(s_2)$ are colinear. The value of δ' is found by setting Equations (5.5) and (5.6) equal to one another:

$$y_{t_1} + \delta \cdot d_\Upsilon(s, t_1) = y_{t_2} + \delta \cdot d_\Upsilon(s, t_2).$$

Solving for δ' yields

$$\delta' = \frac{y_{t_1} - y_{t_2}}{d_\Upsilon(s, t_2) - d_\Upsilon(s, t_1)}.$$

Use the algorithm for fixed bottleneck weight to compute an RBST with bottleneck weight δ'. If no such tree exists, then $\delta^* > \delta'$, and thus the top border of $R_\delta(s)$ corresponds to the top border of $R_\delta(s_1)$. If one of the regions $R_{\delta'}(s)$ is degenerate (i.e., either a point or a line), then $\delta^* = \delta'$. In this case, no further computation is necessary, as the optimal tree has been found. Otherwise, $\delta^* < \delta'$, implying that the top border of $R_\delta(s)$ corresponds to the top border of $R_\delta(s_2)$.

Repeat this process for the right, left, and bottom borders of $R_\delta(s)$. Since each region has four borders, a constant number of these computations is required for each choice of s. If s is the root, then it has three children instead of two, and we must consider each pair of children. There are three such pairs, so the computation for s still requires a constant

number of computations. Each computation involves the application of the linear-time algorithm for computing an RBST with a given bottleneck weight. Since the algorithm is applied a constant number of times for each Steiner point, the total time complexity is $O(n^2)$.

When the algorithm terminates, a description of each $R_{\delta^*}(s)$ has been computed. This description consists of an intersection of four squares $S(t_i, \delta^* \cdot d_\Upsilon(s, t_i))$ for four particular terminals t_i. Given these descriptions, it is straightforward to determine the actual value of δ^* for which every region is nonempty.

As discussed for the E2ST algorithm in Section 5.2.1, the algorithm described above can be used to compute an optimal RBST by enumerating all full topologies. The resulting algorithm has time complexity $O(n^2 \cdot n!)$. The algorithm can also be used to compute good heuristic solutions by applying it to a small number of likely topologies.

It is difficult to generalize these results to the Euclidean metric. While the algorithm of Sarrafzadeh and Wong [133] provides the required decision procedure for a given δ, its control flow does not appear to depend on comparisons involving low-degree polynomials in δ (the degree seems to be exponential in the depth of the topology). It thus does not meet the technical conditions necessary to apply the parametric search technique [115], so it is unclear how to choose each new value δ' to be examined in the parametric search.

5.2.5 Minimum-length rectilinear bottleneck Steiner trees

For many applications, it is desirable to compute a Steiner tree that not only has low bottleneck weight but also low total length. In this section, we describe a modified version of the RBST algorithm of Section 5.2.4 that computes a Steiner tree that has minimum bottleneck weight and that among all Steiner trees with minimum bottleneck weight has minimum total length. Such trees find application in VLSI routing; for example, Chiang, Sarrafzadeh, and Wong [23] perform VLSI routing using Steiner trees that have minimum

bottleneck weight (for a more tractable but less realistic definition of bottleneck weight) and low total weight.

The linear programming formulation of the RBST problem given in Section 5.2.4 is easily modified to minimize any convex combination of bottleneck weight and total length. However, we can modify our geometric algorithm to obtain solutions more efficiently.

The RBST algorithm of Section 5.2.4, given a set of terminals and a topology Υ, computes the optimal bottleneck weight δ^* of an RBST with topology Υ. Application of the algorithm of Sarrafzadeh and Wong [133] results in a region $R_{\delta^*}(s)$ for each Steiner point s such that placing each s anywhere within $R_{\delta^*}(s)$ produces an RBST with bottleneck weight δ^*. We now describe an algorithm that chooses the coordinates of each Steiner point s within $R_{\delta^*}(s)$ such that the resulting Steiner tree not only has bottleneck weight δ^*, but that among all such trees has minimum total length. We call such a tree a *minimum-length rectilinear bottleneck Steiner tree (MLRBST).*

For clarity, denote the region $R_{\delta^*}(s)$ for each Steiner point s simply as $R(s)$.

Each region $R(s)$ is an isothetic rectangle whose sides are four line segments, some of which may have length 0. Every Steiner point s has degree 3; call its neighbors s_1, s_2, and s_3. Extend each of the segments that form the sides of $R(s)$, $R(s_1)$, $R(s_2)$, and $R(s_3)$ maximally, and let P be the set of intersection points. Let $P(s)$ be the set of points in P and within $R(s)$, i.e., $P(s) = P \cap R(s)$. If s is a terminal, then $P(s) = R(s) = s$.

We first prove that there exists an MLRBST in which every Steiner point s is in $P(s)$. This result is analogous to Hanan's theorem (Theorem 2.1 on page 14).

Lemma 5.4 *There exists an optimal MLRBST in which every Steiner point s is in $P(s)$.*

Proof: Call a point s that is in $P(s)$ a *grid point*, and call a Steiner point s not in $P(s)$ a *free Steiner point*. Note that every terminal is a grid point.

Assume that every optimal MLRBST contains at least one free Steiner point, and let τ be an optimal MLRBST that contains a minimum number of free Steiner points. Let s be a free Steiner point in τ that is adjacent to two grid points. Such a point must exist, since if

every free Steiner point was adjacent to two or more free Steiner points, then every vertex in the subgraph induced by the free Steiner points would have degree 2 or more and the topology would contain a cycle.

The Steiner point s has degree 3. Let s_1, s_2, and s_3 be the neighbors of s in Υ. There must not be an embedding of the edges (s, s_i) such that two segments adjacent to s overlap, or else the length of τ could be reduced, contradicting the assumption that it has minimum length. Thus, the three segments adjacent to s intersect it from three different directions. Assume without loss of generality that edge (s, s_1) meets s from the top, (s, s_2) meets s from below, and (s, s_3) meets s from the right.

We have selected s such that two of its neighbors are grid points. We now examine two cases.

Case 1: *The point s_3 is a grid point and either s_1 or s_2 is a grid point.* Assume without loss of generality that s_1 is a grid point. Let $s' = (x_{s_1}, y_{s_3})$; note that s' is a grid point. A straightforward case analysis shows that s' satisfies

$$\|s_1 - s'\| + \|s_3 - s'\| \leq \|s_1 - s\| + \|s_3 - s\|.$$

Thus, the MLRBST formed by replacing s with s' has length at most $\|\tau\|$ and has one fewer free Steiner points, contradicting the assumption that τ has a minimum number of free Steiner points.

Case 2: *The points s_1 and s_2 are grid points.* It must be the case that either $x_s = x_{s_1}$ or $x_s = x_{s_2}$, since otherwise there would be an embedding of the edges in which two overlap. Furthermore, by the same argument, it must be the case that $y_s = y_{s_3}$. Since s_3 is a Steiner point, it has three neighbors, and one of its adjacent segments is orthogonal to (s, s_3). Assume without loss of generality that this segment extends downward from s_3, and slide the segment (s, s_3) downward until it is colinear with either the bottom border of $R(s)$ or the bottom border of $R(s_3)$. In either case, the resulting tree has the same length as τ and s is now a grid point, contradicting the assumption that τ contains a minimum number of free Steiner points. □

We now prove a bound on the number of points in each $P(s)$.

Lemma 5.5 *For every Steiner point s in an MLRBST, $|P(s)| \leq 64$.*

Proof: Let s_1, s_2, and s_3 be the neighbors of s. There are eight vertical lines and eight horizontal lines extending from the borders of the four regions $R(s)$, $R(s_1)$, $R(s_2)$, and $R(s_3)$. There can be degenerate cases where two horizontal lines or two vertical lines intersect, in which case $|P(s)| = \infty$, but such degeneracies can be eliminated by perturbation (see Edelsbrunner [47]). If every horizontal line intersects every vertical line and all intersections lie within $R(s)$, then $|P(s)| = 8^2 = 64$. □

This upper bound is met for four concentric rectangles, though it is unclear whether such a configuration can arise in the construction of an RBST.

Given these results, an MLRBST can be efficiently constructed from the output of the RBST algorithm of Section 5.2.4.

Theorem 5.3 *Given a topology Υ on n terminals and a region $R(s)$ for each Steiner point s in Υ, an MLRBST can be computed in $O(n)$ time.*

Proof: By Lemma 5.4, an optimal MLRBST τ is computed by letting each Steiner point s be a point in $P(s)$, chosen such that $\|\tau\|$ is minimized. This computation is accomplished in $O(n)$ time by dynamic programming. Choose an arbitrary internal vertex r in Υ to be the root, and direct the edges in Υ away from r. As before, let $T_\Upsilon(s)$ be the set of terminals in the subtree of Υ rooted at s. For each Steiner point s and each point p in $P(s)$, let $\ell_s(p)$ be the length of an optimal MLRBST for the set $T_\Upsilon(s)$ of terminals, in which s is located at the point p. The values $\ell_s(p)$ are computed for each s in bottom-up fashion, beginning with the Steiner points adjacent to terminals in Υ. Let s_1, s_2, and s_3 be the neighbors of s. For each s and each p in $P(s)$, the value of $\ell_s(p)$ is computed as follows:

$$\ell_s(p) = \min_{\substack{p_1 \in P(s_1) \\ p_2 \in P(s_2) \\ p_3 \in P(s_3)}} \left\{ \sum_{i=1}^{3} \|p - p_i\| + \ell_{s_i}(p_i) \right\}.$$

Since the size of each $P(s)$ is bounded by a constant, this computation is accomplished in constant time. The operation is performed once for each Steiner point s in Υ, and thus the entire computation requires $O(n)$ time. □

Note that like the dynamic programming algorithms described in Chapter 3, the algorithm described in Theorem 5.3 computes only the length of an optimal MLRBST. A similar second pass computes the actual tree given the lengths computed by the first pass, also in $O(n)$ time.

Figure 5.3 illustrates an optimal RBST, an optimal MLRBST, and an optimal R1ST on six randomly generated terminals.

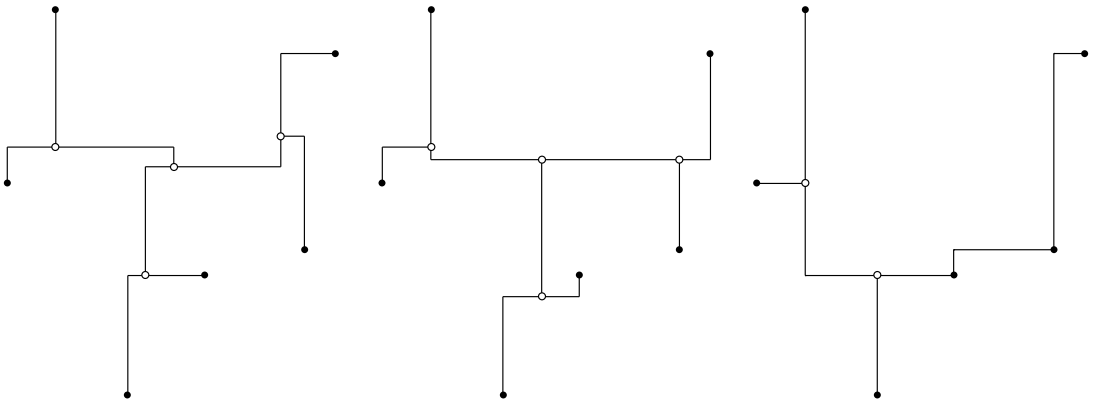


Figure 5.3: (a) An optimal RBST τ_1 with $\omega_B(\tau_1) = 0.27$ and $\|\tau_1\| = 2.19$, (b) An optimal MLRBST τ_2 with $\omega_B(\tau_2) = 0.27$ and $\|\tau_2\| = 2.07$, and (c) An optimal R1ST τ_3 with $\omega_B(\tau_3) = 0.53$ and $\|\tau_3\| = 1.86$.

5.3 Approximate Power-p Steiner Trees

Though it has not been proven that the power-p Steiner tree problem is NP-complete, it it likely that it is (see Section 5.4). Thus, we turn our attention to approximate power-p Steiner trees.

One popular approximation for many Steiner tree problems is a *minimum spanning tree (MST)*. For most variants of the Steiner tree problem, the length of an MST is at most a constant multiple of the length of an optimal Steiner tree. The maximum ratio of the length of an MST to the length of an optimal Steiner tree is called the *Steiner ratio* and is denoted ρ (see also Section 2.2.1).

Define a *minimum power-p spanning tree (MpST)* to be a spanning tree M that minimizes $\omega_p(M)$. It is well known that an MST is also a minimum bottleneck spanning tree, i.e. an M∞ST (see Cormen, Leiserson, and Rivest [35]). In fact, an MST is an MpST for every $p \geq 0$.

Theorem 5.4 *An M1ST is an MpST for every $p \geq 0$.*

Proof: Consider the operation of Kruskal's MST algorithm [101]. First, the edges are sorted according to length. Now, raise the length of each edge to the p power. Since $p \geq 0$, doing so does not change the sorted ordering of the edges. Since the edges are considered in the same order regardless of p, an M1ST is also an MpST for all $p \geq 0$. □

Before examining the power-p and bottleneck Steiner ratios, we prove the following lemma concerning the number of edges in an optimal Steiner tree on a path between two endpoints of an edge in an MST. Let $\{x\}$ denote the fractional part of x, and let $[P(x)]$ be 1 if predicate $P(x)$ is true and 0 if it is false.

Lemma 5.6 *If (a, b) is an edge in an MST, then there is a path in every optimal Steiner tree that contains a and b and contains at most $2\lfloor \lg n \rfloor - [\{\lg n\} < \lg(3/2)]$ edges.*

Proof: Delete the edge (a, b) in the MST, and let T_a and T_b be the sets of terminals in each of the resulting components. Note that if $u \in T_a$ and $v \in T_b$, then $\|u - v\| \geq \|a - b\|$,

or else (a, b) would not be an edge in an MST. Choose $u \in T_a$ and $v \in T_b$ such that u and v have minimum link distance in the optimal Steiner tree τ. We claim that if

$$2^{\lceil \lg n \rceil - 1} < n < 3 \cdot 2^{\lceil \lg n \rceil - 2}$$

(in which case $\{\lg n\} < \lg(3/2)$), then this link distance is at most $2\lceil \lg n \rceil - 1$, and if

$$3 \cdot 2^{\lceil \lg n \rceil - 2} \leq n \leq 2^{\lceil \lg n \rceil}$$

(in which case $\{\lg n\} \geq \lg(3/2)$), then this link distance is at most $2\lceil \lg n \rceil$. In every distance metric in which the triangle inequality holds, these facts imply the lemma.

Consider the former case. Suppose to the contrary that the minimum link distance in an optimal Steiner tree τ between a terminal $u \in T_a$ and a terminal $v \in T_b$ is at least $2\lceil \lg n \rceil$. Remove the edge $e = (c, d)$ on the path from u to v that is $\lceil \lg n \rceil$ links away from u; this breaks τ into two components τ_a and τ_b with u and c in tree τ_a and v and d in tree τ_b.

A terminal t_1 is *closest to* a vertex t_2 if the link distance from t_1 to t_2 is no greater than the link distance from t_2 to any other terminal t_i, $i \neq 1, 2$. We claim that u is closest to c. To prove this, suppose to the contrary that there was some other terminal w closer to c than u. If $w \in T_a$, then w is closer to v than u is to v. On the other hand, if $w \in T_b$, then w is closer to u than v is to u. In either case, the choice of u and v to be closest to one another is contradicted.

Since u has minimum link distance from a and τ_a is a binary tree, τ_a contains at least $2^{\lceil \lg n \rceil - 1}$ terminals. Similarly, τ_b contains at least $2^{\lceil \lg n \rceil - 2}$ terminals, and so τ contains at least $3 \cdot 2^{\lceil \lg n \rceil - 2}$ terminals. However, this contradicts the assumption that $n < 3 \cdot 2^{\lceil \lg n \rceil - 2}$.

