Approvals

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

Doctor of Philosophy

Computer Science

David M. Warme

David M. Warme

Approved:

Jeffrey S. Salowe (Advisor)

James P. Cohoon (Chair)

John L. Pfaltz

Kevin J. Sullivan

Peter A. Beling (Minor Representative)

Accepted by the School of Engineering and Applied Science:

R. W. Miksad

Richard W. Miksad (Dean)

May 1998

Spanning Trees in Hypergraphs with Applications to Steiner Trees

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

David Michael Warme

May 1998

This work is dedicated to the incomparable three-in-one:

- To God the Father almighty, author of all that is created and creative.

- To my Lord and saviour Jesus Christ, who out of love suffered, died and rose again to pay for my wickedness so that I could be with Him forever.

- To the Holy Spirit, who inspires, encourages and convicts me to be more like Jesus.

All glory, honor and praise to God most high!

Acknowledgments

My greatest thanks and appreciation go to my wife Rachel. Without her support, understanding, kindness and encouragement this dissertation would not exist. The writing may be mine, but the triumph belongs to her — nobody else has worked as hard for this degree. It is all too easy for me to believe that Proverbs chapter 31 is really a prophecy about Rachel. Thanks also to Michael and Sina for letting me be your daddy. You are the most special boy and most special girl I know, and I'll be applying for the full-time position very soon!

I sincerely thank my advisor, Jeff Salowe. His guidance, advice, and encouragement have been invaluable throughout both my master's and doctoral studies. I am especially grateful for the day back in 1994 when he tossed a really fun problem into my lap. Little did I know that his chunk of code would bring me so much fun, hard work, discovery, late nights, disappointment, exhilaration, sweat — and a Ph.D. dissertation!

At least a thousand thanks must go to Karla Hoffman, without whom I'd still be struggling to solve 100 point problems. Her two excellent classes on integer programming properly equipped me to make real progress on this problem. Our many after-class discussions were extremely helpful in crystallizing my integer programming formulation. Thank you especially, Karla, for not permitting me to do the normal class project, but for forcing me instead to apply the techniques to my research topic!

A special thank you to Abilio Lucena for agreeing to collaborate with me on my project for Karla's class. Our joint work in April of 1996 yielded the first optimal solutions ever to several of the 100 point OR-library problems. These encouraging results inspired me to apply integer programming techniques directly to the FST concatenation problem. Thanks also to Yash Aneja who kindly referred me to Abilio.

I thank Pawel Winter for the absolutely splendid timing of his recent Euclidean FST generator (I was literally two days away from starting the six month task of writing my own) and for agreeing to collaborate. He gave me a number of suggestions whose impact was always inversely proportional to their length in words. Many heartfelt thanks also to his student, Martin Zachariasen, who worked so closely with me on this collaboration. His hard work, keen insight, and lengthy discussions have contributed immensely to this work. He has become a good friend.

I gratefully thank Maurice Queyranne for solving my separation problem so quickly, and for other thought-provoking conversations. I thank him also for encouraging me to enter the George Nicholson student paper competition.

Thanks also to Jim Cohoon, John Pfaltz, Kevin Sullivan and Peter Beling for serving on my dissertation committee and for their helpful comments on this work.

Many thanks to James Davis for his gracious listening, tracking of progress, encouragement — and for a number of fruitful code reviews. I thank him also for his friendship and for the many social engagements that made me put away the research when distraction was what I needed most.

In addition, I am grateful to the following people for helpful comments on various bits and pieces of this research: Robert Carr, William Cunningham, Joe Ganley, Andrew Kahng, Michael Kaufmann, Alexander Martin, Thomas McCormick, Manfred Padberg, Dana Richards, Giovanni Rinaldi, Warren D. Smith, Alexander Zelikovsky, and the anonymous referees of our papers. My apologies to the others who I have doubtless forgotten to mention.

Special thanks to J.A.N. Lee for inspiration, for being a model of excellence, and without whom I would never have attended the University of Virginia.

I sincerely thank my employers Wayne Zandbergen and Garth Morgan for the tremendous encouragement and support they have provided from the day I joined S^{3}I. I am eternally grateful to Karla Hoffman for hooking me up with them during a time of great need.

Thanks to Pawel Winter and Martin Zachariasen for providing me with Euclidean FST generation code. Additional thanks to Kurt Mehlhorn, Stefan Näher, Christian Uhrig, and LEDA Software GmbH for providing the research version of LEDA (Library of Efficient Data types and Algorithms) — which is used extensively by the Euclidean FST generator.

I thank Joe Ganley for his LaTeX macros that made this dissertation much easier to prepare.

During my doctoral studies, I have received financial support from Telenex Corporation and from System Simulation Solutions, Incorporated. Their support is greatly appreciated.

This document was prepared using LaTeX version 2_ε. Some figures were prepared using `xfig` and `gnuplot`.

Contents

List of Figures

List of Tables

Abstract

This dissertation examines 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. In the process, however, it addresses a much more general and widely applicable problem, that of finding a minimum-weight spanning tree in a hypergraph.

The geometric Steiner tree problem is known to be NP-complete for the rectilinear metric, and NP-hard for the Euclidean metric. The fastest exact algorithms (in practice) for these problems use two phases: First a small but sufficient set of full Steiner trees (FSTs) is generated and then a Steiner minimal tree is constructed from this set. These phases are called FST generation and FST concatenation, respectively, and an overview of each phase is presented. FST concatenation is almost always the most expensive phase, and has traditionally been accomplished via simple backtrack search or dynamic programming.

The spanning tree in hypergraph problem is defined, and is proven to be strongly NP-complete. The minimum-weight spanning tree (MST) in hypergraph problem is then motivated by showing that FST concatenation reduces to MST in hypergraph in a simple way. The MST in hypergraph problem is then formulated as an integer program using subtour elimination constraints.

The spanning tree in hypergraph polytope, STHGP(n), is introduced and a number of its properties are proven. In particular, every constraint used in the integer program is shown to define a facet of STHGP(n). An alternate integer programming formulation based on cutset constraints is presented, but is shown to have an LP relaxation that is weaker than that of the subtour formulation. A simple formula for the number of extreme points in STHGP(n) is shown, thereby generalizing the classical tree enumeration problem of Cayley to hypergraphs.

A branch-and-cut algorithm for the MST in hypergraph problem is presented. This algorithm is applied to the FST concatenation problem. Experimental results are presented for a large set of problem instances of various sizes up to 1000 terminals. Optimal rectilinear and Euclidean Steiner trees are obtained for every instance. A single 2000 terminal Euclidean instance is also solved to optimality. These results show that the new algorithm is by far the fastest in existence, since the best previously published Steiner tree results are 70 terminals for rectilinear and 150 terminals for Euclidean, respectively.

A number of directions for future work are outlined, and in conclusion it is noted that this two-phase approach works for any distance metric in any finite dimension — even the Steiner problem in graphs — provided a suitable FST generation algorithm is available.

1

Introduction

The Steiner tree problem is one of the oldest optimization problems in all of mathematics. Although the ancient Greeks knew that the shortest path connecting two points was a straight line, it was apparently Fermat who first asked what the shortest path was connecting *three* points. Torricelli provided a geometric construction for this by 1640 — 56 years before Johann Bernoulli posed his famous brachistochrone problem. In 1934 Jarník and Kössler [31] posed the general Euclidean problem in the plane, which was popularized by Courant and Robbins in their famous 1941 book "What Is Mathematics?" [13] — although they incorrectly attributed the problem to Steiner! In 1966 Hanan [26] first considered the rectilinear variant, which is currently very important due to its connection with routing of circuit nodes in VLSI and printed circuit boards.

Given a finite set V of points in the plane (called *terminals*), the Steiner tree problem is to find a minimum-length interconnection of those terminals according to some geometric distance metric. The resulting interconnection is a tree, called a Steiner minimal tree. Nodes $s \notin V$ of degree 3 or greater are known as *Steiner points*, and are introduced as necessary to achieve the shortest possible interconnection.