The argument for the latter case is analogous. □

5.3.1 The power-p Steiner ratio

Define the *power-p Steiner ratio* ρ_p to be the maximum ratio of the weight of an MpST to the weight of an optimal power-p Steiner tree. That is,

$$\rho_p(n) = \max_{\substack{T \subset \mathcal{R}^2 \\ |T|=n}} \frac{\omega_p(M(T))}{\omega_p(\tau(T))},$$

where $M(T)$ is an MST of a set T of terminals and $\tau(T)$ is an optimal power-p Steiner tree. We now obtain bounds on $\rho_p(n)$; note that the bounds on ρ_p are functions of n, rather than constant as in the power-1 case.

Theorem 5.5 *In the Euclidean metric,*

$$\rho_p(n) \geq \frac{2[\sqrt{3}(\lfloor \lg(n/3) \rfloor + 1)]^p}{2n-3}$$

for $p > 0$.

Proof: Consider an equilateral triangle whose corners are the points $t_0 = (0,0)$, $t_1 = (1,0)$, and $t_2 = (1/2, \sqrt{3}/2)$. Each side of this triangle has length 1. Replace each of the corners t_i of the triangle with a set S_i of $n/3$ terminals within distance ϵ of point t_i, where $0 < \epsilon \ll 1$. We show how to construct an EpST of $S_0 \cup S_1 \cup S_2$ for which the ratio of the weight of the MpST to the weight of the EpST is $2[\sqrt{3}(\lfloor \lg(n/3) \rfloor + 1)]^p/(2n-3)$. Note that this E$p$ST is not necessarily optimal, but its weight clearly cannot be less than that of an optimal EpST, and since we are finding a lower bound, this is sufficient.

The coarse structure of the EpST is as though each of the sets S_i of terminals were a single terminal, as in the original triangle. Thus, there is a single Steiner point s at $(1/2, \sqrt{3}/6)$ that forms the root of three subtrees, each of which has one of the S_i as its leaves. Note that the Euclidean distance from s to each terminal in some S_i is $1/\sqrt{3} + f(\epsilon)$, or essentially $1/\sqrt{3}$.

The subtree for $s \cup S_i$ is a binary tree whose leaves are the terminals in S_i, with an additional edge connecting the root of the tree to s. All edge lengths are equal; therefore, the length of each edge is $1/[\sqrt{3}(\lfloor \lg(n/3) \rfloor + 1)]$. The power-$p$ weight of each edge is then $(1/[\sqrt{3}(\lfloor \lg(n/3) \rfloor + 1)])^p$.

The EpST contains $2n-3$ edges. Thus, its weight is

$$\omega = \frac{2n-3}{[\sqrt{3}(\lfloor \lg(n/3) \rfloor + 1)]^p}.$$

The weight of the MpST is $2 + f(\epsilon)$, or essentially 2. Thus,

$$\rho_p(n) \geq \frac{2[\sqrt{3}(\lfloor \lg(n/3) \rfloor + 1)]^p}{2n-3},$$

as desired. □

Applying an analogous argument under the rectilinear metric leads to the following theorem.

Theorem 5.6 *In the rectilinear metric,*

$$\rho_p(n) \geq \frac{3[2(\lfloor \lg(n/4) \rfloor + 1)]^p}{2n-4}$$

for $p > 0$.

Proof: The proof is completely analogous to the proof of Theorem 5.5 except for the following changes. There are four groups of terminals, at $(0, 1/2)$, $(1/2, 0)$, $(0, -1/2)$, and $(-1/2, 0)$. The Steiner point is at the origin and is the root of four subtrees. The RpST contains $2n-4$ edges, each of length $1/[2(\lfloor \lg(n/4) \rfloor + 1)]$, and the MST has weight 3. The theorem then follows directly. □

We also obtain an upper bound on $\rho_p(n)$.

Theorem 5.7 *In every metric in which the triangle inequality holds,*

$$\rho_p(n) \leq 2^p \lfloor \lg n \rfloor^{p-1}.$$

Proof: Let τ be an optimal power-p Steiner tree for a given p, and let M be an MST. Replace each edge in M with two parallel edges, and construct an Euler tour C_M of M. Replace each edge in τ with two parallel edges, and then construct an Euler tour C_τ of τ

that visits the terminals in the same order as C_M. Divide C_τ into n paths P_i for $1 \leq i \leq n$; the endpoints of each P_i are terminals and the paths contain no terminals other than their endpoints. Let m_i be the number of edges in path P_i, and let e_{ij} denote edge j in path P_i for $1 \leq j \leq m_i$. The weight $\omega(\tau)$ of the tree is thus:

$$\omega(\tau) = \frac{\sum_{i=1}^{n} \sum_{j=1}^{m_i} \|e_{ij}\|^p}{2}.$$

A collection M' of edges s_i, where each s_i has the same endpoints as path P_i, forms a cycle whose weight cannot be smaller than the weight $\omega(M)$ of the MST M. By the triangle inequality, the length of edge s_i is at most $\sum_{j=1}^{m_i} \|e_{ij}\|$. The weight $\omega(M)$ of M thus satisfies

$$\omega(M) \leq \sum_{i=1}^{n} \left(\sum_{j=1}^{m_i} \|e_{ij}\| \right)^p.$$

The power-p Steiner ratio $\rho_p(n)$ then satisfies

$$\begin{aligned}
\rho_p(n) &\leq \frac{2 \sum_{i=1}^{n} \left(\sum_{j=1}^{m_i} \|e_{ij}\| \right)^p}{\sum_{i=1}^{n} \sum_{j=1}^{m_i} \|e_{ij}\|^p} \\
&\leq \max_{1 \leq i \leq n} \frac{2 \left(\sum_{j=1}^{m_i} \|e_{ij}\| \right)^p}{\sum_{j=1}^{m_i} \|e_{ij}\|^p} \\
&\leq 2 \max_{1 \leq i \leq n} \frac{m_i^p}{m_i} \\
&= 2 \max_{1 \leq i \leq n} m_i^{p-1}.
\end{aligned}$$

By Lemma 5.6, since each path P_i corresponds to an edge in the MST, m_i is bounded from above by $2\lfloor \lg n \rfloor$. Thus,

$$\begin{aligned}
\rho_p(n) &\leq 2(2^{p-1} \lfloor \lg n \rfloor^{p-1}) \\
&= 2^p \lfloor \lg n \rfloor^{p-1},
\end{aligned}$$

as desired. □

We use *Mathematica*™ [152] to find the value n^* that maximizes the lower bound for a given value of p. Table 5.1 gives the value of n^* and the Euclidean lower and upper bounds

p	1	2	3	4	5
n^*	3	6	24	96	192
$\rho_p(n^*) \geq$	1.155	2.667	14.78	123.43	1375.3
$\rho_p(n^*) \leq$	2.0	8.0	128.0	3456.0	76832

Table 5.1: Upper and lower bounds on $\rho_p(n)$ in the Euclidean metric.

p	1	2	3	4	5
n^*	4	8	32	128	256
$\rho_p(n^*) \geq$	1.5	4.0	25.6	246.86	3176.1
$\rho_p(n^*) \leq$	2.0	12.0	200.0	5488.0	131072

Table 5.2: Upper and lower bounds on $\rho_p(n)$ in the rectilinear metric.

on $\rho_p(n^*)$ for $1 \leq p \leq 5$. Table 5.2 gives the analogous results in the rectilinear metric. As the tables indicate, an MST may be a good approximation of a power-p Steiner tree for very small p, but its quality as an approximation rapidly deteriorates for larger p.

While our bounds are functions of n, we conjecture that the power-p Steiner ratio is a constant, as in the power-1 case.

Conjecture 5.2 *For every fixed p, the value of ρ_p is bounded from above by a constant.*

The reader should note that the lower bounds of Theorems 5.5 and 5.6 give the correct values of the power-1 Steiner ratio in both metrics.

5.3.2 The bottleneck Steiner ratio

As mentioned in Section 5.3, an MST is also an optimal bottleneck spanning tree. Let $\rho_B(n)$ denote the *bottleneck Steiner ratio*, i.e.,

$$\rho_B(n) = \max_{\substack{T \subset \mathcal{R}^2 \\ |T|=n}} \frac{\omega_B(M(T))}{\omega_B(\tau(T))},$$

where $M(T)$ is an MST of a set T of terminals and $\tau(T)$ is an optimal bottleneck Steiner tree. We now prove the exact value of $\rho_B(n)$ in every distance metric in which the triangle inequality holds.

Theorem 5.8 $\rho_B(n) = 2\lfloor \lg n \rfloor - [\{\lg n\} < \lg(3/2)]$ *in every distance metric in which the triangle inequality holds.*

Proof: Let $f(n) = 2\lfloor \lg n \rfloor - [\{\lg n\} < \lg(3/2)]$. We first show that $\rho_B(n) \geq f(n)$. To do this we exhibit a set of n terminals for which the longest edge in an MST has length 1. We then exhibit a Steiner tree in which the length of a longest edge is $1/f(n)$, which implies that $\rho_B(n) \geq f(n)$.

Consider a set of n terminals divided into two groups of roughly equal size. The terminals in the first group are placed within distance ϵ of the point $(0,0)$, and the terminals in the second group are placed within distance ϵ of the point $(1,0)$, where $0 < \epsilon \ll 1$. We now examine two cases.

Case 1: $2^{\lceil \lg n \rceil - 1} < n < 3 \cdot 2^{\lceil \lg n \rceil - 2}$. Place $2^{\lceil \lg n \rceil - 2}$ terminals in one group T_1 and the remainder in the other group T_2. Note that $2^{\lceil \lg n \rceil - 2} < |T_2| < 2^{\lceil \lg n \rceil - 1}$. Connect T_1 with a binary tree τ_1 whose height (number of edge links) is $\lceil \lg n \rceil - 1$. Connect T_2 by a binary tree of minimum height, all of whose terminals are leaves. The resulting tree τ_2 has height at least $\lceil \lg n \rceil - 1$. Finally, connect the roots of the binary trees with an edge to form tree τ. Tree τ has the property that the shortest link distance between a terminal in T_1 and a terminal in T_2 is at least $2\lceil \lg n \rceil - 1$, and therefore witnesses the lower bound.

Case 2: $3 \cdot 2^{\lceil \lg n \rceil - 2} \leq n \leq 2^{\lceil \lg n \rceil}$. Place $2^{\lceil \lg n \rceil - 1}$ terminals in T_1 and the remainder in T_2. Note that $2^{\lceil \lg n \rceil - 2} < |T_2| < 2^{\lceil \lg n \rceil - 1}$. A similar analysis to that for Case 1 shows

that the shortest link distance between a terminal in T_1 and a terminal in T_2 is at least $2\lceil \lg n \rceil$.

The upper bound, $\rho_B(n) \leq f(n)$, follows directly from Lemma 5.6. Since there are at most $f(n)$ edges in some path through a and b in an optimal BST, the length of the longest edge must be at least $1/f(n)$. □

5.3.3 Better approximate bottleneck Steiner trees

Elzinga, Hearn, and Randolph [50] and Love, Wesolowsky, and Kraemer [113] describe algorithms based on nonlinear optimization that compute an ϵ-approximation to a Euclidean bottleneck Steiner tree for a given topology.

We obtain a simpler ϵ-approximation algorithm for any distance metric by applying the algorithm of Sarrafzadeh and Wong [133] in binary-search fashion (the Sarrafzadeh and Wong algorithm is described in Section 5.2.4). Initialize $a = 0$ and initialize b to the length of the longest edge in an MST. Apply the algorithm of Sarrafzadeh and Wong for $\delta = (b - a)/2$. If a bottleneck Steiner tree τ exists with $\omega(\tau) \leq \delta$, then set $b = \delta$; otherwise, set $a = \delta$. Repeat this process k times. If m is the length of a longest edge in an MST and ω^* is the length of a longest edge in an optimal bottleneck Steiner tree, then this approach computes a bottleneck Steiner tree τ with $\omega(\tau) \leq \omega^* + m/2^k$ in $O(kf(n))$ time, where $f(n) = O(n \log n)$ is the time required for each execution of the Sarrafzadeh and Wong algorithm.

A *fully polynomial-time approximation scheme (FPTAS)* is an algorithm that computes a solution to an instance with input length L that is within ϵ of optimal in time polynomial in L and $1/\epsilon$ [64].

Theorem 5.9 *The algorithm described above is an FPTAS for the BST problem.*

Proof: For a given ϵ, consider the number of iterations required to converge to a solution within ϵ of optimal. That is, we want to find k such that $m/2^k \leq \epsilon$. Solving $m/2^k \leq \epsilon$ for k yields $k \geq \lg(m/\epsilon)$. Thus, the approximation algorithm computes a solution that is

within ϵ of optimal in $O(n \log n \log(m/\epsilon))$ time, where m is the length of a longest edge in an MST. Since $\log(m/\epsilon) = \log(m)\log(1/\epsilon)$, and $\log(m) < L$, the time complexity of the algorithm is polynomial in L and $1/\epsilon$, and is therefore an FPTAS. □

Note that the algorithm computes an approximate BST for a given topology, not one that approximates the overall optimal BST.

5.4 Conclusions and Future Work

We have introduced the power-p Steiner tree problem, which is to find a geometric Steiner tree that minimizes the sum of the edge lengths each raised to the p power.

We have described algorithms for computing optimal E2STs, optimal RBSTs, and rectilinear Steiner trees that minimize a combination of bottleneck weight and total length.

We have provided evidence that the EpST problem cannot be solved exactly if $p \geq 5$.

We have also proven bounds on the power-p Steiner ratio, which measures the quality of an MST as an approximation of a power-p Steiner tree, and we have proven the exact value of the bottleneck Steiner ratio.

Several aspects of the power-p Steiner tree problem remain open, which will be addressed in future research.

5.4.1 NP-completeness

It is not known that any power-p Steiner tree problem is NP-complete (except, of course, for the power-1 problems). However, it seems likely, in light of the fact that many similar problems are known to be NP-complete. Nonlinear versions of the p-center and p-median problems (here p describes the number of Steiner points, not the edge weight function) are analogous to the power-p and bottleneck Steiner tree problems, respectively, with the addition of restrictions on the possible locations of the Steiner points. Hooker [78] states that these problems are NP-complete. Additionally, it is easily shown (for example, by a transformation from *geometric connected dominating set* [110]) that both the power-p and

bottleneck Steiner tree problems are NP-complete if one imposes an upper bound (of less than $n-2$) on the number of Steiner points allowed.

The NP-completeness proofs for the power-1 problems [62, 63] do not easily generalize to the power-p or bottleneck problems. The reason for this difficulty is the absence of the triangle inequality. For the power-1 problems, the NP-completeness proofs exploit the triangle inequality to construct a transformation in which the possible locations of the Steiner points are strictly limited. The lack of the triangle inequality makes such constructions difficult for the power-p or bottleneck problems. It seems that a different approach is required.

5.4.2 Approximate power-p Steiner trees

Currently the best known approximation algorithms for the power-p and bottleneck Steiner tree problems have an approximation bound of $O(\log n)$ (the former given by Theorem 5.7 and the latter given by Theorem 5.8). A topic of ongoing research is to find approximation algorithms with constant bounds, if possible. For the power-p Steiner tree problem, a proof of Conjecture 5.2 would imply that an MST is such an approximation. On the other hand, the bounds given by Theorem 5.8 for the bottleneck Steiner ratio are tight, so a constant approximation will require a different approach.

An approach that has proved effective for other Steiner tree problems is k-restriction. A *k-restricted Steiner tree* is one in which every full tree contains at most k terminals. The notion of k-restriction has been applied to produce approximations for the Steiner problem in graphs and in the rectilinear metric with approximation bounds better than the Steiner ratio [12, 159]. This concept may prove useful for power-p Steiner tree problems as well.

5.4.3 R2ST and EBST algorithms

In Sections 5.2.2 and 5.2.4, respectively, we briefly mention the difficulties in devising algorithms for computing optimal R2STs and EBSTs. We believe that different approaches

than those examined here will be required, which exploit more fully the geometric properties of the problems. Such algorithms are left as a topic for future research.

5.4.4 Power-p Steiner trees for larger p

In Section 5.2.3, we conjecture that the EpST problem is essentially not finitely solvable for $p \geq 5$, and we provide some computational evidence that the conjecture is true. An important topic of ongoing research is to prove or disprove this conjecture and to prove or disprove the analogous result for the RpST problem.

Another topic is to devise approaches to these problems. If the conjecture proves to be true, then a reasonable approach is to devise iterative numerical algorithms that converge to optimal solutions, similar to Smith's [139] for the E1ST problem.

In addition, even if the conjecture is true, computation of E3STs and E4STs remains an open problem.

5.4.5 Multiobjective power-p Steiner trees

Many VLSI routing applications suggest the formulation of multiobjective power-p Steiner tree problems. In Section 5.2.5 we described an algorithm for computing RBSTs with minimum total weight, which find application in VLSI routing [23, 76].

Other important multiobjective power-p Steiner tree problems arise in the computation of Steiner trees with minimum electrical delay. For example, the Elmore estimate of electrical delay [49] is essentially a combination of power-2 weight and path lengths from a designated *source* terminal to the remaining terminals, which are *sinks*.

"O look at the trees!" they cried, "O look at the trees!"

— Robert Bridges, *London Snow*

6

FPGA Placement and Routing

6.1 Introduction

Field-programmable gate arrays (FPGAs) are electronically reconfigurable integrated circuits that provide an inexpensive alternative to the fabrication of custom integrated circuits. The type of FPGA considered here is a *symmetrical array* FPGA typical of those available from Xilinx [157]. A symmetrical-array FPGA consists of a number of identical programmable *logic blocks* arranged in a grid, interconnected by programmable routing resources. A portion of such an FPGA is illustrated in Figure 6.1. The size of an FPGA is given by its x and y dimensions in number of logic blocks. The sets of parallel routing segments that lie between adjacent logic blocks are called *channel segments*; the number of parallel segments in a channel is the *channel width*, denoted W. Logic blocks are depicted as shaded squares. *Connection blocks* connect sets of logic block pins to channel segments. A connection block has a device-specific topology that connects each logic block pin to F_c different channel segments (in Figure 6.1, for example, $F_c = 2$). *Switch blocks*, depicted as white squares in Figure 6.1, connect different sets of channel segments to one another. A switch block contains an interconnect topology that connects each incoming edge to F_s

Earlier versions of portions of this chapter appear in Ganley and Cohoon [4, 55, 58, 59, 60].

Figure 6.1: A symmetrical-array FPGA and the internal topology of a switch block (inset).

edges exiting the other three sides of the switch block. The inset in Figure 6.1 illustrates a switch block with $F_s = 3$; the darkened lines indicate the three possible exit paths for the bottom channel segment entering from the left. FPGAs are available in various values of W, F_c, and F_s.

The physical design phase of FPGA design takes place immediately following the *technology mapping* phase [17]. Technology mapping transforms an electrical description of a circuit into logic elements suitable for mapping onto FPGA logic blocks, as well as a specification of the electrical interconnections among the logic blocks. Thus, the input to the

and programs the wires in the connection blocks and switch blocks to realize the desired interconnections.

Typically, the primary objective in FPGA layout is to minimize the maximum channel width required by the layout. It may also be desirable to minimize total wire length or maximum path length, as these are heuristically related to circuit performance.

For purposes of placement and global routing, it suffices to consider an abstract representation of the FPGA. Construct a *global routing graph* in which the vertices correspond to logic blocks, connection blocks, and switch blocks. A single edge represents each set of parallel channel segments. Since logic blocks to not behave as the other vertices do—the different pins of the logic block are not directly connected to one another—logic block vertices are not explicitly represented. The presence of a logic block pin in a net is implemented by setting the appropriate adjacent connection block vertex to be a terminal. A portion of a global routing graph corresponding to the FPGA illustrated in Figure 6.1 is illustrated in Figure 6.2. The (implicit) logic blocks and their pin edges are shaded, the connection block vertices are filled circles, and the switch block vertices are open circles.

6.1.1 Notation

As mentioned above, the input to the physical design phase is a number c of unplaced logic blocks and a set R of nets. Each net R_i in R specifies a set of vertices in the global routing graph that must be interconnected. The number of nets is $r = |R|$.

6.2 Previous Work

Historically, placement research has been divided into two main camps. *Partitioning-based placement* involves recursive partitioning strategies, such as that used by our system. The other camp is *local search* strategies such as simulated annealing. Until recently, local search techniques were considered inherently superior to partitioning-based methods, but recently

Figure 6.2: An example FPGA global routing graph.

---

ated partitioning-based algorithms such as GORDIAN [97] have provided tion for local search strategies.

of partitioning-based placement are represented by the *min-cut bisection* Günther [68], Breuer [15, 16], and Lauther [102]. In these algorithms, a conceptually drawn down the approximate center of the circuit area. The

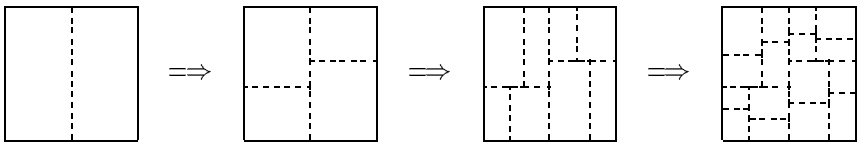


Figure 6.3: An illustration of a min-cut partitioning technique.

the borders between regions so that the placement within each region better reflects its connections to neighboring regions.

All of the min-cut bisection techniques described above perform only placement; global and detailed routing are accomplished afterwards by different methods.

An extension of the min-cut approach is the *quadrisection* approach of Suaris and Kedem [143]. A quadrisection algorithm is similar to a min-cut bisection algorithm except that the vertical and horizontal cuts are performed simultaneously. Suaris and Kedem later revised this technique to simultaneously perform placement and global routing [144].

Mayrhofer and Lauther [114] extended the quadrisection technique to larger grids, and based their partitioning strategies on (heuristic) rectilinear Steiner trees in these grids. However, their algorithm also performs only placement; global and detailed routing are accomplished afterward.

The most direct progenitor of our thumbnail partitioning strategy is the *sharp partitioning* technique of Bapat and Cohoon [8]. Sharp partitioning is a strategy for simultaneous placement and global routing of standard-cell circuits. The sharp partitioning methodology pioneered many of the ideas used in our thumbnail partitioning technique: partitioning based on optimal rectilinear Steiner trees in a small grid (their algorithm uses a 3×3 grid), selection of different optimal Steiner trees to optimize congestion, and a type of terminal propagation.

We further generalize and enhance the sharp partitioning methodology to form the paradigm of *thumbnail partitioning*. We emphasize that though thumbnail partitioning is

present in a thumbnail on a $k \times k$ grid.

In a thumbnail partitioning algorithm, both partitioning-based placement and global routing are based on these thumbnails.

6.4 The MONDRIAN System

We now describe the specifics of our thumbnail partitioning algorithm for FPGA placement and global routing. We call our algorithm MONDRIAN because of the uncanny resemblance between circuit layouts and the works of the artist Piet Mondrian [19, 117].

A crucial aspect of thumbnail partitioning is the computation of optimal thumbnails. In our implementation, $k = 3$, and thus all optimal thumbnails for every possible terminal set are precomputed and stored, and simply looked up during the execution of the algorithm. The extension of the thumbnail partitioning technique to larger values of k is discussed in Sections 6.4.5, 6.8, and 6.9.2.

In overlaying the partitioning template onto the FPGA circuit area, the positions of the cut lines in the partitioning template are chosen to pass through switch blocks, so that the connection blocks adjacent to each logic block all lie in the same region of the template as the logic block itself (see Figure 6.5(a)).

After the circuit area is conceptually divided according to the partitioning template, a *partitioning* step assigns each logic block to one of the regions in the partitioning template. The partitioning step optimizes objective functions based on the thumbnails for the nets.

After the logic blocks are partitioned among the regions in the partitioning template, one of a number of alternate optimal thumbnails is assigned to each net. The assignment of thumbnails to nets is made as to balance the congestion at the intersections of thumbnail edges and cut lines in the partitioning template. Performing this *congestion balancing* step leads to lower congestion in the final global routing.

Once the logic blocks have been assigned to regions in the partitioning template and each net has been assigned a thumbnail, the *virtual terminal assignment* step performs a

type of terminal propagation. At each point where an edge in the thumbnail for a net crosses a cut line in the partitioning template, one of the switch blocks along the corresponding portion of the cut line is chosen to be a virtual terminal for the net. This virtual terminal is added as a terminal in the net, and then the partitioning, congestion balancing, and virtual-terminal assignment steps are repeated recursively for the portion of the circuit that lies within each region of the partitioning template. The addition of the virtual terminals allows the placement and global routing problem within each region to be solved independently of the other regions, such that the union of the solutions for each of the regions is a valid and high-quality solution for the entire FPGA, and such that the global routing of each net corresponds to the topology of its thumbnails.

The recursion stops when a region contains only one logic block. In this *bottom-level routing* phase, the nets that have terminals in the region are routed in the cycle of edges surrounding the logic block.

The steps in the MONDRIAN algorithm are illustrated in Figure 6.5 and are described in detail in the next four sections.

6.4.1 Partitioning

The purpose of the partitioning step is to assign logic blocks to regions in the partitioning template such that the total length of the thumbnails of all nets is minimized. For performance-driven placement, the partitioning algorithm also minimizes source-sink path lengths (see Section 6.5).

In our implementation, the algorithm begins with an arbitrary partition and then improves the partition using *simulated annealing*. Simulated annealing [96] is a general local search strategy that produces high-quality solutions to a wide range of combinatorial optimization problems.

The key concept in local search algorithms is the *neighborhood* of a solution. The neighborhood $\Gamma(S)$ of a solution S is the set of solutions that can be obtained from S by a

(f)

Figure 6.5: The steps in the MONDRIAN algorithm, illustrated for a single net. (a) The FPGA is overlaid with the partitioning template. (b) The logic blocks are assigned to regions. (c) A thumbnail is assigned to the net. (d) Virtual terminals are assigned at intersections of thumbnail edges and region borders.

small perturbation called a *move*. In our algorithm, a move is to swap the positions of two logic blocks that lie in different regions of the partitioning template. The logic blocks on the border of the FPGA are used for input and output, and thus cannot be moved.

A simulated annealing algorithm maintains a current solution S and a temperature t. The temperature is initialized to value t_0. Let $\Phi(S)$ denote the value, for a solution S, of the objective function being optimized; note that in our case, the goal is always to minimize the objective function. Randomly chosen neighbors S' of S are examined, and the difference Δ in the objective function is computed: $\Delta = \Phi(S') - \Phi(S)$. If $\Delta < 0$, then the solution S' has lower cost than S, and S' becomes the new current solution. Otherwise, S' becomes the new current solution with probability $e^{-\Delta/t}$ (where e has its conventional meaning as the base of the natural logarithm); thus, a move that increases the value of the objective function is accepted with probability directly proportional to the temperature and inversely proportional to the magnitude of the change in the objective function. The algorithm also maintains S_B, the best solution seen so far. This process is repeated c times. Then the temperature is decreased according to a *cooling schedule* and the entire loop is repeated. When some *stop criterion* is met, the algorithm halts and returns S_B. Figure 6.6 describes the simulated annealing algorithm in detail.

We use the cooling schedule and stop criterion of Aarts and van Laarhoven [1]. The time complexity of the resulting algorithm is $O(f_m(n)c \log |S^*|)$, where $f_m(n)$ is the time complexity of computing a single move, c is the number of iterations in the inner loop of the algorithm, and S^* is the set of all possible partitioning solutions. For the thumbnail partitioning problem, $f_m(n) = d$, where d is the maximum number of nets that contain any single logic block (i.e., the maximum degree of any logic block). In Xilinx FPGAs, logic blocks have eight pins, and with the exception of the input/output blocks on the border of the FPGA, no two nets can share a logic block pin. Since the input/output blocks cannot be moved, $d = 8$. The value of c, as prescribed by Aarts and van Laarhoven, is the maximum neighborhood size $\max_S |\Gamma(S)|$, which for the thumbnail partitioning problem is $O(n^2)$.

After the partitioning process is complete, the algorithm may assign any one of these alternate optimal thumbnails to each net. The thumbnails are chosen to minimize congestion across the edges in the partitioning template, which tends to minimize congestion in the final global routing.

The congestion balancing problem is formally defined as follows. Each net R_i is to be assigned a thumbnail $\tau(R_i)$. For a given assignment of thumbnails to nets, congestion is measured in the following way: compute a vector s of length $s(k)$ in which each entry $b[e]$ is given by

$$b[e] = |\{R_i : e \in \tau(R_i)\}|.$$

The congestion γ of a given assignment of thumbnails is proportional to the variance of the entries in b:

$$\gamma = \sum_{e=1}^{s(k)} (b[e] - \overline{b})^2,$$

where $\overline{b}$ is the mean of the entries in b.

If k is specified as part of the problem instance, then congestion balancing is NP-complete. Even if all nets have only two terminals, the congestion balancing problem with k as a parameter can be reduced from finding edge-disjoint paths of minimum total length in a grid graph, which has been proven NP-complete [99, 111, 126, 128].

For each fixed k, the congestion balancing problem can be solved exactly in polynomial time. For r nets, the vector s has at most $r^{s(k)}$ different possible values. Thus, an optimal solution to the congestion balancing subproblem is computed in a manner similar to the well-known dynamic programming algorithm for the *partition* problem [64]. Let $B^{(i)}$ denote a set of possible values of b for nets R_1 through R_i. The set $B^{(i+1)}$ is computed by adding each possible thumbnail for net R_{i+1} to each entry $b^{(i)} \in B^{(i)}$. When the process completes, the set $B^{(r)}$ contains all vectors s that correspond to possible solutions to the current instance, of which there are at most $r^{s(k)}$, and the one with minimum γ is then chosen as the solution.

There are at most $2^{s(k)}$ different thumbnails for each net, so this algorithm has time complexity at most $O(2^{s(k)}r^{s(k)+1})$. While this time complexity is polynomial for every fixed k, even for $k = 2$ the resulting time complexity is $O(r^5)$, which is too inefficient to be used in practice.

Instead, we use the following heuristic, similar to the *first-fit decreasing* algorithm for bin packing [89]:

1. Sort the nets in ascending order according to the number of different optimal thumbnails for each net. Let R_{π_i} denote the i^{th} net in this sorted ordering, for $1 \leq i \leq r$.
2. For $i = 1$ to n:
   - ◇ Choose the thumbnail for net R_{π_i} that minimizes the congestion value γ for nets R_{π_1} to R_{π_i}.

Intuitively, the algorithm saves the nets for which there are more choices until later in the algorithm, where they are chosen to compensate for the less avoidable congestion incurred by nets with fewer choices.

While there are clearly at most $2^{s(k)}$ different optimal thumbnails for each net, there are actually far fewer. For $k = 3$, there are at most 192 different optimal thumbnails for each net (see also Section 6.9.2.1), whereas $2^{s(3)} = 4096$. Since the sorting keys can have only a constant number of different values, the congestion balancing step is performed in $O(r)$ time.

6.4.3 Virtual-terminal assignment

Once a thumbnail has been assigned to each net, for each edge e in the thumbnail of each net, the net is assigned a switch block on the template border corresponding to e. This switch block becomes a *virtual terminal* for the net in each of the neighboring regions of the partitioning template, in a similar manner to Bapat and Cohoon [8] and Dunlop and Kernighan [46]. Virtual terminals allow the subproblems within each region of the template to be solved independently of the rest of the circuit, such that the union of the

solutions to the subproblems is a solution for the entire circuit area within the current template. The choice of virtual terminals for each edge in the thumbnail of each net is called *virtual-terminal assignment*.

Consider a pair of adjacent regions in the partitioning template, and let e be the corresponding edge in the template. Let $R(e)$ be the set of nets whose thumbnails contain edge e. The virtual-terminal assignment problem is to assign each of the nets in $R(e)$ to a switch block on the cut line corresponding to e. Let $S(e)$ be the set of switch blocks on the cut line corresponding to e.