Let $u = (u_x, u_y)$ and $v = (v_x, v_y)$ be two points in $\mathbb{R}^2$. Then the distance in the L_p-metric, $1 \leq p \leq \infty$, between u and v (or simply the L_p distance) is $(|u_x - v_x|^p + |u_y - v_y|^p)^{1/p}$.

For the Steiner tree problem the most common special cases are $p = 1$ and $p = 2$: the L_1 (rectilinear or Manhattan) distance $|u_x - v_x| + |u_y - v_y|$, and the L_2 (Euclidean) distance $\sqrt{(u_x - v_x)^2 + (u_y - v_y)^2}$, respectively. The corresponding Steiner tree problem variants are known as the *rectilinear Steiner minimal tree* (RSMT) and *Euclidean Steiner minimal tree* (ESMT) problems. The decision form of RSMT is known to be NP-complete [21]. The decision form of ESMT would be NP-complete, except that the problem is not known to be in NP. This follows from the fact that the lengths of Steiner trees can be complicated algebraic numbers, and it is not yet clear whether trustworthy computation with such numbers can be done in polynomial time. A suitably discretized version of the ESMT problem has been shown to be NP-complete, however [20].

The rectilinear problem is equivalent to requiring that all interconnecting line segments be horizontal or vertical. See Figure 1.1 for an illustration of an RSMT for 70 terminals.

The Euclidean problem is characterized by line segments forming angles that are always 120 degree or more. In particular, all Steiner points have degree 3 and form angles of precisely 120 degrees. See Figure 1.2 for an illustration of an ESMT for 100 terminals.

The RSMT problem has numerous applications in the area of VLSI design automation as well as printed circuit board layout. For example, an RSMT for a set of electron devices can be used as a lower bound estimate on the wire length of a route connecting all of the devices together. An RSMT of the points represents only a lower bound since a real interconnect satisfies additional constraints requiring it to avoid other obstacles that are also present on the chip. Recent work by Ganley [17] treated such *obstacle-avoiding* RSMTs directly. In addition to global wire length estimation, RSMTs have also been used to evaluate the merit of functional block placements in floor-planners such as the *MONDRIAN* system [17]. Wagner [57] reduces certain cases of parallel expression evaluation to the RSMT problem.

The ESMT problem has applications in the design of electrical power distribution networks, oil and natural gas pipelines and other network design problems.

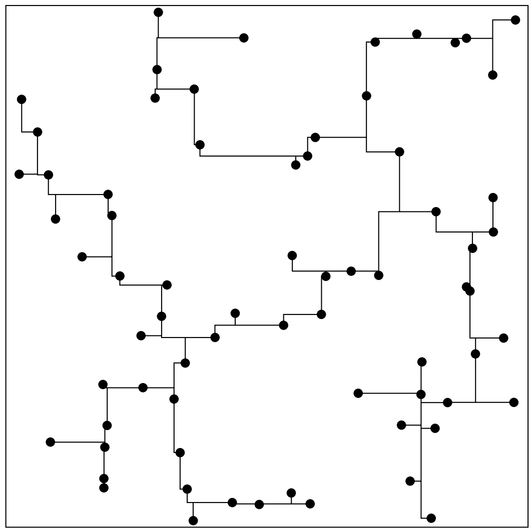


Figure 1.1: A rectilinear Steiner minimal tree for 70 terminals.

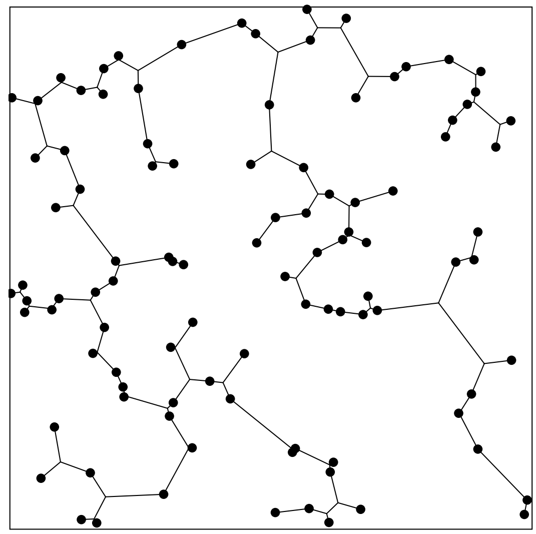


Figure 1.2: A Euclidean Steiner minimal tree for 100 terminals. (Problem 1 from OR-library `estein100.txt` file.)

Some of these applications require the solution of problem instances containing many hundreds or even thousands of terminals. Provably optimal solutions to such instances were well beyond the capabilities of previous methods, but are becoming feasible with the algorithm presented here.

The research described here focused initially on the rectilinear problem, adapting Winter's groundbreaking Euclidean work [60] to the rectilinear problem. Although the initial results of these efforts represented a significant advance for the rectilinear problem, they fell disappointingly short of the 100 terminal solutions obtained for the Euclidean problem. During the efforts to close this gap, however, it was discovered that a much more general and widely applicable problem — the minimum spanning tree in hypergraph problem — was lurking inside. The solution presented here for the MST in hypergraph problem represents a quantum breakthrough for computing Steiner trees. Nevertheless, the MST in hypergraph results are likely to be more important and generally applicable in the long run.

1.1 Definitions

In order to further discuss the Steiner tree problem, a number of key terms must be formally defined, the most important one being *full Steiner tree* (abbreviated *FST*).

Let V be a set of n points in the plane called *terminals*. A Steiner minimal tree T for V is said to have a *full topology* if every vertex in V is a leaf in T. A terminal set V is said to be a *full set* if every Steiner minimal tree for V has a full topology. A terminal set that is a full set and also has size k is said to be a *full set of size* k. With respect to a point set V, a set $S \subseteq V$ is said to be a *full set with respect to* V if S is a full set, and there is some Steiner minimal tree for V that contains a full topology of S as a subgraph. A Steiner minimal tree T for a full set $S \subseteq V$ is said to be a *full Steiner tree* (FST) of V. For any FST F we define $|F|$ to be the total length of F according to the appropriate distance metric. If $\mathcal{F}$ is a set of FSTs we define $\cup\mathcal{F} = \cup_{F \in \mathcal{F}} F$, the union of these FSTs in the plane.

The key concept to be grasped here is that if a subset $S \subseteq V$ is a *full set*, then it is possible to achieve a minimal interconnection of the terminals S (in the context of an SMT for V) only by routing *to* them, not *through* them (nor through any other terminals in V). A full Steiner tree (FST) is simply a particular such minimal tree interconnecting S.

In an intuitive sense this means that the terminals S reside at the periphery of some region, and all interconnections between the terminals of S lie inside this region, which is empty of terminals. Although this is literally true for rectilinear FSTs, it is only figuratively true of Euclidean FSTs, where this routing region can have a complicated branching tree structure — even forming arbitrary spirals.

1.2 Previous Work

The first finite algorithm for the Euclidean Steiner tree problem was given by Melzak [40]. It works by explicitly enumerating all possible tree topologies, computing a relatively minimal configuration for each. The shortest is retained and is the ESMT. Cockayne [9] improved the method, which was later coded by Cockayne and Schiller [12] and handled problems with up to 7 terminals. Boyce and Seery [7] improved the method so that 10 and later 12 terminal problems could be solved. Hwang provided an $O(n)$ solution to the Melzak FST algorithm, a crucial subroutine in the method [28].

Winter [60] devised a totally different approach that first generates all possible FSTs, and then constructs a Steiner minimal tree by choosing a subset of the FSTs that span the terminals with minimal length. Problems up to 15 terminals were solved quite rapidly. Further improvements were made by Cockayne and Hewgill [10, 11], who reported solutions of problems up to 100 terminals.