Virtual terminal assignment is accomplished in a manner similar to the ΦROUTE algorithm of Spruth, Johannes, and Antreich [142]. Associate a cost with the assignment of each net in $R(e)$ to each switch block in $S(e)$. Initially the cost of assigning net R_i to switchblock s is equal to the sum of the minimum distances from s to the nearest terminals in net R_i in each of the two regions. The cost is increased if s lies at the intersection of two or more regions, as otherwise such s become congestion hot spots.

Now, build a complete bipartite graph B in which the vertices in one partition correspond to nets in $R(e)$ and the vertices in the other partition correspond to switch blocks in $S(e)$. The weight of each edge is the cost of assigning its corresponding net to its corresponding switch block.

Replace each switch block vertex v_s in B with $\lceil |R(e)|/|S(e)| \rceil$ vertices, whose adjacent edges have the same weights as the edges adjacent to v_s. This constrains the assignment such that at most $\lceil |R(e)|/|S(e)| \rceil$ nets are assigned to each switch block, thus minimizing global routing congestion.

The virtual-terminal assignment problem is now solved by finding a *minimum-cost perfect matching* in B, which is computed in $O(|R(e)|^{5/2})$ time by the algorithm of Hopcroft and Karp [79]. Note that this time complexity is at most $O(r^{5/2})$. Once the assignment of switch blocks to nets is accomplished, each switch block is added as a virtual terminal to the corresponding net. The MONDRIAN algorithm is then called recursively for the portion

in the cycle that spans the terminals in R_i, such that *congestion*—the maximum number of nets that use any one of the cycle edges—is minimized. Obtaining good solutions to the bottom-level routing problem is crucial to computing good overall global routing solutions, since it is here that all actual routing is performed.

We now define a more general version of the bottom-level routing problem, where the cycle in which the nets are to be routed contains an arbitrary number of vertices. We call this the *minimum-congestion routing in a cycle (MCRC)* problem. We then prove that the MCRC problem is NP-complete when the cycle contains a variable number m of vertices. We then proceed to describe an integer programming formulation that produces optimal solutions to the MCRC problem, algorithms that compute a solution with congestion C (if one exists) for any fixed C, a heuristic algorithm, and an approximation algorithm that computes a solution with congestion at most four times optimal.

The MCRC problem is defined formally as follows (all index arithmetic on cycle vertex indices is taken to be modulo the length m of the cycle; thus, $a < b$ is understood to imply that vertex v_a is counterclockwise of vertex v_b in the cycle):

MINIMUM-CONGESTION ROUTING IN A CYCLE
INSTANCE: A graph $G = (V, E)$, where $|V| = m$ and $E = \{(v_i, v_{i+1}) : 0 \leq i < m\}$ (i.e., G is a cycle on m vertices), a set R of nets where each $R_i \in R$ satisfies $R_i \subseteq V$, and an integer C.
QUESTION: Is there a set of paths P_i in G such that each P_i spans all $v \in R_i$ and such that $\max_{e \in E} |\{P_i : e \in P_i\}| \leq C$?

6.4.4.1 NP-Completeness

Frank, Nishizeki, Saito, Suzuki, and Tardos [53] show that the MCRC problem is solvable in polynomial time if every net contains 2 vertices. However, their algorithm relies on the planar multicommodity flow theorem of Okamura and Seymour [121], which does not apply if nets can contain more than 2 vertices. We now show that the MCRC problem is NP-complete if nets can contain more than 2 vertices.

We rephrase the MCRC problem as the following equivalent problem:

CYCLE COVER BY MULTIPLE CHOICE PATHS (CCMCP)
INSTANCE: Same as for MCRC, except denote the congestion bound by C'.
QUESTION: Is there a set of paths P_i in G such that each P_i spans two vertices $v_a, v_b \in R_i$, there is no $v_c \in R_i$ such that $a < c < b$, and $\min_{e \in E} |\{P_i : e \in P_i\}| \geq C'$?

The MCRC problem is transformed to the CCMCP problem by setting $C' = |R| - C$. If every edge in the cycle is used by at least C' paths in a solution to the CCMCP problem, then a solution to the MCRC problem is constructed by setting each path to be the complement of the path in the CCMCP solution, thus producing an MCRC solution in which each edge is used by at most $|R| - C'$ paths.

We now prove that the CCMCP problem, and thus the MCRC problem, is NP-complete.

Theorem 6.1 *The MCRC problem is NP-complete.*

Proof: Inclusion in NP is obvious.

To prove NP-hardness, we perform a transformation from *numerical matching with target sums (NMTS)* [64] to CCMCP. The NMTS problem is defined as follows:

NUMERICAL MATCHING WITH TARGET SUMS (NMTS)
INSTANCE: Disjoint sets X and Y, each containing q elements, a size $s(a) \in \mathcal{Z}^+$ for each element $a \in X \cup Y$, and a target vector $< B_1, B_2, \ldots, B_q >$ with positive integer entries.
QUESTION: Can $X \cup Y$ be partitioned into q disjoint sets $A_1, A_2, \ldots, A_q$, each containing exactly one element from each of X and Y, such that for $1 \leq i \leq q$, $\sum_{a \in A_i} s(a) = B_i$?

Note that it must be the case that

$$\sum_{a \in X \cup Y} s(a) \quad = \quad \sum_{1 \leq i \leq q} B_i \tag{6.1}$$

and also that $s(a) < B_i$ for all $a \in X \cup Y$ and $1 \leq i \leq q$ [64].

We now prove that CCMCP is NP-hard by giving a transformation from every instance of NMTS to an instance of CCMCP, such that the answer to the CCMCP problem is "yes" if and only if the answer to the NMTS problem from which it was constructed is "yes."

Let m (the number of vertices in the cycle) be $2q + \sum_{1 \le i \le q} B_i$. Thus, we are performing a *pseudo-polynomial-time transformation*. However, the NMTS problem is NP-complete in the strong sense, meaning that it remains NP-complete even if its input is represented in unary notation. Thus, the pseudo-polynomial-time transformation is acceptable.

We label certain vertices a_i, b_i, and c_i, $1 \le i \le q$, as follows:

$$\begin{aligned} a_i &= 2(i-1) + \sum_{1 \le j < i} B_i, \\ b_i &= a_i + 1, \text{ and} \\ c_i &= b_i + B_i. \end{aligned}$$

This labeling is illustrated in Figure 6.9. Note that there is an edge from c_q to a_1.

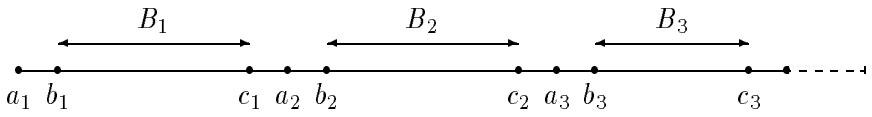


Figure 6.9: Labeling of vertices.

Now, for each element $x \in X$ we construct an *x-net* R_x, and similarly for each $y \in Y$, we construct a *y-net* R_y. For $x \in X$,

$$R_x = \{a_1, b_1 + s(x), c_1, a_2, b_2 + s(x), c_2, \ldots, a_q, b_q + s(x), c_q\}.$$

These vertices are depicted as double circles in Figure 6.10.
Similarly, for $y \in Y$,

$$R_y = \{a_1, b_1, c_1 - s(y), a_2, b_2, c_2 - s(y), \ldots, a_q, b_q, c_q - s(y)\}.$$

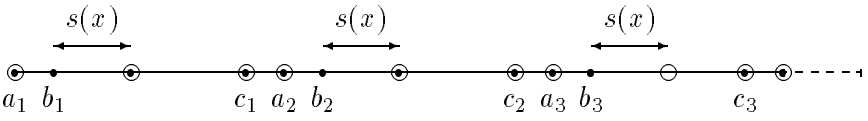


Figure 6.10: An x-net R_x.

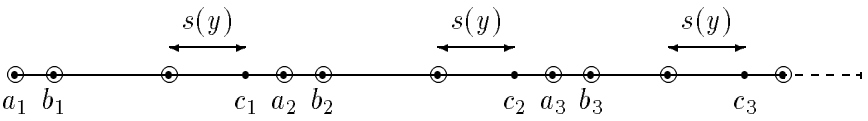


Figure 6.11: A y-net R_y.

These vertices are depicted as double circles in Figure 6.11.

We claim that an instance of CCMCP constructed as described above has a solution with $C' = 1$ if and only if the instance of NMTS from which it was constructed has a solution.

($\Rightarrow$) *If an instance of NMTS has a solution, then an instance of CCMCP constructed from it as described above has a solution with* $C' = 1$. For $1 \leq i \leq q$, if $x \in A_i$, then let the path for net R_x cover the interval $[a_i, b_i + s(x)]$. Similarly, if $y \in A_i$, then let the path for net R_y covers the interval $[c_i - s(y), a_{i+1}]$. The claim follows directly.

($\Leftarrow$) *If an instance of CCMCP, constructed from an instance of NMTS as described above, has a solution with* $C' = 1$*, then the instance of NMTS from which it was constructed has a solution.* Suppose we have a solution to the CCMCP problem with $C' = 1$. The cycle must be covered exactly; if some edge in the cycle were covered by more than one path, then by Equation (6.1), some other edge would not be covered, contradicting the assumption that we have a solution with $C' = 1$. Thus, all that remains to be proven is that the x- and y-nets cover the appropriate intervals. Consider the edge (a_i, b_i)

for any i. We claim that each such edge must be covered by the path $[a_i, b_i + s(x)]$ in some x-net. If an edge (a_i, b_i) were covered by a y-net R_y, then this edge would be the entire path for net R_y, as both a_i and b_i are in every y-net for every i. If the path for R_y were the edge (a_i, b_i), then there would be a portion of the cycle $s(y)$ edges long that would be uncovered, and by Equation (6.1), solution would be impossible, forming a contradiction. A similar argument requires that the path for each R_y cover an interval $[c_i - s(y), a_{i+1}]$ for some i. We construct a solution to NMTS by setting $A_i = \{x, y\}$ for the R_x and R_y that together cover the interval $[a_i, a_{i+1}]$.

We have proven that our constructed instance of CCMCP has a solution if and only if the instance of NMTS from which it was constructed has a solution. Therefore, CCMCP is NP-complete, and thus so is MCRC. □

Since the number m of vertices in the cycle is eight for the bottom-level routing problem in MONDRIAN, NP-completeness does not hold. Indeed, since there are at most $O(r^8)$ possible solutions, the bottom-level routing problem is solvable in polynomial time by a dynamic programming algorithm similar to the one described for congestion balancing in Section 6.4.2. Unfortunately, this $O(r^9)$ algorithm is the best known for the problem, and its time complexity is too high to use in practice. Furthermore, the NP-completeness of the general MCRC problem suggests that there is probably no algorithm that is significantly more efficient than this one.

6.4.4.2 Integer programming formulation

We can construct an *integer programming* formulation of the MCRC problem that quickly computes *optimal* solutions for most instances that arise in the execution of MONDRIAN. Direct the edges in the cycle clockwise. Associate a *flow* value $f_i(v_j, v_{j+1})$ with each net R_i and each edge (v_j, v_{j+1}) in the cycle. For each net R_i, add a *shortcut edge* (v_j, v_k) for each $v_j, v_k \in R_i$ for which there exists no v_q such that $j < q < k$. The directed cycle with example shortcut edges is illustrated in Figure 6.12. Associate a flow value $s_i(v_j, v_k)$ with

: The augmented cycle used in the integer program..

---

hortcut edge (v_j, v_k). Define f_{max} to be the maximum flow along
) edge:

$$f_{max} = \max_{0 \le j < m} \sum_{1 \le i \le n} f_{ij},$$

l flow along all cycle edges:

$$f_{sum} = \sum_{1 \le i \le n} \sum_{0 \le j < m} f_{ij}.$$

Equations (6.2) and (6.3) are flow conservation constraints, which assert that the flow into a vertex v must equal the flow out of v. Equation (6.4) asserts that for each net, the flow into the vertices in that net must be 1; this constraint ensures that all vertices in the net are spanned. Equation (6.5) asserts that the flows on the shortcut edges must total 1, and Equation (6.6) asserts that the flows on the shortcut edges must be integers. These two constraints together force one shortcut edge in each net to have flow 1 and all other shortcut edges in the net to have flow 0. Since the graph is directed, setting the flow along a shortcut edge to 1 and the rest to 0 completely determines the rest of the flow. The routing of net R_i is then the path containing every edge (v_j, v_{j+1}) for which $f_i(v_j, v_{j+1}) = 1$.

It is difficult to determine the time complexity of a given integer program, but in general it is roughly exponential in the number of variables that are constrained to be integers. While this may seem alarming, the number of integer variables in the problems that arise in the execution of MONDRIAN is typically small enough that the program is quickly solved. This subject is discussed further in Section 6.7.

A strength of this integer programming approach is that it finds a solution with minimum congestion, and among all solutions with minimum congestion, it finds a solution with minimum total wire length. Indeed, we make the surprising empirical observation that optimizing total wire length drastically *decreases* the time required to solve the integer program, since it eliminates from consideration large portions of the solution space that have equal congestion but differing total wire length.

6.4.4.3 Polynomial-Time Solution for Fixed Congestion

We can determine in polynomial time whether an instance of the MCRC problem has a solution with $C = 1$. We do so by transforming the MCRC problem to a maximum independent set problem in a circular arc graph. A *circular arc graph* is a graph in which the vertices in the graph correspond to arcs on a circle and in which there is an edge between two vertices if their corresponding arcs intersect.

Let the vertices in the cycle correspond to equally spaced points on a circle. We will now construct from the nets in R a circular arc graph G_R. For each net R_i, denote the vertices in R_i as $u_0, u_1, \ldots, u_{|R_i|-1}$, in clockwise order. For each R_i, add to G_R the arcs $[u_{(j+1) \bmod |R_i|}, u_j]$ for all $0 \leq j < |R_i|$. Note that each arc includes all the vertices in R_i. Note also that if $|R_i| > 2$, then the arcs constructed from R_i are all pairwise intersecting.

Now find a *maximum independent set (MIS)* in G_R. This is accomplished in $O(a^2)$ time, where a is the number of arcs in G_R [69]. The number a of arcs is at most $O(rm)$. Thus, an MIS in G_R is computed in at most $O((rm)^2)$ time. Since all intervals in a net with 3 or more vertices are pairwise intersecting, the MIS cannot contain more than one arc from a net R_i unless $|R_i| = 2$. If it contains both arcs from a net R_i with $|R_i| = 2$, then the size of the MIS is 2, and we can find an MIS that does not contain two arcs from the same net exhaustively in $O(r^2)$ time. Thus, an MIS that does not contain 2 arcs from the same net is computed in $O((rm)^2)$ time.

The instance of MCRC from which G_R was constructed has a solution with $C = 1$ if and only if the MIS contains an arc from every net in R. It requires at most $O(rm)$ time to construct G_R. Thus, the algorithm computes a solution with $C = 1$, or determines that no such solution exists, in $O((rm)^2)$ time.

These ideas can also be used to devise a heuristic for the general MCRC problem. Construct G_R as described above, and find an MIS in G_R. For each net with one of its arcs in the MIS, route the net as the path corresponding to that arc. Repeat for the remaining nets. We call this heuristic the *iterated maximum independent set (IMIS)* heuristic. Since each pass requires $O((rm)^2)$ time, and at most $O(r)$ passes are performed, the IMIS heuristic runs in $O(r(rm)^2)$ time.

For any fixed C, a solution with congestion at most C can be computed in $O((rm)^{C+1})$ time, or it can be determined that such a solution does not exist. Select an edge e in the cycle. Examine every possible routing of each set S of C nets, and for each S, examine each

routing of the nets in S such that the routing uses edge e. For each choice of S and each routing of the nets in S, route the nets not in S according to the unique routing that does not use edge e. If the algorithm terminates without computing an routing with congestion at most C, then no such routing exists.

Each of the r nets has at most m possible routings. Thus, the total number of distinct sets S that must be examined is $\sum_{i=1}^{C} \binom{rm}{i} = O((rm)^C)$. Routing the nets not in S requires an additional $O(rm)$ time, so the algorithm has time complexity $O((rm)^{C+1})$.

6.4.4.4 An Approximation Algorithm

Any pair of edges $e_1 = (v_a, v_b)$ and $e_2 = (v_c, v_d)$ in the cycle G forms a *cut*, i.e., the removal of e_1 and e_2 disconnects G into two components. Call $\{e_1, e_2\}$ a *cut set*. If a net R_i includes vertices in both the interval $[v_b, v_c]$ and the interval $[v_d, v_a]$, then R_i is *cut* by $\{e_1, e_2\}$; otherwise, it is *uncut*. Choose a cut set $\{e_1, e_2\}$ such that the number of nets cut by $\{e_1, e_2\}$ is maximum. Let N denote the number of cut nets. Having chosen a maximum cut $\{e_1, e_2\}$, call $[v_b, v_c]$ the *right* interval and call $[v_d, v_a]$ the *left* interval. Now route every net that is uncut by $\{e_1, e_2\}$ within the interval that contains its vertices. Such a routing is unique and is easily computed in linear time. We now obtain the following lemma concerning the congestion incurred by this routing.

Lemma 6.1 *If $\{e_1, e_2\}$ is a maximum cut that cuts N nets, then the congestion incurred by routing all uncut nets within their corresponding intervals is at most N.*

Proof: The right and left intervals are considered independently; without loss of generality consider the left interval. Suppose that routing all nets contained in the left interval completely within the left interval results in a congestion exceeding N. Let e be an edge whose congestion exceeds N. There is another cut in G consisting of e and any edge e' from the right interval, such that the number of nets that cross the cut $\{e, e'\}$ exceeds N. This contradicts the assumption that $\{e_1, e_2\}$ was a maximum cut, thus proving the lemma. □

Having routed the uncut nets within their corresponding intervals, now route the cut nets arbitrarily. Since there are N cut nets, routing them increases the congestion by at most N. (Note that in practice one would not want to route the cut nets arbitrarily; instead, one might use some heuristic such as the IMIS heuristic described in Section 6.4.4.3.) Thus, the congestion in the routing is at most $2N$. Since N nets must cross the cut $\{e_1, e_2\}$, the optimal congestion is at least $\lceil N/2 \rceil$. Thus, this algorithm computes a solution with congestion at most $2N/\lceil N/2 \rceil \leq 4$ times optimal.

The time complexity of the approximation algorithm is determined as follows. The maximum cut is found in at most $O(rm^3)$ time, as there are $O(m^2)$ cuts to be checked and each is checked in at most $O(rm)$ time. The routings of the uncut nets is computed in $O(rm)$ time. If the cut nets are routed arbitrarily, then they are also routed in $O(rm)$ time, and the total time complexity of the algorithm is $O(rm^3)$. If instead the IMIS heuristic is used to route the cut nets, then routing the cut nets requires at most $O(r(rm)^2)$ time, and the total time complexity of the algorithm is $O(rm^3 + r(rm)^2)$.

6.4.5 Time complexity

The time complexity of MONDRIAN is derived by examining the time complexity of each of its constituent steps (assume for the present that k is a small constant, as in our implementation):

- ◇ The partitioning step requires $O(n^3)$ time.
- ◇ The congestion balancing step requires $O(r)$ time.
- ◇ The virtual-terminal assignment step requires $O(r^{5/2})$ time.

These separate time complexities result in the following recurrence for the total time complexity of the MONDRIAN algorithm:

$$\begin{aligned} T(n,r) &= n^3 + r + r^{5/2} + k^2 T(n/k^2, r) \\ &= n^3 + nr + nr^{5/2} \\ &= O(n^3 + nr^{5/2}). \end{aligned}$$

In addition, the bottom-level routing step is applied once for every logic block, resulting in an additional $O(nf(r))$ time, where $f(r)$ is the time required for the particular bottom-level routing algorithm chosen. If the integer programming algorithm of Section 6.4.4.2 is used, then the exact value of $f(r)$ is difficult to determine since it depends heavily on the number of pins in the bottom-level routing instances. If the IMIS heuristic of Section 6.4.4.3 or the approximation algorithm of Section 6.4.4.4 is used, then $f(r) = O(r^3)$. Thus, the total time complexity of the MONDRIAN algorithm, using the IMIS heuristic or the approximation algorithm for bottom-level routing, is $O(n^3 + nr^{5/2} + nr^3) = O(n^3 + nr^3)$.

In future work, we intend to examine the effect of extending MONDRIAN to $k \times k$ decompositions for larger k. If k is considered to be variable rather than constant as in the above computations, then the time compelxity of each step is:

- ◇ The partitioning step requires $O(n^3 \log k)$ time.
- ◇ The congestion balancing step requires at most $O(r2^{k^2})$ time.
- ◇ The virtual-terminal assignment step requires $O(k^2 r^{5/2})$ time.

It is in the congestion balancing step that the choice of k most drastically affects the time complexity of the algorithm, since the time complexity of congestion balancing as it is currently implemented is exponential in the value of k. To extend MONDRIAN to large k, it will be necessary to perform congestion balancing by some other means than enumerating every optimal thumbnail for each net.

The affects of k in practice are discussed further in Section 6.8.

6.5 Performance-driven Placement and Routing

Since the cost of an FPGA is a function of its channel width, optimizing maximum channel width serves to optimize the cost of implementing a particular design. Optimizing total wire length heuristically optimizes both channel width and circuit performance. However, to overcome the slow speed of FPGAs, it may be desirable to optimize more precise measures of circuit performance.

6.6 Postprocessing

After implementing the algorithm, we observed that often every edge with maximum congestion is contained in a single net. Thus, we add a postprocessing step to the algorithm that reroutes such nets.

Suppose the maximum congestion is C. The postprocessing algorithm finds a net that contains every edge with congestion C, and reroutes it using the *iterated 1-Steiner* heuristic of Kahng and Robins [90]. The graph is modified so that the new route does not use any edge with congestion C or $C-1$. The maximum congestion is recomputed, and the process is repeated until there does not exist a net that contains every maximum-congestion edge.

Experimentally, the postprocessing step typically reduces the channel width by several units.

6.7 Experimental Results

We have implemented MONDRIAN in order to compare it against previous FPGA layout tools. We use code written by Saltzman [132] to solve the matching problem in the virtual-terminal assignment step (see Section 6.4.3) and we use Berkelaar's LP_SOLVE code [11] to solve the integer programs in the bottom-level routing step (see Section 6.4.4).

For testing purposes we use two sets of benchmarks, one for each of the Xilinx 3000-series and 4000-series FPGAs. The 3000-series benchmarks are those used by Brown, Rose, and Vranesic to test their FPGA router, which is called CGE [18]. The 4000-series benchmarks are those used by Lemieux and Brown to test their SEGA router [107] and by Wu and Marek-Sadowska to test their GPB router [156]. We use the same suggested parameters used by these previous works. For the 3000-series benchmarks, $F_c = \lceil 0.6W \rceil$ and $F_s = 6$. For the 4000-series benchmarks, $F_c = W$ and $F_s = 3$. Details on Xilinx FPGAs are found in the Xilinx data book [157].

Table 6.1 summarizes the name, size, and number of nets for each of these benchmarks. The table also gives the channel widths computed by CGE [18] for the 3000-series benchmarks and the channel widths computed by SEGA [107] and GPB [156] for the 4000-series benchmarks.

Name	Size	Nets	CGE	SEGA	GPB
busc	13×12	151	10	—	—
dma	18×16	213	10	—	—
bnre	22×21	352	12	—	—
dfsm	23×22	420	10	—	—
z03	27×26	608	13	—	—
9symml	11×10	79	—	10	9
term1	10×9	88	—	10	10
apex7	12×10	115	—	13	11
alu2	15×13	153	—	11	11
too_large	14×14	186	—	12	12
example2	14×12	205	—	17	13
vda	17×16	225	—	13	13
alu4	19×17	255	—	15	14
k2	22×20	404	—	17	17

Table 6.1: Statistics on the benchmark circuits and channel widths computed by previous tools. The first five benchmarks are Xilinx 3000-series circuits and the last nine are 4000-series circuits.

We have tested MONDRIAN on the benchmark circuits using three different algorithms for bottom-level routing: the integer programming (IP) formulation of Section 6.4.4.2, the IMIS heuristic of Section 6.4.4.3, and the approximation algorithm of Section 6.4.4.4. Detailed routing is performed using the algorithm of Alexander, Cohoon, Ganley, and Robins [3, 4], which developed out of the work of Alexander and Robins [5]. The channel widths resulting from these tests are given in Table 6.2, along with the best channel width computed by previous tools for each benchmark. The most noteworthy feature of these results is

		IP		IMIS		Approx.	
Name	W_P	W	Time (s)	W	Time (s)	W	Time (s)
busc	10	9	77.5	9	47.7	9	47.5
dma	10	9	196.8	11	64.0	10	63.3
bnre	12	11	645.4	12	149.2	11	142.8
dfsm	10	11	528.6	11	220.4	11	224.8
z03	13	13	1042.3	13	371.6	13	358.7
9symml	9	10	56.1	10	35.6	10	36.3
term1	10	7	30.9	7	32.4	7	32.1
apex7	11	9	48.0	9	40.6	9	40.2
alu2	11	10	154.4	11	52.7	11	52.3
too_large	12	11	216.5	11	55.8	11	55.2
example2	13	11	139.7	12	54.8	13	55.5
vda	13	13	1019.3	13	67.7	13	67.9
alu4	14	13	1105.8	13	85.9	13	84.5
k2	17	17	12198.8	17	133.3	17	130.3

Table 6.2: Channel widths computed by MONDRIAN for the benchmark circuits. W is the channel width of the actual detailed routing for the MONDRIAN solution and W_P is the best value computed by previous tools (see Table 6.1).

that they compare quite favorably to those of previous tools. Using integer programming for the bottom-level routing, the channel widths computed by MONDRIAN are superior to those computed by previous tools for 9 of the 14 benchmarks, and are the same as those computed by previous tools for 3 of the remaining benchmarks. The channel widths of the solutions produced by MONDRIAN improve those computed by previous tools by an average of roughly 6.8%.

However, the integer programming formulation, while typically quite fast, can require a great deal of computation time. In particular, the benchmark k2 appears to be somewhat pathological, in that it requires over 3 hours to place and route while the other benchmarks require less than 20 minutes.

Fortunately, the IMIS heuristic and the approximation algorithm compare well with the integer programming formulation for bottom-level routing. For 11 of the 14 benchmarks, using either the IMIS heuristic or the approximation algorithm results in the same channel width as using the integer programming formulation. The differences in channel width and running time between the IMIS heuristic and the approximation algorithm do not appear statistically significant.

6.7.1 Performance-driven placement and routing results

This section reports experimental results for the performance-driven version of MONDRIAN described in Section 6.5. Unfortunately, the previous works against which we compare ourselves in Section 6.7 do not report total wire length or source-sink path lengths. Thus, we compare the performance-driven version of MONDRIAN, which we call MONDRIAN-PD, against the standard Steiner-tree-based version of MONDRIAN.

These results are summarized in Table 6.3, where we observe that a modest (1.0%) increase in wirelength yields a significant (6.7%) decrease in average maximum source-sink path length. The channel widths computed by the two versions of MONDRIAN are comparable. For one benchmark (9symml), MONDRIAN-PD computes a channel width of 9 while MONDRIAN computes a channel width of 10. For another (term1), MONDRIAN-PD computes a channel width of 8 while MONDRIAN computes a channel width of 7. The channel widths for all other benchmarks are the same. We do not believe that the difference in the two algorithms in terms of channel width is statistically significant.

Figure 6.14 illustrates the placed and routed 9symml benchmark.

6.8 Effects of Decomposition Size

Implicit in this and previous work is the idea that performing a $k \times k$ decomposition becomes more effective as k grows. This premise has never been tested in practice.

	Average Wirelength			Average Max Radius		
Name	ST	PD	Δ%	ST	PD	Δ%
busc	9.3	9.1	−2.2	6.6	6.0	−9.1
dma	13.2	13.0	−1.5	8.6	7.7	−10.5
bnre	14.0	14.0	0.0	9.0	7.9	−12.2
dfsm	12.0	12.7	5.8	7.3	7.1	−2.7
z03	13.7	14.1	2.9	8.9	8.7	−2.2
9symml	11.4	10.9	−4.4	7.1	6.1	−14.1
term1	7.0	7.4	5.7	5.2	5.2	0.0
apex7	9.1	9.5	4.4	6.3	6.7	6.3
alu2	12.2	12.5	2.5	7.5	7.2	−4.0
too_large	11.9	11.5	−3.4	8.5	7.2	−15.3
example2	9.3	9.4	1.1	7.2	6.8	−5.6
vda	15.0	15.0	0.0	10.6	9.4	−11.3
alu4	14.5	14.9	2.8	9.5	9.0	−5.3
k2	17.7	17.7	0.0	13.1	12.1	−7.6
Overall	12.2	12.3	1.0	8.2	7.6	−6.7

Table 6.3: Comparison of performance-oriented MONDRIAN-PD against Steiner-tree-based MONDRIAN (ST). The column labeled Δ% gives the percentage change from MONDRIAN to MONDRIAN-PD.

Our implementation of MONDRIAN is parameterized such that the identical algorithm can be run for a 2×2 decomposition rather than a 3×3 decomposition. Table 6.4 compares the channel widths and running times of MONDRIAN using these two values of k, using the approximation algorithm for bottom-level routing. As can be seen in the table, the standard version of MONDRIAN using $k = 3$ substantially outperforms the version with $k = 2$ for almost all the benchmarks.

The time complexity of MONDRIAN increases with larger k. Surprisingly, however, the version of MONDRIAN using a 2×2 decomposition often requires more running time than the standard 3×3 version. This effect is caused by the bottom-level routing algorithm: since the

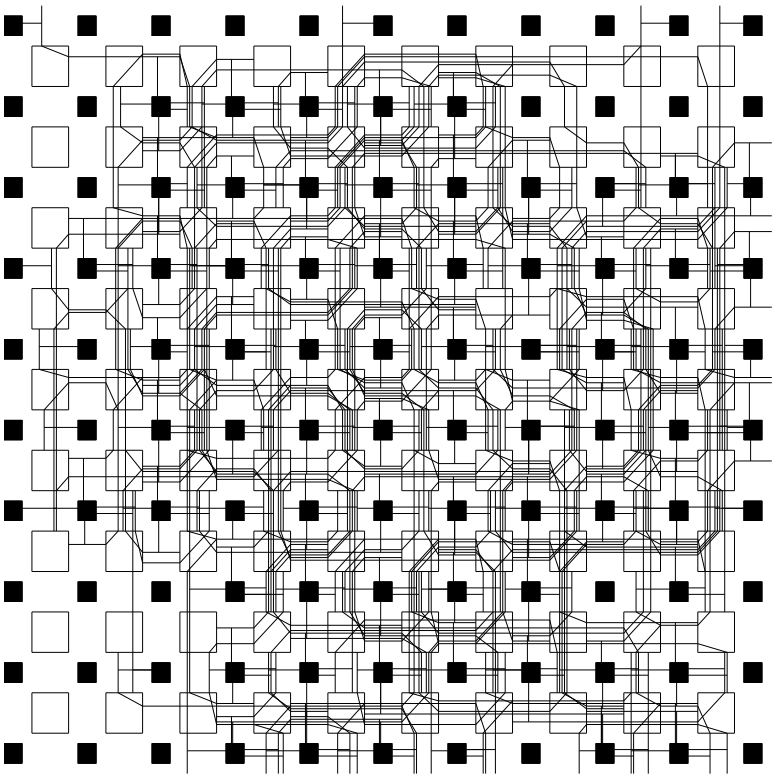


Figure 6.14: The placed and routed 9symml benchmark

channel widths produced by the 2×2 version are larger, the bottom-level routing instances that must be solved during the execution of MONDRIAN are correspondingly larger.