Recently Winter and Zachariasen [62] refined these methods even further, solving problems up to 150 terminals.

Other exact ESMT algorithms include the *negative edge algorithm* of Trietsch and Hwang [56], and the *luminary algorithm* of Hwang and Weng [30]. Neither of these algorithms have been implemented.

The rectilinear problem was introduced in 1966 by Hanan [26], who characterized optimal solutions for $n \leq 5$ terminals. Hanan also showed that an RSMT always exists as a subgraph of a *grid graph*, obtained by constructing horizontal and vertical lines through each terminal. The first exact algorithm in the literature appeared in 1972 by Yang and Wing [63], who report solving problems with up to 9 terminals. No further computational advances appear in the literature until 1989.

In 1976, Hwang completely characterized the rectilinear FSTs [27]. This important result forms the basis of all known rectilinear FST generators, including the rectilinear results reported in this dissertation.

Further computational progress resumed in 1989 when Sidorenko [50] reported an algorithm applicable up to 11 terminals. Similar results were reported by Lewis, Pong and Van Cleave [36] in 1992. Thomborson, Alpern and Carter [54] report solving problems with up to about 16 terminals in 1992. The algorithms of Ganley and Cohoon [18, 19] handle about 18 and 28 terminals, respectively in 1994.

In 1993, Salowe and Warme [48] made a significant advance by adapting the Euclidean results of Winter [60] and Cockayne and Hewgill [10, 11] to the rectilinear problem — solving most 30 terminal instances in an average of 30 minutes. Further refinements [49] increased this to about 35 points. In 1997, Fößmeier and Kaufmann further refined the approach so that most 70 terminal problems are solved, which are the best results currently appearing in the literature.

Virtually all other exact algorithms for the rectilinear problem use the seductively simple Hanan grid graph reduction to the Steiner problem in graphs. This reduction has been by far the most popular approach to computing RSMTs. Various exact algorithms for the Steiner problem in graphs have been tried on grid graphs, including the dynamic programming method of Dreyfus and Wagner [15, 54], Hakimi's method [25] as well as sophisticated branch-and-cut methods [38, 34]. However, even the most sophisticated branch-and-cut codes fail to solve instances much larger than 40 terminals due to the extreme degeneracy of the Hanan grid graph.

In 1996 the author in collaboration with Abilio Lucena solved several of the 100 terminal instances from the OR-library. Lucena's branch-and-cut code was used to solve the Steiner problem in a graph obtained by taking the union of all the rectilinear FSTs. The resulting graphs are extremely sparse compared to Hanan grid graphs (see Figures 1.3 and 1.4), and are much easier to solve. Although further improvement in these graphs seem possible using the graph reductions devised by Winter [61], this approach seems unlikely to meet or overtake the methods presented here.

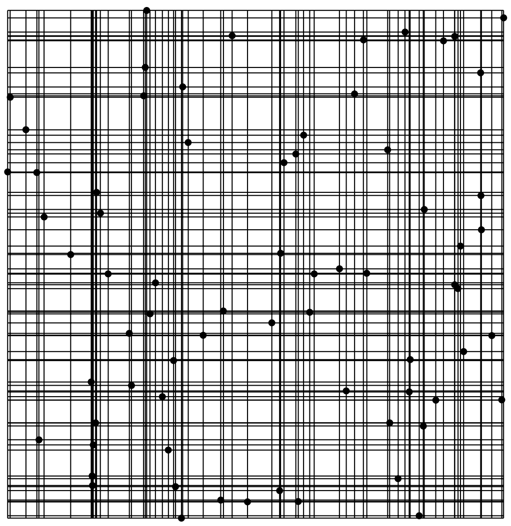


Figure 1.3: Hanan grid graph for problem in Figure 1.1.

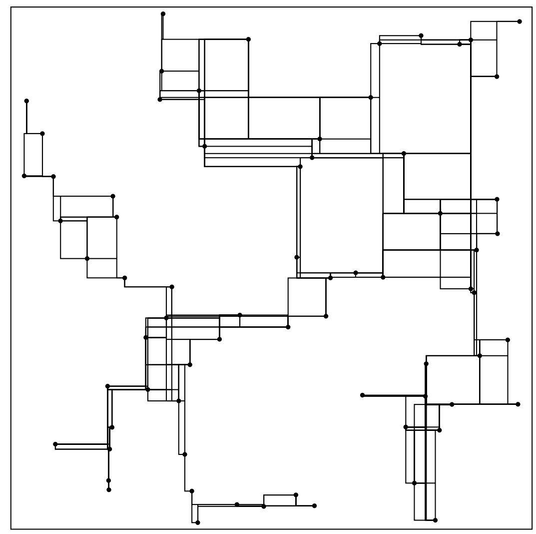


Figure 1.4: Union of FSTs graph for problem in Figure 1.1.

The research described in this dissertation builds upon the author's previous breakthrough [48, 49] achieved during his M.S. studies. The new method results in provably optimal solutions to random problem instances having up to 1000 terminals. Winter and Zachariasen generously provided source code for their new Euclidean FST generator [62], permitting these results to be re-applied to the Euclidean problem — resulting in optimal solutions to problems having up to 2000 terminals. See Figure 1.5 for a timeline showing progress on the Euclidean and rectilinear Steiner tree problems.

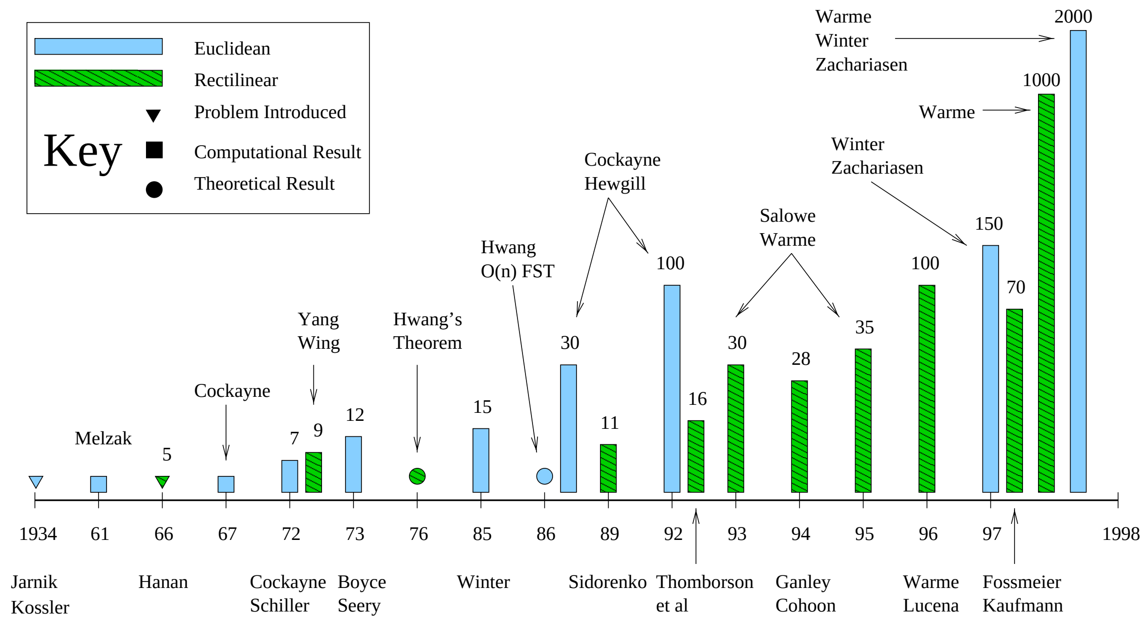


Figure 1.5: Progress on Euclidean and rectilinear Steiner tree problems.