We conjecture that extending MONDRIAN to $k > 3$ will result in even higher-quality solutions. The computation of optimal rectilinear Steiner trees in a $k \times k$ grid is discussed further in Section 6.9.2.

	$k = 3$		$k = 2$	
Name	W	Time	W	Time
busc	9	47.5	10	42.2
dma	10	63.3	12	130.7
bnre	11	142.8	13	249.0
dfsm	11	224.8	12	390.3
z03	13	358.7	15	423.8
9symml	10	36.3	10	16.8
term1	7	32.1	11	11.7
apex7	9	40.2	11	35.8
alu2	11	52.3	13	69.1
too_large	11	55.2	11	54.3
example2	13	55.5	14	121.0
vda	13	67.9	15	132.6
alu4	13	84.5	16	161.7
k2	17	130.3	19	397.3

Table 6.4: Comparison of 2×2 and standard 3×3 decompositions.

6.9 Other Issues

This section addresses a pair of other issues that arise within the thumbnail partitioning framework. The first is the quality of Steiner trees generated by a thumbnail partitioning algorithm. The second is the computation of optimal thumbnails for larger k, which will be used in future work to extend the thumbnail partitioning strategy to larger grids.

6.9.1 Quality of thumbnail-generated Steiner trees

6.9.1.1 Expected quality on random terminal sets

We now show that if the terminals in a net are distributed uniformly at random, then with probability $1 - o(1)$, the ratio of the length of a tree τ produced by thumbnail partitioning

to the length of an optimal tree τ^* is $O(1)$. This result is similar to that of Komlós and Shing [98] for a different recursive partitioning algorithm.

Theorem 6.2 *Let τ be a Steiner tree produced by thumbnail partitioning for any set of terminals, and let τ^* be an optimal Steiner tree for the same set of terminals. If the terminals are distributed uniformly at random, then with probability $1 - o(1)$,*

$$\frac{\|\tau\|}{\|\tau^*\|} = O(1).$$

Proof: Assume without loss of generality that the terminals are contained in the unit square. It is easily shown that for a region whose length on a side is s, the maximum total length of the edges added (not including those added lower in the recursion) is $O(s)$. If the terminals are uniformly distributed, then the depth of the recursion is roughly $\log_{k^2} n$. Thus, the total length of the edges added by the thumbnail partitioning algorithm is given by

$$\begin{aligned} \sum_{i=0}^{\log_{k^2} n} k^i &= O(k^{\log_{k^2} n + 1}) \\ &= O(\sqrt{n}) \end{aligned}$$

Komlós and Shing show that for terminals distributed uniformly at random, $\|\tau^*\| \geq O(\sqrt{n})$ with probability $1 - o(1)$. Thus, with probability $1 - o(1)$, $\|\tau\|/\|\tau^*\| = O(\sqrt{n}/\sqrt{n}) = O(1)$. □

6.9.1.2 Worst-case length

Let $\tilde{\rho}$ denote the maximum ratio of the length of an RST produced by thumbnail partitioning to the length of an optimal RST.

Theorem 6.3 $\tilde{\rho} \geq 3$.

Proof: Suppose that all the logic blocks are in a net, and that in the bottom-level routing step, all 4 corner pins surrounding each logic block are present. The total length of the

edges added for each logic block is 6, and for c logic blocks, the total length of the edges added by a thumbnail partitioning algorithm is $6c$. An optimal routing is a spanning tree of the pins, and has length approximately $2c$. Thus, $\tilde{\rho} \geq 3$. □

We have been unable to derive an upper bound on $\tilde{\rho}$ that is better than $O(\sqrt{n})$ (see the previous section), but we conjecture that in fact, $\tilde{\rho}$ is a constant.

Conjecture 6.1 *$\tilde{\rho}$ is bounded from above by a constant.*

6.9.2 Computing thumbnail rectilinear Steiner trees

One interesting topic for future research is to extend the thumbnail partitioning paradigm to a $k \times k$ grid for $k > 3$. We now consider the general version of the *thumbnail rectilinear Steiner tree (TRST) problem*: compute an optimal RST of a set of terminals drawn from a $k \times k$ grid, for small k.

Researchers have previously investigated Euclidean Steiner trees of terminal sets from small grids [26, 27, 28] and triangular and hexagonal grids [83], but this is the first known study of the rectilinear problem. Note that the Euclidean problem is very different from the rectilinear one. For the Euclidean case, most previous work [26, 27] focuses on simple, mechanical constructions that are conjectured but not proven to be optimal. In the rectilinear case there seem to be no such simple constructions for which optimality can be reasonably conjectured. In particular, the Euclidean constructions largely focus on the complete $k \times k$ terminal set, which is uninteresting in the rectilinear case since a *minimum spanning tree (MST)* is an optimal TRST for such an instance. Cockayne and Hewgill [28] use their exact algorithm for the general Euclidean Steiner tree problem to compute optimal Euclidean Steiner trees for a number of terminal sets on a small integer grid. They point out their algorithm performs far less efficiently on such instances than on instances from large grids, due to the large number of degeneracies—a phenomenon that is likely to occur in the rectilinear case as well.

Our goal in this section is to exploit the special nature of the inputs to devise algorithms that are more efficient than simply applying existing algorithms to these terminal sets.

The reader should note that the conventional measures of time complexity as a function of input size are not particularly meaningful here; since no input size exceeds k^2, by the standard measures any reasonable algorithm runs in constant time for every fixed k. Since k^2 is an upper bound on both the number of terminals and the number of candidate Steiner points, our measures of time complexity are functions of k, in much the same way that the exact time complexity of linear-time sorting algorithms such as radix sort depends on the size of the numbers to be sorted.

We now present some preliminary results on the TRST problem. We devise a full-set decomposition algorithm for computing optimal TRSTs, which uses the special nature of the inputs to improve the screening of candidate full sets. We then present experimental results comparing this algorithm with two existing algorithms for computing optimal rectilinear Steiner trees: Hakimi's spanning tree enumeration algorithm [70] and the dynamic programming algorithm of Aho, Garey, and Hwang [2] for terminals that lie on a small number of parallel lines. As before, the reader should note that all results presented here are easily generalized to rectangular, rather than square, grids.

6.9.2.1 Simple Approaches

VLSI placement algorithms that use TRSTs, such as MONDRIAN, typically do so by precomputing the optimal TRST for every possible terminal set and simply looking them up at runtime. This is obviously quite time-efficient, but requires $O(2^{k^2})$ space, which is prohibitive for k larger than 4. By exploiting the fact that the grid is symmetrical under a 90-degree rotation, one can save a factor of four in the space requirements by sacrificing the $O(1)$ table lookup for an $O(k^2)$ binary search[1]. Even with this reduction, however, storing a TRST for each possible terminal set for $k = 5$ requires approximately 40 megabytes

[1] A practical note: if the terminals are represented as a bit vector and labeled in a spiral fashion, then a 90-degree rotation of a terminal set can be accomplished using $\lfloor k/2 \rfloor$ bit vector operations (shift, and, or).

of storage. While this might be feasible on some machines, for $k = 6$, storage would require roughly 120 gigabytes of space, which is currently impractical.

Another simple approach is to enumerate all TRSTs for a given terminal set, and choose the one with minimum length. The question here is: how many TRSTs can there be for a given terminal set? This question is also interesting since many of the placement algorithms that use TRSTs enumerate all optimal TRSTs.

The number of TRSTs is clearly bounded by the number of MSTs for the complete $k \times k$ grid graph $G_{k\times k}$, since every TRST is a subgraph of some such MST. Since every edge in $G_{k\times k}$ has length 1, every spanning tree is an MST; thus we are interested in the *spanning tree number* of $G_{k\times k}$, denoted $t(G_{k\times k})$. Kreweras [100] proves that the spanning tree number of $G_{k\times k}$ is

$$t(G_{k\times k}) = \prod_{\substack{1\le x<k \\ 1\le y<k}} \left(4\sin^2\frac{x\pi}{2k} + 4\sin^2\frac{y\pi}{2k}\right).$$

The value of this quantity for a few small values of k is shown in Table 6.5. There is no

k	2	3	4	5	6
t	4	192	100352	5.6×10^8	3.3×10^{13}

Table 6.5: Number t of spanning trees in $G_{k\times k}$.

known closed form for this product, but it is easily shown that it grows faster than $O(k^2 2^{k^2})$, and therefore enumerating all spanning trees is slower than the spanning tree enumeration algorithm described in the next section.

6.9.2.2 Spanning Tree Enumeration

Another approach is to apply an existing algorithm for the RST problem to thumbnail-sized inputs. One such algorithm is Hakimi's spanning tree enumeration algorithm for the graph Steiner tree problem [70]. Hakimi's algorithm has time complexity $O(n2^n)$ in a graph with

equally weighted edges, where n is the number of candidate Steiner points. Applied to the grid graph $G_{k\times k}$, an upper bound on the time complexity of Hakimi's algorithm is $O(k^2 2^{k^2})$.

For the standard RST problem, Hakimi's algorithm is less efficient than dynamic programming algorithms such as that of Dreyfus and Wagner [43] and those described in Chapter 3. However, these algorithms examine every subset of the set of terminals, so when applied to thumbnail-sized inputs, they have time complexity $\Omega(2^{k^2})$. The best time complexity among these is our *screened full-set dynamic programming* algorithm (see Chapter 3), whose time complexity is $O(n^2 2.62^n)$, where n is the number of terminals. For thumbnail-sized inputs, this can be as large as $O(k^4 2.62^{k^2})$, which is worse than Hakimi's algorithm.

6.9.2.3 A Dynamic Programming Algorithm

Another natural approach is to apply the Aho, Garey, and Hwang (AGH) algorithm for terminals on a small number of parallel lines [2]. For TRSTs, the terminals lie on a set of at most k parallel lines.

The AGH algorithm proceeds as follows. Imagine sweeping a vertical line from $x = 0$ to $x = k - 1$ through the set of terminals. Let $G_{x\times k}$ denote the portion of the $k \times k$ grid to the left of x. For each x, optimal solutions are built from the optimal solutions in $G_{(x-1)\times k}$. The reader is referred to Aho, Garey, and Hwang [2] for details.

The AGH algorithm has time complexity $O(k16^k)$. This is clearly asymptotically superior to the other two algorithms described above. However, as will be shown empirically below, for the small values of k considered here, the AGH algorithm is slower than the others described earlier. In addition, the AGH algorithm requires $O(k8^k)$ space, and thus it is currently inapplicable to the TRST problem with $k > 7$.

6.9.2.4 Full-Set Decomposition

As in the algorithms of Chapter 3, we use full set screening to produce a set of candidate full sets. The fact that inputs are drawn from a small grid allows us some screening improvements beyond those for the standard RST problem. As in many previous full-set decomposition algorithms [28, 29, 131, 149], we then use a branch and bound algorithm to find a set of candidate full sets of minimum total length, whose union spans all the terminals.

Let $M(T)$ denote an MST of a set T of terminals. A set T of terminals is *compact* if $\|M(T)\| = |T| - 1$ (i.e., if an MST of T contains only unit-length edges).

Theorem 6.4 *There exists an optimal TRST τ in which every compact subset T of the set of terminals induces a connected component in τ.*

Proof: Suppose to the contrary that in every optimal TRST, some compact subset S induces two or more connected components $C_1, \ldots, C_q$. Let τ be an optimal TRST. Since S is compact, there is edge e between two components C_i and C_j that has unit length. Adding e to τ creates a cycle, some of whose edges are outside S, or else C_i and C_j would be connected. Furthermore, at least one edge e' in the cycle must have length at least 2, or else the entire cycle would be within S, and C_i and C_j would be connected. Thus, we can add edge e and delete edge e', creating a TRST that is shorter than τ, contradicting the assumption that τ is optimal. □

Corollary 6.4.1 *If terminals a and b are in a candidate full set, then $\|a - b\| > 1$.* □

These results allow us to eliminate many full sets from candidacy. For every compact subset S of the terminals, every full set containing more than two terminals in S is eliminated from candidacy. The subset S is then added to the set of candidate full sets. While it is not technically a full set, it behaves as one in the decomposition since no other candidate full set intersects it at more than one terminal.

We can also eliminate candidate full sets of cardinality exceeding k.

Theorem 6.5 *For $k \geq 4$, every candidate full set contains at most k terminals.*

Proof: Suppose that S is a full set on $k+1$ or more terminals in a $k \times k$ grid, and assume without loss of generality that the backbone segment of the full tree is horizontal. By the pigeonhole principle, some pair of terminals a and b must share the same x coordinate; assume without loss of generality that $y_a > y_b$. Let c be a terminal nearest to a and b on the x-axis that is not extremal, i.e., that does not have maximum or minimum x value among all terminals. Since $k > 4$, such a terminal must exist. If $y_c < y_a$, then the segment s of the backbone in the range x_a to x_c can be slid so that $y_s = y_c$, giving c degree 2 and contradicting the assumption that S is a full set. Similarly, if $y_c > y_a$, then s can be slid so that $y_s = y_a$, giving a degree 2 and contradicting the assumption that S is a full set. □

In our implementation, we precompute every possible full tree on a $k \times k$ grid. The algorithm then reads in those full sets that are subsets of the input set T of terminals. Since no two terminals in a full set share an x or y coordinate, the total number of full sets is at most $O(k^k)$, and consequently the time complexity of the algorithm is at most $O(2^{k^k})$. Note that $O(k^k) = O(c^{k \log k})$ for constant c, which is an improvement on the best known bound of $O(n1.62^n) = O(k^2 1.62^{k^2})$ on the number of full sets in an unrestricted terminal set (see Section 3.5). Table 6.6 shows the total number of possible full sets of cardinality 3

k	3	4	5	6	7
Full sets	17	196	1258	5368	20157
k^k	27	256	3125	46656	823543

Table 6.6: Number of possible full sets in a $k \times k$ grid.

or greater on a $k \times k$ grid, along with the value of the upper bound k^k, for small values of k. We also apply a number of other tests from the Salowe-Warme algorithm [131] to eliminate full sets from candidacy; these tests eliminate many more full sets, accounting for the gap between k^k and the actual number of possible full sets.

If an edge in a full tree of a full set S intersects a terminal in $T - S$, then that terminal would have degree greater than 1, and thus the full set is eliminated from candidacy. Thus, the number of full sets that must be considered for a given instance is typically far smaller than the total number of possible full sets. Empirical results on the total number of candidate full sets are given below.

6.9.2.5 Experimental Results

We have implemented the spanning tree enumeration, dynamic programming, and full set decomposition algorithms in order to compare them experimentally. The results of these experiments are shown in Table 6.7. For $k \leq 4$, the values shown result from one run on each of the $\binom{k^2}{n}$ possible terminal sets. For $k \geq 5$, the values shown result from one run on each of 1000 different randomly generated terminal sets. There are a number of noteworthy features of these results.

First, note that the running time of the dynamic programming algorithm is essentially the same regardless of n. This is not surprising, as the algorithm performs essentially the same computations regardless of the number and positions of the terminals. In spite of its asymptotic superiority, the dynamic programming algorithm is too slow to be used on instances with $k \leq 7$. (Also, recall that the space requirements of the algorithm prohibit its use on instances with $k > 7$.)

Second, note that the spanning tree enumeration and full-set decomposition algorithms are quite fast on very sparse or very dense instances. This is not surprising: for sparse instances, there are few terminals, and for dense instances, there are few candidate Steiner points (for Hakimi's algorithm) or candidate full sets (for the FSD algorithm).