2

The Steiner Tree Problem

This chapter discusses the Steiner tree problem in depth, and its solution using FST generation followed by FST concatenation. An overview of the key ideas behind Euclidean and rectilinear FST generation are presented — primarily so that the dissertation may be more self-contained. For the entire story, consult [60, 62] for Euclidean FST generation and [49, 64] for rectilinear FST generation.

See [29] for a more comprehensive treatment of Steiner tree results and methods.

2.1 Overview of the FST Concatenation Method

In this section we give a brief overview of the FST concatenation method for computing Steiner minimal trees.

The following is a well-known *folk theorem* of Steiner tree lore:

Theorem 2.1 *Let V be a set of terminals, with $|V| \geq 2$. Then V has a Steiner minimal tree that consists of one or more full topologies over full sets with respect to V. These full topologies intersect only at terminals of degree two or greater.*

This theorem validates a two-phase scheme originally suggested by Winter [60] for the Euclidean problem. The idea is as follows: In the first (FST generation) phase we generate a (usually small) set $\mathcal{F}$ of FSTs containing at least one SMT identified as a subset. In the second (FST concatenation) phase we find a subset $\mathcal{F}^* \subseteq \mathcal{F}$ with minimum total length that fully connects V.

This scheme was first applied to the rectilinear problem by Salowe and Warme [48, 49]. To illustrate the method on a rectilinear problem, Figures 2.1 and 2.2 present all 216 members of $\mathcal{F}$ (rectilinear FSTs) obtained by the Salowe-Warme FST generation algorithm [48, 49] for the 70 point problem shown in Figure 1.1. The reader may verify that the RSMT shown in Figure 1.1 is the union of 35 FSTs, each of which can be found in Figures 2.1 and 2.2.

In general, members of $\mathcal{F}$ are identified by efficiently *eliminating* those subsets of V that *cannot* be full sets with respect to V. Those subsets that remain might not all be true full sets with respect to V, so we refer to them as *candidate full sets*, and their corresponding full topologies as *candidate FSTs*. Note that it is neither practical nor necessary to establish that the members of $\mathcal{F}$ are true SMTs over full sets with respect to V — we need only guarantee that at least one SMT be present as a subset of $\mathcal{F}$. In the sequel we will neglect the distinction between true FSTs and candidate FSTs.

We would like $|\mathcal{F}|$ to be as small as possible. Although there are point sets that give rise to an exponential number of FSTs [16], empirical data shows the expected number to be linear for uniformly distributed V. This is often considered a weakness of the FST approach, since it yields a doubly-exponential algorithm in the worst case. In practice it is by far the fastest exact algorithm known.

It is often possible for an FST generation algorithm to compute an *incompatibility relation* $C \subset \mathcal{F} \times \mathcal{F}$ such that $(F, G) \in C$ implies that F and G cannot appear together in an optimal SMT for V. The validity of $(F, G) \in C$ ultimately appeals to showing that any solution containing both F and G is necessarily suboptimal. Having a significant number of incompatible FST pairs greatly reduces the search space during FST concatenation. There are also FST *pruning* methods that can rule out additional members of $\mathcal{F}$ once the entire

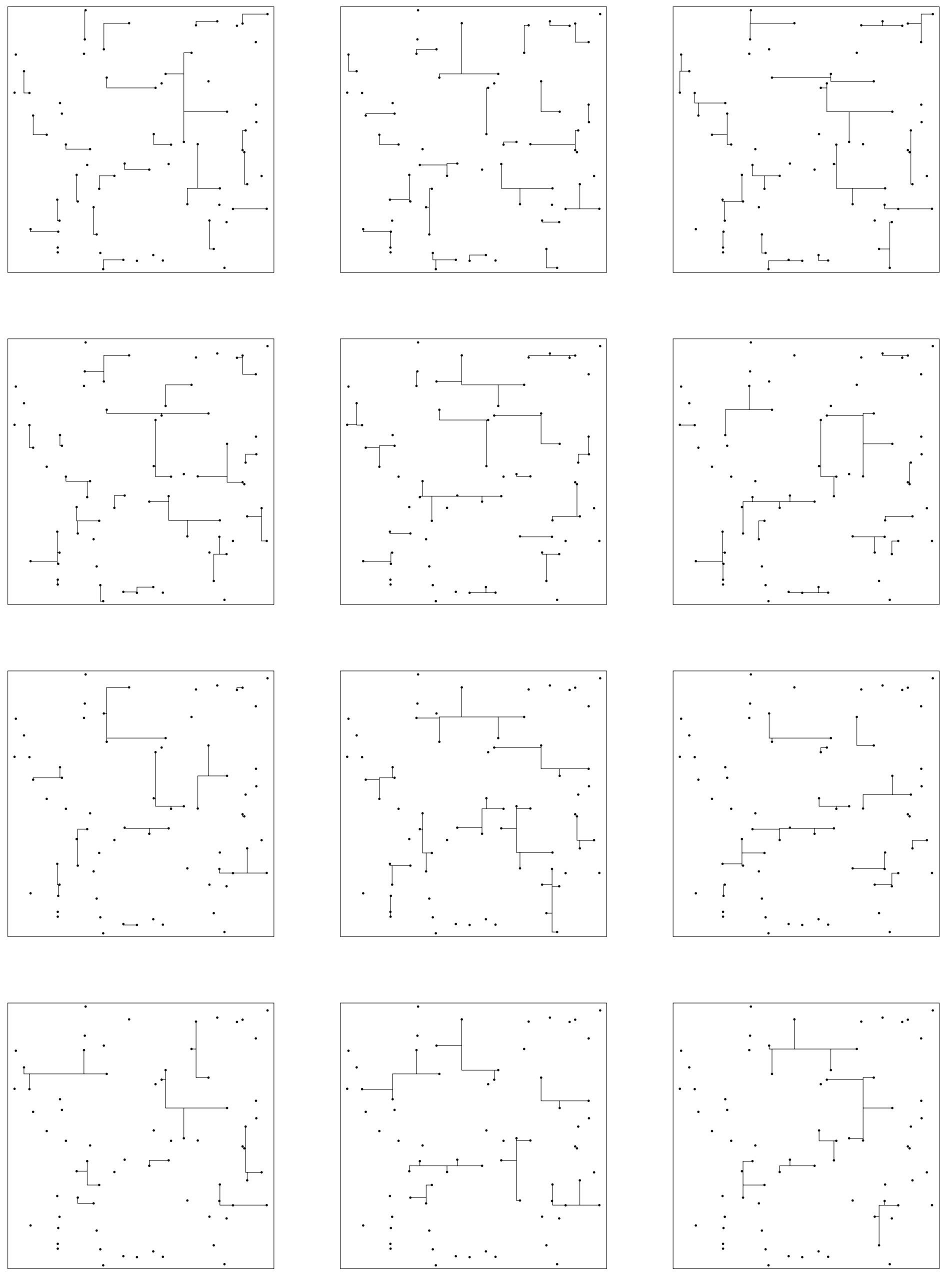


Figure 2.1: All rectilinear FSTs for problem in Figure 1.1.

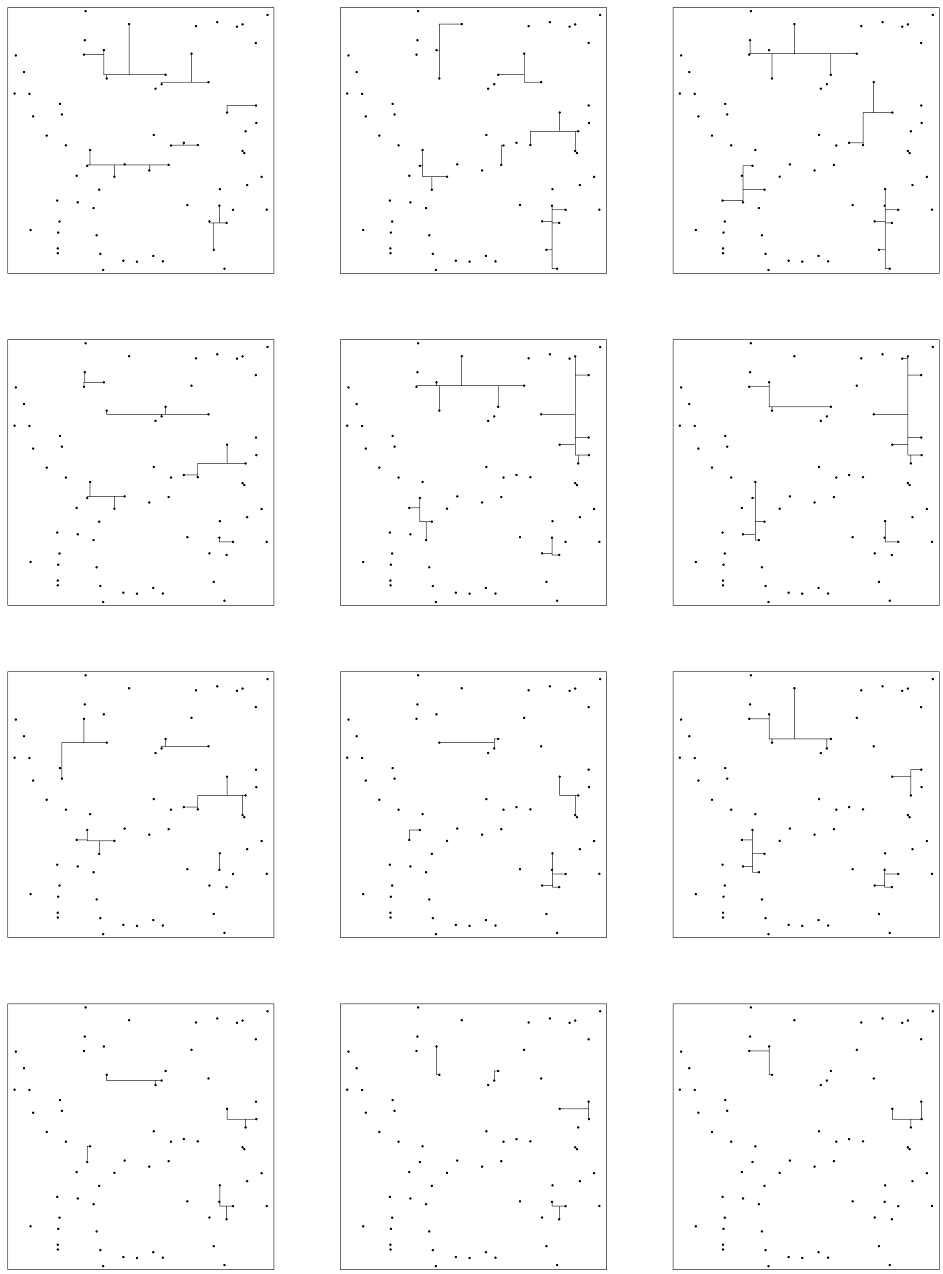


Figure 2.2: All rectilinear FSTs for problem in Figure 1.1 (cont).

set is available. The cost of current pruning methods is greater than their benefit to the FST concatenation algorithm presented here.

2.2 General Properties of FSTs

There are several tests used to eliminate FSTs from consideration that work for any metric. The most important of these are the *lune test*, the *bottleneck Steiner test* and *upper bounds*.

2.2.1 Lune Property

Consider two vertices u and v in a Steiner minimal tree that are connected by a segment containing no intervening terminals or Steiner points. (The two vertices may be any mix of terminals or Steiner points.) Suppose there is a terminal $w \in V$ such that $|w - u| < |u - v|$ and $|w - v| < |u - v|$, where $|a - b|$ is the distance between a and b under the metric being used. Now delete segment uv from the tree, splitting the tree into two connected components. If terminal w is in the same component as u, reconnect the tree by adding segment wv, otherwise reconnect the tree by adding segment wu. The resulting tree is shorter in either case, contradicting the assumption that the original tree was a Steiner minimal tree. No such terminal w can therefore exist.

This is a simple but powerful concept. Figure 2.3 illustrates the Euclidean case. The shaded region is called a *lune* and its interior must be devoid of terminals or the line segment must be removed from consideration. Figures 2.4 and 2.5 show the analogous regions in the rectilinear metric. For consistency, such regions are also called *lunes* regardless of what shape they have in a particular metric.

2.2.2 Bottleneck Steiner Distances

Construct a minimum spanning tree (MST) for the set V of terminals. For every $u, v \in V$ let b_{uv} denote the length of the longest edge on the unique path from terminal u to terminal

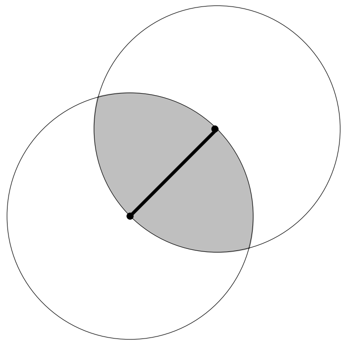


Figure 2.3: Euclidean lune.

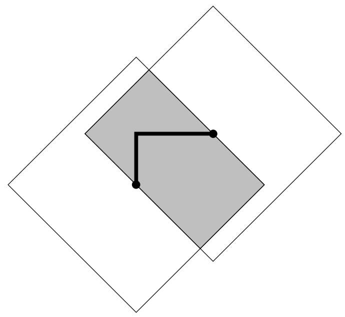


Figure 2.4: Rectilinear corner lune.

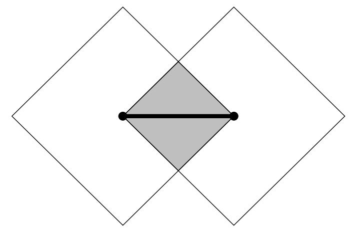


Figure 2.5: Rectilinear diamond.

v in the MST. We refer to b_{uv} as the *bottleneck Steiner distance*. Consider a Steiner minimal tree T for V. Suppose the longest edge between u and v in T has length $l > b_{uv}$. Delete this segment from T thereby splitting T into two connected components — one containing u, the other v. Let $S \subset V$ be the terminals in the component containing u. The terminals in the other component are therefore $V - S$. Consider the unique path from u to v in the MST. At least one of these edges will span the cut from S to $V - S$; any such edge can be used to reconnect T. Furthermore all such edges have length at most b_{uv} making the resulting tree shorter. This contradicts the assumption that T is a Steiner minimal tree.

This is another powerful tool for eliminating FSTs from consideration. Bottleneck Steiner distances for all pairs of terminals can be computed as a preprocessing step. The MST can be computed in $O(n \log n)$ time. The bottleneck Steiner distance from one terminal to all others can be computed in $O(n)$ time via depth-first traversal, implying $O(n^2)$ total preprocessing time. Thereafter a potential FST F can be eliminated if any edge on the unique path in F between two terminals $u, v \in F$ is longer than b_{uv}.

Consider an FST F spanning terminals $S \subseteq V$. It is easy to show that if the length of F exceeds that of a minimum spanning tree for S computed using bottleneck Steiner distances, then F cannot be part of a Steiner minimal tree.

2.2.3 Upper Bounds

Any heuristic that generates valid Steiner trees (not necessarily minimal) for a given set of terminals can be used as an upper bound test. Suppose an FST F spanning terminals S has length that exceeds that of a heuristic Steiner tree for S. Then F cannot be part of a Steiner minimal tree.

2.3 Euclidean FST Generation

We now give a brief overview of the FST generation process for the Euclidean distance metric. These results are **not** original, and are presented for completeness only. The full details are in Winter and Zachariasen [62].

All line segments within a Euclidean SMT must meet at angles of 120° or more, otherwise the tree can be easily shortened. We refer to this property as the *angle condition*. Steiner points therefore always have degree three, forming angles of exactly 120°.