For all but the smallest instances, the FSD algorithm is faster than both the STE and DP algorithms. In addition, the current branch and bound algorithm for finding the optimal full-set decomposition it quite simple. We believe that its efficiency can be considerably improved by using more elaborate strategies to prune the search space [131].

k	n	STE	DP	FSD	%MST	f
3	3	0.001	0.627	0.011	81.0	3
3	6	0.001	0.629	0.013	95.2	4
4	4	0.002	0.682	0.013	66.0	8
4	8	0.009	0.823	0.020	72.2	37
4	12	0.002	0.695	0.014	98.2	5
5	5	0.020	2.020	0.020	47.7	14
5	10	0.737	2.010	0.055	51.3	52
5	15	0.120	2.007	0.051	72.0	41
5	20	0.004	2.030	0.023	99.0	8
6	6	0.450	25.33	0.030	34.1	28
6	12	152.5	25.97	1.639	31.9	91
6	18	23.01	26.19	9.616	44.8	78
6	24	1.048	25.91	0.213	77.7	35
6	30	0.007	26.67	0.040	99.2	7
7	7	13.55	830.7	0.079	21.0	35
7	14	18632	889.8	55.69	17.3	173
7	21	15867	862.6	205.2	29.1	144
7	28	455.0	901.0	74.69	46.6	74
7	35	6.317	907.8	0.223	84.1	32
7	42	0.014	906.6	0.109	99.5	8

Table 6.7: Average runtimes (in seconds) for the spanning tree enumeration (STE), dynamic programming (DP), and full-set decomposition (FSD) algorithms. %MST indicates the percentage of the instances for which the TRST has the same length as the MST. The value of f is the maximum number of candidate full sets.

Lastly, note that often an MST is an optimal TRST. Clearly a TRST cannot have length less than $n-1$, so the algorithms first compute an MST, and if it has length $n-1$, then it is returned as the optimal TRST. However, in many cases the MST has length exceeding $n-1$, but is nonetheless an optimal TRST.

Figure 6.15 illustrates an optimal TRST on 14 terminals in a 7×7 grid, computed by the FSD algorithm.

gure 6.15: An optimal TRST on 14 terminals in a 7×7 grid.

nclusions and Future Work

bed a system called MONDRIAN that performs simultaneous placement and in FPGAs. MONDRIAN compares favorably to several previous tools, pro- results for a number of industrial benchmarks. In addition, MONDRIAN can performance-driven placement and global routing, and the resulting layouts ed source-sink path lengths with comparable channel widths and a modest

in practice, a geometric algorithm tailored to the thumbnail partitioning problem might produce comparable results more quickly than the simulated annealing algorithm.

6.10.2 Virtual terminal assignment

We intend to try replacing the minimum-cost perfect matching algorithm used for virtual terminal assignment (see Section 6.4.3), which computes optimal matching solutions in $O(r^{5/2})$ time, with a heuristic algorithm. A greedy matching can be computed in $O(rs)$ time, where r is the number of nets and s is the number of switch blocks on a cut line. If the greedy algorithm produces solutions comparable to those using the exact algorithm, then a substantial savings in running time may be realized.

6.10.3 Bottom-level routing

Another modification that may improve the quality of solutions computed by MONDRIAN is to extend the algorithm at the basis of the recursion to handle regions containing more than one logic block. The restricted nature of the current bottom-level routing problem makes it possible to efficiently compute high-quality bottom-level routing solutions, but with some effort, comparable algorithms might be devised for larger numbers of logic blocks.

A place for everything, and everything in its place.
— Samuel Smiles, *Thrift*

7

Conclusion

In this dissertation, we have examined a number of geometric interconnection, partitioning, and placement problems that arise in the field of electronic physical design automation.

We have presented two new dynamic programming algorithms for computing optimal rectilinear Steiner trees (RSTs). The first algorithm, called Full-set Dynamic Programming (FDP), has time complexity $O(n3^n)$. The FDP algorithm is very simple and easily implemented. The second algorithm, called Screened Full-set Dynamic Programming (SFDP) uses full-set screening to improve the time complexity and the running time in practice of the FDP algorithm. These algorithms improve upon the time complexity of previous popular algorithms for computing optimal RSTs, and we have presented empirical evidence that they are also superior in terms of running time in practice.

We have examined the problem of multi-terminal routing in the presence of obstacles, which we formulate as the obstacle-avoiding rectilinear Steiner tree (OARST) problem. We proved a theorem, analogous to Hanan's theorem [71] for the standard RST problem, that enables the first algorithms that compute optimal OARSTs in time corresponding to the size of the instance rather than the size of the routing area. We have also shown that this theorem can be used to devise effective heuristics for computing good OARST solutions. We have also examined two important special cases of the OARST problem, showing that

the problem remains NP-complete if all terminals lie on obstacle perimeters but is solvable in polynomial time if all terminals lie on the perimeter of the routing region.

We have introduced the power-p Steiner tree problem, which is to find a geometric Steiner tree that minimizes the sum of the individual edge lengths each raised to the p power. The power-p Steiner tree problem finds applications in VLSI routing as well as facility location and other network design applications. We also examine a special case of the power-p Steiner tree problem called the bottleneck Steiner tree problem, which is to find a geometric Steiner tree in which the length of the longest edge is minimized. We have given algorithms for computing optimal Euclidean power-2 Steiner trees, optimal rectilinear bottleneck Steiner trees, and rectilinear Steiner trees that minimize a combination of bottleneck weight and total length. We have given evidence that power-p Steiner trees cannot be exactly computed for $p \geq 5$. In addition, we have given bounds on the power-p Steiner ratio, which measures the quality of a minimum spanning tree as an approximation of a power-p Steiner tree. We have also derived the exact value of the bottleneck Steiner ratio and given a fully polynomial-time approximation scheme for computing bottleneck Steiner trees.

Finally, we have described a system called MONDRIAN for performing simultaneous placement and global routing in field-programmable gate arrays (FPGAs). MONDRIAN is a recursive partitioning strategy that uses optimal rectilinear Steiner trees for sets of terminals from a small grid. We have presented experimental results indicating that MONDRIAN produces results superior to those of previous FPGA layout tools. In addition, we describe a modified version of MONDRIAN that computes performance-driven placement and global routing solutions. The development of MONDRIAN includes several results of independent interest, such as minimum-congestion routing in a cycle and computing optimal rectilinear Steiner trees for terminals in a small grid.

Bibliography

[1] E. H. L. Aarts and P. J. M. van Laarhoven. A new polynomial-time cooling schedule. In *Proceedings of the International Conference on Computer-Aided Design*, pages 206–208, 1985.

[2] A. V. Aho, M. R. Garey, and F. K. Hwang. Rectilinear Steiner trees: Efficient special-case algorithms. *Networks*, 7:37–58, 1977.

[3] M. J. Alexander, J. P. Cohoon, J. L. Ganley, and G. Robins. An architecture-independent approach to FPGA routing based on multi-weighted graphs. In *Proceedings of the European Design Automation Conference*, pages 259–264, 1994.

[4] M. J. Alexander, J. P. Cohoon, J. L. Ganley, and G. Robins. Performance-oriented placement and routing for field-programmable gate arrays. In *Proceedings of the European Design Automation Conference*, 1995. (To appear).

[5] M. J. Alexander and G. Robins. A new approach to FPGA routing based on multi-weighted graphs. In *Proceedings of the International Workshop on Field-Programmable Gate Arrays*, 1994.

[6] T. Asano. Generalized Manhattan path algorithm with applications. *IEEE Transactions on Computer-Aided Design*, 7:797–804, 1988.

[7] H. Bakoglu. *Circuits, Interconnections, and Packaging for VLSI.* Addison-Wesley, Reading, Massachusetts, 1990.

[8] S. Bapat and J. P. Cohoon. A parallel VLSI circuit layout methodology. In *Proceedings of the Sixth International Conference on VLSI Design*, pages 236–241, 1993.

[9] S. Bapat, J. P. Cohoon, P. L. Heck, A. Ju, and L. J. Randall. Examining routing solutions. In *Proceedings of the Second Great Lakes Symposium on VLSI*, February 1992.

[10] J. E. Beasley. A heuristic for Euclidean and rectilinear Steiner problems. *European Journal of Operational Research*, 58:284–292, 1992.

[11] M. R. C. M. Berkelaar. LP_SOLVE user's manual (version 1.5). 1994.

[12] P. Berman and V. Ramaiyer. Improved approximations for the Steiner tree problem. *Journal of Algorithms*, 17:381–408, 1994.

[13] M. Bern. Faster exact algorithms for Steiner trees in planar networks. *Networks*, 20:109–120, 1990.

[14] K. D. Boese, A. B. Kahng, and G. Robins. High-performance routing trees with identified critical sinks. In *Proceedings of the Thirtieth Design Automation Conference*, pages 182–187, 1993.

[15] M. A. Breuer. A class of min-cut placement algorithms. In *Proceedings of the Fourteenth Design Automation Conference*, pages 284–290, 1977.

[16] M. A. Breuer. Min-cut placement. *Journal of Design Automation and Fault-Tolerant Computing*, 1:343–362, 1977.

[17] S. D. Brown, R. J. Francis, J. Rose, and Z. G. Vranesic. *Field-Programmable Gate Arrays*. Kluwer Academic Publishers, Boston, Massachusetts, 1992.

[18] S. D. Brown, J. S. Rose, and Z. G. Vranesic. A detailed router for field-programmable gate arrays. In *Proceedings of the International Conference on Computer-Aided Design*, pages 382–385, 1990.

[19] J. Burgess. Completing the circuit of science and art. *The Washington Post*, pages H1,H6–H7, October 4, 1992.

[20] T. Chao and Y. Hsu. Rectilinear Steiner tree construction by local and global refinement. In *Proceedings of the International Conference on Computer-Aided Design*, pages 432–435, 1990.

[21] D. S. Chen. Constrained wirelength minimization of a Steiner tree. Technical report, Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, Illinois, 1994.

[22] S. Cheng, A. Lim, and C. Wu. Optimal rectilinear Steiner tree for extremal point sets. In *Proceedings of the International Symposium on Algorithms and Computation*, volume 762 of *Lecture Notes in Computer Science*, pages 523–532. Springer-Verlag, Berlin, Germany, 1993.

[23] C. Chiang, M. Sarrafzadeh, and C. K. Wong. Global routing based on Steiner min-max trees. *IEEE Transactions on Computer-Aided Design*, 9:1318–1325, 1990.

[24] C. Chiang, M. Sarrafzadeh, and C. K. Wong. An optimal algorithm for rectilinear Steiner trees for channels with obstacles. *International Journal of Circuit Theory and Applications*, 19:551–563, 1991.

[25] C. Chiang, M. Sarrafzadeh, and C. K. Wong. An algorithm for exact rectilinear Steiner trees for switchbox with obstacles. *IEEE Transactions on Circuits and Systems*, 39:446–455, 1992.

[26] F. R. K. Chung, M. Gardner, and R. L. Graham. Steiner trees on a checkerboard. *Mathematics Magazine*, 62:83–96, 1989.

[27] F. R. K. Chung and R. L. Graham. Steiner trees for ladders. *Annals of Discrete Mathematics*, 2:173–200, 1978.

[28] E. J. Cockayne and D. E. Hewgill. Exact computation of Steiner minimal trees in the plane. *Information Processing Letters*, 22:151–156, 1986.

[29] E. J. Cockayne and D. E. Hewgill. Improved computation of plane Steiner minimal trees. *Algorithmica*, 7:219–229, 1992.

[30] J. P. Cohoon and J. L. Ganley. Rectilinear interconnections in the presence of obstacles. In Y. T. Wong and M. Pecht, editors, *Advanced Routing in Electronic Modules*. CRC Press, Boca Raton, Florida. (To appear).

[31] J. P. Cohoon and D. S. Richards. Optimal two-terminal α-β wire routing. *Integration: the VLSI Journal*, 6:35–57, 1988.

[32] J. P. Cohoon, D. S. Richards, and J. S. Salowe. An optimal Steiner tree algorithm for a net whose terminals lie on the perimeter of a rectangle. *IEEE Transactions on Computer-Aided Design*, 9:398–407, 1990.

[33] J. Cong, A. Kahng, G. Robins, M. Sarrafzadeh, and C. K. Wong. Probably good performance-driven global routing. *IEEE Transactions on Computer-Aided Design*, 11:739–752, 1992.

[34] J. Cong, K. S. Leung, and D. Zhou. Performance-driven interconnect design based on distributed RC delay model. In *Proceedings of the Thirtieth Design Automation Conference*, pages 606–611, 1993.

[35] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, Massachusetts, 1990.

[36] D. M. Cvetković, M. Doob, and H. Sachs. *Spectra of Graphs: Theory and Applications*. Academic Press, New York, New York, 1979.

[37] P. J. de Rezende, D. T. Lee, and Y. F. Wu. Rectilinear shortest paths with rectangular barriers. In *Proceedings of the Seventeenth Symposium on Computational Geometry*, pages 204–213, 1985.

[38] P. M. Dearing and R. L. Francis. A network flow solution to a multifacility location problem involving rectilinear distances. *Transportation Science*, 8:126–141, 1974.

[39] W. A. Dees and P. G. Karger. Automated rip-up and reroute techniques. In *Proceedings of the Nineteenth Design Automation Conference*, pages 432–439, 1982.

[40] W. A. Dees and R. J. Smith II. Performance of interconnection rip-up and reroute strategies. In *Proceedings of the Eighteenth Design Automation Conference*, pages 382–390, 1981.

[41] L. L. Deneen and J. B. Dezell. Using partitioning and clustering techniques to generate rectilinear Steiner trees. In *Proceedings of the Second Canadian Conference on Computational Geometry*, pages 315–318, 1990.

[42] L. L. Deneen, G. M. Shute, and C. D. Thomborson. A probably fast, provably optimal algorithm for rectilinear Steiner trees. *Random Structures and Algorithms*, 5:535–557, 1994.

[43] S. E. Dreyfus and R. A. Wagner. The Steiner problem in graphs. *Networks*, 1:195–207, 1972.

[44] Z. Drezner and G. O. Wesolowsky. Layout of facilities with some fixed points. *Computers and Operations Research*, 12:603–610, 1985.

[45] D. Z. Du and F. K. Hwang. A proof of the Gilbert-Pollak conjecture on the Steiner ratio. *Algorithmica*, 7:121–135, 1992.

[46] A. E. Dunlop and B. W. Kernighan. A procedure for placement of standard-cell VLSI circuits. *IEEE Transactions on Computer-Aided Design*, 4:92–98, 1985.

[47] H. Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer-Verlag, Berlin, Germany, 1987.

[48] A. El Gamal, J. Greene, J. Reyneri, E. Rogoyski, K. El-Ayat, and A. Mohsen. An architecture for electrically configurable gate arrays. *IEEE Journal of Solid State Circuits*, 24:394–398, 1988.

[49] W. C. Elmore. The transient response of damped linear network with particular regard to wideband amplifiers. *Journal of Applied Physics*, 19:55–63, 1948.

[50] J. Elzinga, D. Hearn, and W. D. Randolph. Minimax multifacility location with Euclidean distances. *Transportation Science*, 10:321–336, 1976.

[51] R. E. Erickson, C. L. Monma, and A. F. Veinott Jr. Send-and-split method for minimum-concave-cost network flows. *Mathematics of Operations Research*, 12:634–664, 1987.

[52] J. W. Eyster and J. A. White. Some properties of the squared Euclidean distance location problem. *AIIE Transactions*, 5:275–280, 1973.

[53] A. Frank, T. Nishizeki, N. Saito, H. Suzuki, and É. Tardos. Algorithms for routing around a rectangle. *Discrete Applied Mathematics*, 40:363–378, 1992.

[54] J. L. Ganley and J. P. Cohoon. A faster dynamic programming algorithm for exact rectilinear Steiner minimal trees. In *Proceedings of the Fourth Great Lakes Symposium on VLSI*, pages 238–241, 1994.

[55] J. L. Ganley and J. P. Cohoon. FPGA layout by congestion-driven simultaneous placement and routing. Technical Report CS–94–47, Department of Computer Science, University of Virginia, Charlottesville, Virginia, 1994.

[56] J. L. Ganley and J. P. Cohoon. Optimal rectilinear Steiner minimal trees in $O(n^2 2.62^n)$ time. In *Proceedings of the Sixth Canadian Conference on Computational Geometry*, pages 308–313, 1994.

[57] J. L. Ganley and J. P. Cohoon. Routing a multi-terminal critical net: Steiner tree construction in the presence of obstacles. In *Proceedings of the International Symposium on Circuits and Systems*, pages 113–116, 1994.