Let p and q be two points in the plane. The *equilateral point* e_{pq} is the point obtained by rotating point q counter-clockwise by an angle of 60° around point p. Points p, q and e_{pq} are then the vertices (in counter-clockwise order) of an equilateral triangle. Note that e_{qp} is different from e_{pq}. Points p and q are called the *base points* of e_{pq}.

The circle circumscribing $\triangle p\, e_{pq}\, q$ is called the *equilateral circle* of p and q and is denoted C_{pq}. Its center is denoted o_{pq}. The *Steiner arc from p to q* is the counter-clockwise arc from p to q on C_{pq}, and is denoted $\widehat{pq}$. The same notation is used to denote subarcs of the Steiner arc: if $p', q' \in \widehat{pq}$, then the subarc from p' to q' is denoted $\widehat{p'q'}$. Such arcs and subarcs are always considered to be counter-clockwise, so that if $p' \in \widehat{q'q} \setminus \{q'\}$, then $\widehat{p'q'}$ is empty.

Consider the equilateral triangle and circle for p and q shown in Figure 2.6. The point r is such that line segment re_{pq} intersects the interior of arc $\widehat{pq}$ at point s. It is easy to see that $\angle q\, s\, e_{pq} = \angle p\, s\, e_{pq} = 60°$: Let x be the intersection of segments pq and se_{pq}. Then $\triangle q\, s\, x \sim \triangle p\, x\, e_{pq}$, because $\angle q\, x\, s = \angle p\, x\, e_{pq}$ and $\angle s\, q\, p = \angle s\, e_{pq}\, p$ since they both subtend arc $\widehat{ps}$. This implies that $\angle q\,s\,x = \angle x\,p\,e_{pq} = 60°$. The same argument applies to $\triangle q\,x\,e_{pq}$ and $\triangle s\,x\,p$ with arc $\widehat{sq}$. Therefore s satisfies the $120°$ angle property required by the Steiner point for terminals p, q and r. It can also be shown that the total length of segments ps, qs and rs is equal to the length of segment re_{pq}, which is also known as the *Simpson line* for the FST over terminals p, q and r.

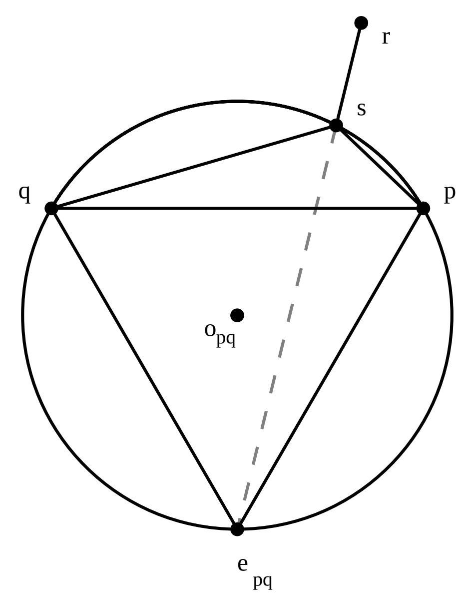


Figure 2.6: Simpson line construction of Steiner point.

Any FST can be constructed via recursive application of this principle. If terminals p and q are both adjacent to Steiner point s, then points p, q, s and their adjoining segments ps, qs and rs can be replaced with point e_{pq} and segment re_{pq}. The procedure is iterated until only a single Simpson line (from an equilateral point to a terminal) remains. Figure 2.7 presents an example in which the entire FST is represented by the Simpson line from z_6 to e_4. The resulting FST of terminals z_1 through z_6 is illustrated with bold lines. Figure 2.8

shows the tree structure by which the equilateral points e_1 through e_4 are derived. For example, e_3 is constructed from base points e_2 and e_1 so that $e_3 = e_{e_2\,e_1}$.

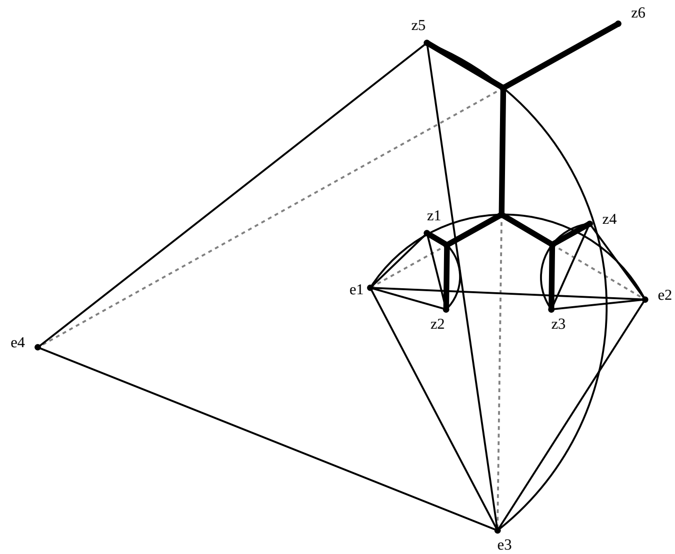


Figure 2.7: Construction of FST from recursive equilateral points.

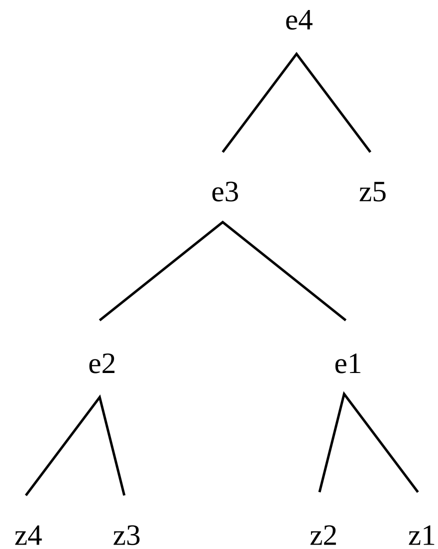


Figure 2.8: Tree structure of equilateral points.

In general, the base points of equilateral points can be either terminals or other equilateral points. For any equilateral point or terminal x we define the *order* of x, $ORD(x)$, to be the maximum depth of the derivation tree by which point x is constructed. Consequently $ORD(p) = 0$ for all terminals p, and $ORD(e) \geq 1$ for all equilateral points e. For a given point x (equilateral or terminal) the set of all terminals in x's derivation tree is denoted $Z(x)$. Consequently, $Z(p) = \{p\}$ for all terminals p.

The key idea of Winter's Euclidean FST generation method is to generate all possible equilateral points by combining pairs of existing equilateral points whose derivation trees are disjoint. When no new equilateral points are possible, the process terminates.

A list $\mathcal{E}$ initially contains the terminals (i.e., equilateral points of zero order). For each $p, q \in \mathcal{E}$ an attempt is made to construct e_{pq}. Equilateral point e_{pq} is appended to $\mathcal{E}$ if and only if $Z(p) \cap Z(q) = \emptyset$, $|Z(p)| + |Z(q)| < n$, and e_{pq} passes a series of pruning tests (described below). Each member p of $\mathcal{E}$ is given a distinct index variable i_p that indicates the next member $q \in \mathcal{E}$ to try combining with p. Whenever a new equilateral point p is added to $\mathcal{E}$, i_p is initialized to point to the beginning of the list $\mathcal{E}$. The process terminates when all of the i_p have advanced to the end of the list $\mathcal{E}$. This guarantees that each pair (p, q) is tested exactly once.

This process would create a combinatorial explosion of equilateral points, except that the pruning tests are efficient and highly effective at identifying equilateral points that cannot give rise to valid FSTs. Such equilateral points are not retained. Note that when an equilateral point e is pruned it eliminates the need to ever consider any other equilateral point having e's derivation tree as a subtree.

Let e_{xy} be an equilateral point of non-zero order with base points x and y. In most cases it is possible to deduce that any Steiner point on $\widehat{xy}$ would be invalid unless confined to a subarc $\widehat{x'y'}$ of $\widehat{xy}$. Consequently each such $e_{xy} \in \mathcal{E}$ has an associated *feasible Steiner subarc* $\widehat{x'y'}$ that is a subarc of $\widehat{xy}$. Most of the pruning tests work by further restricting the feasible Steiner subarc. If this subarc becomes empty, e_{xy} can be pruned.

Once all equilateral points have been generated, it is easy to contruct all of the FSTs. Let $e_{xy} \in \mathcal{E}$. For every terminal $v \notin Z(e_{xy})$, the corresponding FST exists if and only if line segment ve_{xy} intersects the feasible Steiner subarc $\widehat{x'y'}$ of e_{xy}. To obtain the FST, process Simpson line ve_{xy} recursively as follows: a Simpson line ze (where z is a known point and e an equilateral point) results in line segment ze if e is of zero order. Otherwise $e = e_{pq}$, so let $s = ze \cap \widehat{pq}$, add line segment zs to the FST, and process sp and sq recursively. Note that by symmetry, it is necessary to consider only $v \notin Z(e_{pq})$ whose index exceeds that of all terminals in $Z(e_{pq})$, according to an arbitrary ordering of the terminals.

We now very briefly present several of the pruning tests that equilateral points must pass in order to be retained in $\mathcal{E}$. For the complete discussion including additional tests, see [62].

2.3.1 Projections

Let p and q be two equilateral points and suppose that p is of nonzero order. Let a and c be the base points of p so that $p = e_{ac}$. Furthermore, let $\widehat{a'c'}$ be the feasible Steiner subarc of e_{ac}. The relative locations of a' and c' with respect to p and q can be used to rule out portions of Steiner arc $\widehat{pq}$, thus reducing the feasible Steiner subarc. In most cases it can be shown that *any* Steiner point on $\widehat{pq}$ would violate an angle condition, indicating that e_{pq} can be pruned. In fact, only four specific subcases are retained. These cases are (Figure 2.9):

1. $0° < \angle c' \, p \, q \leq \angle a' \, p \, q \leq 120°$, q is not in the interior of C_{ac}, a' is not in the interior of C_{pq}, and c' is in the interior of C_{pq}. Only the portion $\widehat{xq'}$ of $\widehat{pq}$ that is outside of C_{ac} and visible from p through $\widehat{a'c'}$ is feasible.

2. $\angle c' \, p \, q \leq 0° < \angle a' \, p \, q \leq 60°$, q is not in the interior of C_{ac}, and a' is not in the interior of C_{pq}. Only the portion $\widehat{xq}$ of $\widehat{pq}$ that is outside of C_{ac} is feasible.

3. $0° < \angle c' \, p \, q \leq \angle a' \, p \, q \leq 60°$, q is not in the interior of C_{ac}, and a' is in the interior of C_{pq}. Only the portion $\widehat{p'q'}$ of $\widehat{pq}$ that is visible from p through $\widehat{a'c'}$ is feasible.

4. $\angle c' \, p \, q \leq 0° < \angle a' \, p \, q \leq 60°$, q is not in the interior of C_{ac}, and a' is in the interior of C_{pq}. Only the portion $\widehat{p'q}$ of $\widehat{pq}$ that is visible from p through $\widehat{a'c'}$ is feasible.

A point $x \in \widehat{pq}$ is *visible* from p through a point y if and only if x is the projection of y onto $\widehat{pq}$ from p. This is why these tests are called the *projection* tests. The arguments are completely symmetric if q is an equilateral point of nonzero order.

2.3.2 Lune Property

Every line segment in an FST must satisfy the lune property (i.e., no terminal may reside in the lune of an FST line segment). Let $L(u, v)$ be the lune for segment uv (i.e., $L(u, v) = \{x : |ux| < |uv| \wedge |xv| < |uv|\}$). Let p and q be equilateral points of any order. Let the feasible Steiner subarc of $\widehat{pq}$ be $\widehat{tu}$. Let $s \in \widehat{tu}$ be a potential Steiner point on this

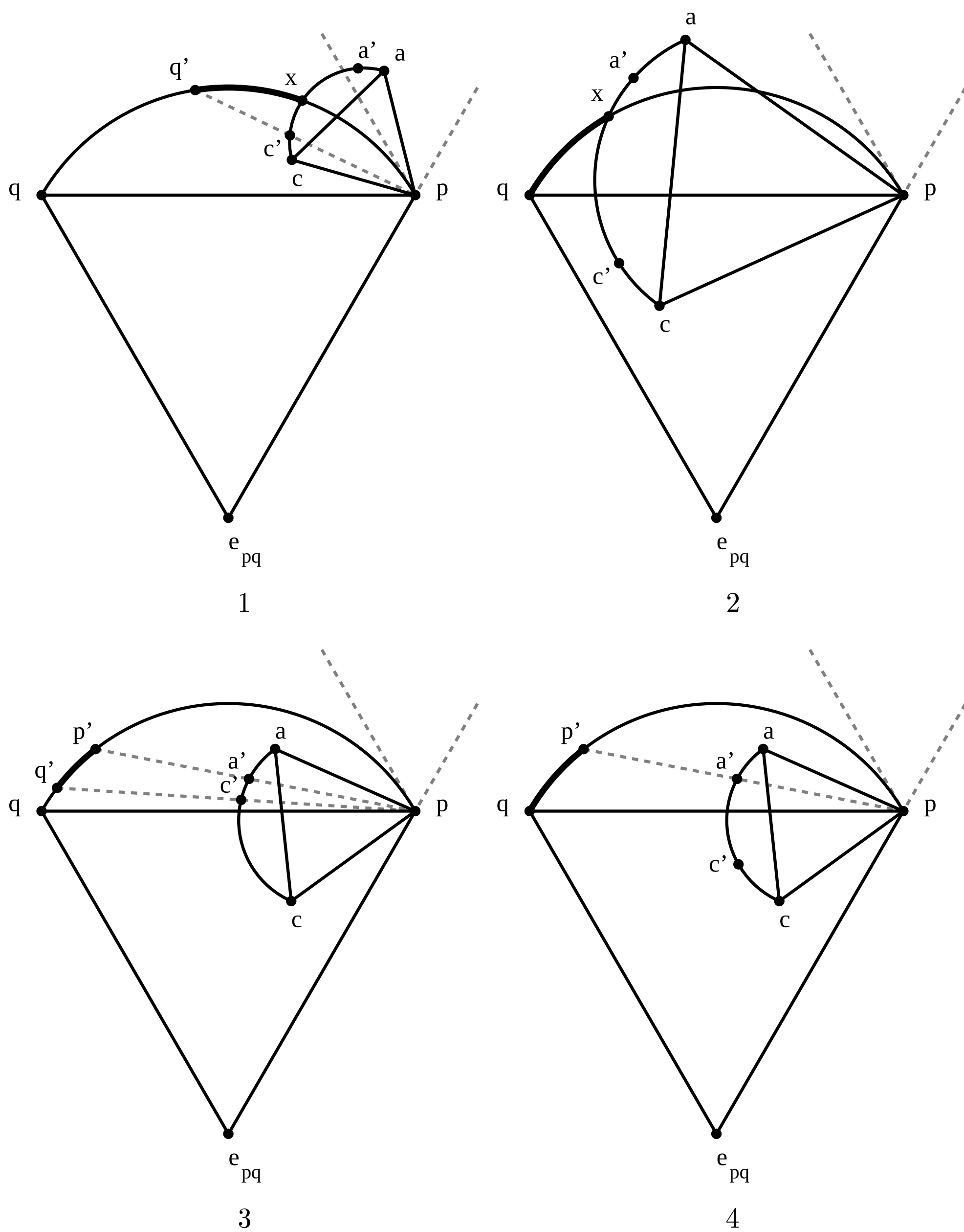


Figure 2.9: Projections: four cases retained.

arc. Then two of the segments incident to s will be known, one directed toward p, the other toward q. Let $E_p(s)$ and $E_q(s)$ be the other end point of the line segment directed toward p and q, respectively. If p (or q) happens to be a terminal then $E_p(s) = p$ (or $E_q(s) = q$). Otherwise $p = e_{bd}$ (or $q = e_{ac}$) is an equilateral point of non-zero order and $E_p(s) = sp \cap \widehat{bd}$ (or $E_q(s) = sq \cap \widehat{ac}$). If $L(s, E_p(s))$ or $L(s, E_q(s))$ contain one or more terminals then we can conclude that s is not a feasible Steiner point.