[58] J. L. Ganley and J. P. Cohoon. Minimum-congestion hypergraph embedding in a cycle. Technical Report CS–95–04, Department of Computer Science, University of Virginia, Charlottesville, Virginia, 1995.

[59] J. L. Ganley and J. P. Cohoon. Provably good moat routing. Technical Report CS–95–13, Department of Computer Science, University of Virginia, Charlottesville, Virginia, 1995.

[60] J. L. Ganley and J. P. Cohoon. Thumbnail rectilinear Steiner trees. In *Proceedings of the Fifth Great Lakes Symposium on VLSI*, pages 46–49, 1995.

[61] J. L. Ganley and J. S. Salowe. Optimal and approximate bottleneck Steiner trees. manuscript, 1994.

[62] M. R. Garey, R. L. Graham, and D. S. Johnson. The complexity of computing Steiner minimal trees. *SIAM Journal on Applied Mathematics*, 32:835–859, 1977.

[63] M. R. Garey and D. S. Johnson. The rectilinear Steiner tree problem is NP-complete. *SIAM Journal on Applied Mathematics*, 32:826–834, 1977.

[64] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman and Company, New York, New York, 1979.

[65] E. N. Gilbert and H. O. Pollak. Steiner minimal trees. *SIAM Journal on Applied Mathematics*, 16:1–29, 1966.

[66] C. D. Godsil. Spectra of trees. In M. Rosenfeld and J. Zaks, editors, *Convexity and Graph Theory*, volume 20 of *Annals of Discrete Mathematics*, pages 151–159. North-Holland, Amsterdam, Netherlands, 1983.

[67] G. Golub and J. M. Ortega. *Scientific Computing: An Introduction with Parallel Computing*. Academic Press, San Diego, California, 1993.

[68] T. Günther. Die räumliche Anordnung von Einheiten mit Wechselbeziehungen. *Elektronische Datenverarbeitung*, 11:209–211, 1969.

[69] U. I. Gupta, D. T. Lee, and J. Y.-T. Leung. Efficient algorithms for interval graphs and circular arc graphs. *Networks*, 12:459–467, 1982.

[70] S. L. Hakimi. Steiner's problem in graphs and its implications. *Networks*, 1:113–133, 1971.

[71] M. Hanan. On Steiner's problem with rectilinear distance. *SIAM Journal on Applied Mathematics*, 14:255–265, 1966.

[72] N. Hasan, G. Vijayan, and C. K. Wong. A neighborhood improvement algorithm for rectilinear Steiner trees. In *Proceedings of the International Symposium on Circuits and Systems*, pages 2869–2872, 1990.

[73] D. W. Hightower. A solution to the line-routing problem on the continuous plane. In *Proceedings of the Sixth Design Automation Workshop*, pages 1–24, 1969.

[74] D. W. Hightower. The interconnection problem—a tutorial. In *Proceedings of the Tenth Design Automation Workshop*, pages 1–21, 1973.

[75] J. Ho, G. Vijayan, and C. K. Wong. New algorithms for the rectilinear Steiner tree problem. *IEEE Transactions on Computer-Aided Design*, 9:185–193, 1990.

[76] N. D. Holmes, N. A. Sherwani, and M. Sarrafzadeh. Utilization of vacant terminals for improved over-the-cell channel routing. *IEEE Transactions on Computer-Aided Design*, 12:780–792, 1993.

[77] X. Hong, T. Xue, E. S. Kuh, C.-K. Cheng, and J. Huang. Performance-driven Steiner tree algorithms for global routing. In *Proceedings of the Thirtieth Design Automation Conference*, pages 177–181, 1993.

[78] J. N. Hooker. Solving nonlinear multiple-facility network location problems. *Networks*, 19:117–133, 1989.

[79] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2:225–231, 1973.

[80] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, Massachusetts, 1979.

[81] T. W. Hungerford. *Algebra.* Springer-Verlag, New York, New York, 1974.

[82] F. K. Hwang. On Steiner minimal trees with rectilinear distance. *SIAM Journal on Applied Mathematics*, 30:104–114, 1976.

[83] F. K. Hwang and D. Z. Du. Steiner minimal trees on the Chinese checkerboard. *Mathematics Magazine*, 64:332–339, 1991.

[84] F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner Tree Problem*, volume 53 of *Annals of Discrete Mathematics.* North-Holland, Amsterdam, Netherlands, 1992.

[85] R. Z. Hwang, R. C. Chang, and R. C. T. Lee. The searching over separators strategy to solve some NP-hard problems in subexponential time. *Algorithmica*, 9:398–423, 1993.

[86] T. Ichimori. A shortest path approach to a multifacility minimax location problem with rectilinear distances. *Journal of the Operations Research Society of Japan*, 28:269–284, 1985.

[87] J. P. Ignizio and T. M. Cavalier. *Linear Programming*. Prentice Hall, Englewood Cliffs, New Jersey, 1994.

[88] J. JáJá and S. A. Wu. On routing two-terminal nets in the presence of obstacles. *IEEE Transactions on Computers*, 8:563–570, 1989.

[89] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst-cast performance bounds for simple one-dimensional bin packing algorithms. *SIAM Journal on Computing*, 3:299–325, 1974.

[90] A. B. Kahng and G. Robins. A new class of iterative Steiner tree heuristics with good performance. *IEEE Transactions on Computer-Aided Design*, 11:893–902, 1992.

[91] A. B. Kahng and G. Robins. *On Optimal Interconnections for VLSI*. Kluwer Academic Publishers, Norwell, Massachusetts, 1995.

[92] K. Kanchanasut. A shortest-path algorithm for Manhattan graphs. *Information Processing Letters*, 49:21–25, 1994.

[93] R. M. Karp. On the computational complexity of combinatorial problems. *Networks*, 5:45–68, 1975.

[94] M. Kaufmann, S. Gao, and K. Thulasiraman. On Steiner minimal trees in grid graphs and its application to VLSI layout. In *Proceedings of the International Symposium on Algorithms and Computation*, volume 834 of *Lecture Notes in Computer Science*, pages 351–359. Springer-Verlag, Berlin, Germany, 1994.

[95] L. G. Khachian. A polynomial algorithm in linear programming. *Soviet Mathematics Doklady*, 20:191–194, 1979.

[96] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–679, 1983.

[97] J. M. Kleinhans, G. Sigl, F. M. Johannes, and K. J. Antreich. GORDIAN: VLSI placement by quadratic programming and slicing optimization. *IEEE Transactions on Computer-Aided Design*, 10:356–365, 1991.

[98] J. Komlós and M. T. Shing. Probabilistic partitioning algorithms for the rectilinear Steiner problem. *Networks*, 15:413–423, 1985.

[99] M. R. Kramer and J. van Leeuwen. Wire-routing is NP-complete. Technical Report RUU-CS-82-4, Department of Computer Science, University of Utrecht, Utrecht, Netherlands, 1982.

[100] G. Kreweras. Complexité et circuits Eulériens dan les sommes tensorielles de graphes. *Journal of Combinatorial Theory, Series B*, 24:202–212, 1978.

[101] J. B. Kruskal. On the shortest spanning tree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7:48–50, 1956.

[102] U. Lauther. A min-cut placement algorithm for general cell assemblies based on a graph representation. *Journal of Digital Systems*, 4:21–34, 1979.

[103] C. Y. Lee. An algorithm for path connections and its applications. *IRE Transactions on Electronic Computers*, 10:346–365, 1961.

[104] D. T. Lee, C. D. Yang, and C. K. Wong. Rectilinear paths among rectilinear obstacles. Technical Report 92-AC-123, Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, Illinois, September 1992.

[105] D. T. Lee, C. D. Yang, and C. K. Wong. On bends and distances of paths among obstacles in 2-layer interconnection model. *IEEE Transactions on Computers*, 43:711–724, 1994.

[106] J. H. Lee, N. K. Bose, and F. K. Hwang. Use of Steiner's problem in suboptimal routing in rectilinear metric. *IEEE Transactions on Circuits and Systems*, pages 470–476, 1976.

[107] G. G. Lemieux and S. D. Brown. A detailed routing algorithm for allocating wire segments in field-programmable gate arrays. In *Proceedings of the Fourth Physical Design Workshop*, pages 215–226, 1993.

[108] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout*. John Wiley and Sons, Chichester, England, 1990.

[109] F. D. Lewis, W. C. Pong, and N. Van Cleave. Optimum Steiner tree generation. In *Proceedings of the Second Great Lakes Symposium on VLSI*, pages 207–212, 1992.

[110] D. Lichtenstein. Planar formulae and their uses. *SIAM Journal on Computing*, 11:329–343, 1982.

[111] W. W.-L. Lin. Wire length minimization in a simple single-layer circuit. Bachelor's thesis, Department of Electrical Engineering, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1983.

[112] W. Lipski Jr. An $O(n \log n)$ Manhattan path algorithm. *Information Processing Letters*, 19:99–102, 1984.

[113] R. F. Love, G. O. Wesolowsky, and S. A. Kraemer. A multifacility minimax location method for Euclidean distances. *International Journal of Production Research*, 11:37–45, 1973.

[114] S. Mayrhofer and U. Lauther. Congestion-driven placement using a new multi-partitioning heuristic. In *Proceedings of the International Conference on Computer-Aided Design*, pages 332–335, 1990.

[115] N. Megiddo. Combinatorial optimization with rational objective functions. *Mathematics of Operations Research*, 4:414–424, 1979.

[116] K. Mikami and K. Tabuchi. A computer program for optimal routing of printed circuit connectors. *IFIPS Proceedings*, H47:1475–1478, 1968.

[117] J. Milner. *Mondrian*. Abbeville, New York, New York, 1992.

[118] S. Mirayala, J. Hashmi, and N. Sherwani. Switchbox Steiner tree problem in presence of obstacles. In *Proceedings of the International Conference on Computer-Aided Design*, pages 536–539, 1991.

[119] J. S. B. Mitchell. L_1 shortest paths among polygonal obstacles in the plane. *Algorithmica*, 8:55–88, 1992.

[120] E. F. Moore. Shortest path through a maze. *Annals of the Computational Laboratory of Harvard University*, 30:285–292, 1959.

[121] H. Okamura and P. D. Seymour. Multicommodity flows in planar graphs. *Journal of Combinatorial Theory, Series B*, 31:75–81, 1981.

[122] B. T. Preas. Benchmarks for cell-based layout systems. In *Proceedings of the Twenty-fourth Design Automation Conference*, pages 319–320, 1987.

[123] B. T. Preas and M. J. Lorenzetti, editors. *Physical Design Automation of VLSI Systems*. Benjamin/Cumming Publishing Company, Menlo Park, California, 1988.

[124] R. C. Prim. Shortest connection networks and some generalizations. *Bell System Technical Journal*, 36:1389–1401, 1957.

[125] J. S. Provan. A polynomial algorithm for the Steiner tree problem on terminal-planar graphs. Technical Report 83/10, Department of Operations Research, University of North Carolina, Chapel Hill, North Carolina, 1983.

[126] R. Raghavan, J. P. Cohoon, and S. Sahni. Single bend wiring. *Journal of Algorithms*, 7:232–257, 1986.

[127] S. K. Rao, P. Sadayappan, F. K. Hwang, and P. W. Shor. The rectilinear Steiner arborescence problem. *Algorithmica*, 7:277–288, 1992.

[128] D. S. Richards. Complexity of single-layer routing. *IEEE Transactions on Computers*, 33:286–288, 1984.

[129] D. S. Richards and J. S. Salowe. A linear-time algorithm to construct a rectilinear Steiner minimal tree for k-extremal point sets. *Algorithmica*, 7:247–276, 1992.

[130] N. Robertson and P. D. Seymour. Graph minors XIII: The disjoint paths problem. *Journal of Combinatorial Theory, Series B*, 63:65–110, 1995.

[131] J. S. Salowe and D. M. Warme. 35-point rectilinear Steiner minimal trees in a day. *Networks*, 25:69–87, 1995.

[132] M. J. Saltzman. Personal communication. 1994.

[133] M. Sarrafzadeh and C. K. Wong. Bottleneck Steiner trees in the plane. *IEEE Transactions on Computers*, 41:370–374, 1992.

[134] M. Schönert et al. GAP: Groups, algorithms, and programming (version 3 release 4). 1994.

[135] N. Sherwani. *Algorithms for VLSI Physical Design Automation.* Kluwer Academic Publishers, Boston, Massachusetts, 1993.

[136] H. Shin and A. Sangiovanni-Vincentelli. A detailed router based on incremental routing modifications: Mighty. *IEEE Transactions on Computer-Aided Design*, 6:942–955, 1987.

[137] A. F. Sidorenko. On minimal rectilinear Steiner trees. *Diskretnaya Matematika*, 1:28–37, 1989. (In Russian).

[138] 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.

[139] W. D. Smith. How to find Steiner minimal trees in Euclidean d-space. *Algorithmica*, 7:137–177, 1992.

[140] J. Soukup. On minimum cost networks with nonlinear costs. *SIAM Journal on Applied Mathematics*, 29:571–581, 1975.

[141] J. Soukup. Circuit layout. *Proceedings of the IEEE*, 69:1281–1304, 1981.

[142] H. Spruth, F. Johannes, and K. Antreich. PHIroute: A parallel hierarchical sea-of-gates router. In *Proceedings of the International Symposium on Circuits and Systems*, pages 487–490, 1994.

[143] P. R. Suaris and G. Kedem. An efficient algorithm for quadrisection and its applications to standard cell placement. *IEEE Transactions on Circuits and Systems*, 35:294–303, 1988.

[144] P. R. Suaris and G. Kedem. A quadrisection-based place and route scheme for standard cells. *IEEE Transactions on Computer-Aided Design*, 8:234–244, 1989.

[145] C. D. Thomborson, B. Alpern, and L. Carter. Rectilinear Steiner tree minimization on a workstation. In N. Dean and G. E. Shannon, editors, *Computational Support for Discrete Mathematics*, volume 15 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 119–136. American Mathematical Society, Providence, Rhode Island, 1994.

[146] B. L. van der Waerden. *Modern Algebra*. Frederick Ungar Publishing Company, New York, New York, 1949.

[147] J. A. White. A quadratic facility location problem. *AIIE Transactions*, 3:156–157, 1971.

[148] P. Widmayer. On graphs preserving rectilinear shortest paths in the presence of obstacles. *Annals of Operations Research*, 33:557–575, 1991.

[149] P. Winter. An algorithm for the Steiner problem in the Euclidean plane. *Networks*, 15:323–345, 1985.

[150] P. Winter. Personal communication. 1994.

[151] P. Winter. Reductions for the rectilinear Steiner tree problem. manuscript, 1994.

[152] S. Wolfram. *Mathematica: A System for Doing Mathematics by Computer.* Addison-Wesley, Redwood City, California, 1988.

[153] Y. T. Wong. Personal communication. 1993.

[154] Y. T. Wong and M. Pecht. A solution for Steiner's problem. In M. Pecht, editor, *Placement and Routing of Electronic Modules*, pages 261–304. Marcel Dekker, New York, New York, 1993.

[155] Y. Wu, P. Widmayer, M. D. F. Schlag, and C. K. Wong. Rectilinear shortest paths and minimum spanning trees in the presence of rectilinear obstacles. *IEEE Transactions on Computers*, 36:321–331, 1987.

[156] Y. L. Wu and M. Marek-Sadowska. An efficient router for 2-D field programmable gate arrays. In *Proceedings of the European Design and Test Conference*, pages 412–416, 1994.

[157] Xilinx. *The Programmable Gate Array Data Book.* Xilinx, San Jose, California, 1992.

[158] Y. Y. Yang and O. Wing. Optimal and suboptimal solution algorithms for the wiring problem. In *Proceedings of the International Symposium on Circuit Theory*, pages 154–158, 1972.

[159] A. Z. Zelikovsky. An 11/6-approximation algorithm for the network Steiner problem. *Algorithmica*, 9:463–470, 1993.

Vita

Joseph Lavinus Ganley was born in 1968 in Arlington, Virginia. He received the Bachelor of Science in Computer Science from Virginia Tech in 1990, and the Master of Science in Computer Science from Virginia Tech in 1992.