In particular, if $s = t$ causes non-empty lunes then we can further restrict the feasible Steiner subarc $\widehat{tu}$ by moving t toward q to the first position t' at which both lunes $L(t', E_p(t'))$ and $L(t', E_q(t'))$ are empty. Similary, if $L(u, E_p(u))$ or $L(u, E_q(u))$ are non-empty, we can move u toward p to the first position u' at which both $L(u', E_p(u'))$ and $L(u', E_q(u'))$ are empty. This can actually done in four sequential steps: move t until $L(t, E_q(t))$ is empty, move u until $L(u, E_q(u))$ is empty, move t until $L(t, E_p(t))$ is empty, move u until $L(u, E_p(u))$ is empty.

Figure 2.10 illustrates the first two of these steps. In this figure, $x = E_q(t)$, $y = E_q(u)$, and z is a terminal that makes the corresponding lune non-empty. Figure 2.10a moves t to t' such that $L(t', E_q(t'))$ is empty, and Figure 2.10b moves u to u' such that $L(u', E_q(u'))$ is empty. Note that emptying one lune in this way can cause another to become non-empty, so these tests can be iterated until all four lunes are empty. Of course the equilateral point e_{pq} can be pruned immediately if the feasible arc $\widehat{ut}$ becomes empty during this process.

2.3.3 Bottleneck Property

Let p and q be equilateral points of any order. Let $\widehat{tu}$ be the feasible Steiner subarc of $\widehat{pq}$. Let $x = E_q(t)$ as in Subsection 2.3.2. Let $z_p \in Z(p)$ and $z_q \in Z(q)$ such that $b_{z_p z_q}$ is minimized. If

$$b_{z_p z_q} < |xt|,$$

then t is not a feasible location for a Steiner point, since the bottleneck property is violated by segment xt along the path between z_p and z_q. Point t can be moved toward q until

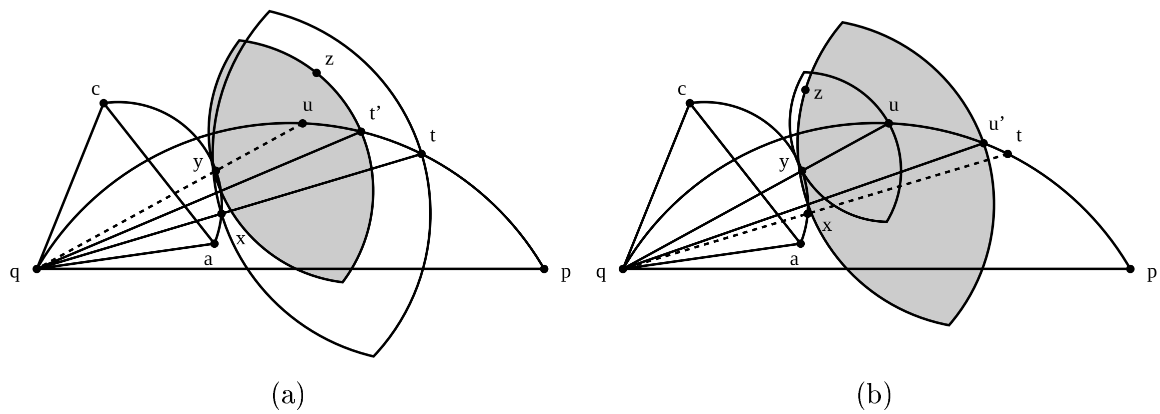


Figure 2.10: The lune property.

equality is achieved (or t moves beyond u). The test is symmetric for segment yu, where $y = E_p(u)$.

2.3.4 Wedge Property

Let p and q be equilateral points of any order. Let $\widehat{tu}$ be the feasible subarc of Steiner arc $\widehat{pq}$. Construct the four rays $r_1 = \vec{pu}$, $r_2 = \vec{e_{pq}u}$, $r_3 = \vec{e_{pq}t}$ and $r_4 = \vec{qt}$ (see Figure 2.11). Let R_1 be the region bounded by r_2, r_3, and $\widehat{tu}$. Let R_2 be the region bounded by r_1 and r_2. Let R_3 be the region bounded by r_3 and r_4.

If R_1 contains no terminals then any Steiner point $s \in \widehat{ut}$ must connect to some other Steiner point s' in R_1. Since R_1 contains no terminals, Steiner point s' resides on the Steiner arc of some equilateral point $e_{ac} \in R_1$. It can be shown that such an e_{ac} cannot be constructed unless there is at least one terminal in R_2 and at least one terminal in R_3. If either R_2 or R_3 is empty then equilateral point e_{pq} can be pruned.

Suppose on the other hand that region R_1 contains at least one terminal. If R_2 is empty, let z be a terminal in R_1 that minimizes $\angle z\, e_{pq}\, u$, and let $u' = ze_{pq} \cap \widehat{tu}$. Then the feasible

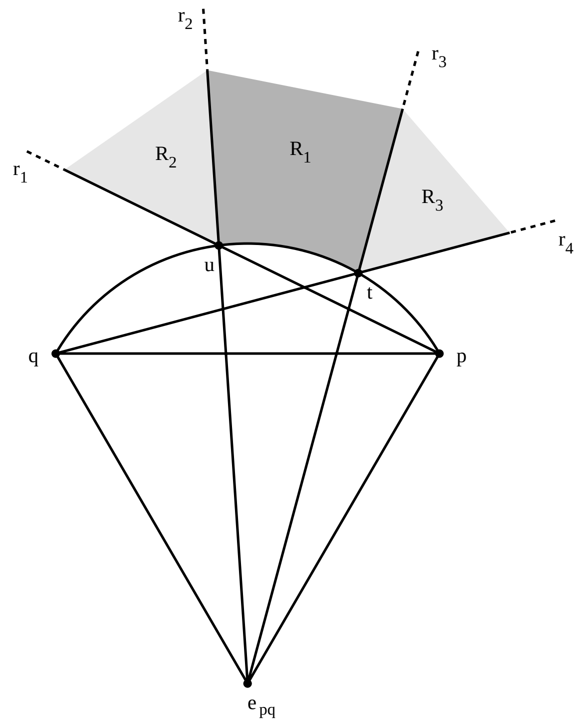


Figure 2.11: The wedge property.

subarc can be narrowed to $\widehat{tu'}$. In similar fashion if R_3 is empty, let z be a terminal in R_1 that minimizes $\angle t\, e_{pq}\, z$, and let $t' = ze_{pq} \cap \widehat{tu}$. Then the feasible subarc can be narrowed to $\widehat{t'u}$.

2.3.5 Euclidean Compatibility Tests

Two FSTs F_i and F_j are incompatible if they intersect anywhere other than at a single terminal. If F_i and F_j meet at a single terminal v, they can be declared incompatible if their line segments form an angle of less than 120° at v.

There are other pruning and compatibility tests that can be used. For complete details, refer to [62].

2.4 Rectilinear FST Generation

We now give a brief overview of the FST generation process for the rectilinear distance metric. These results were previously given in Salowe and Warme [49], and are presented here for completeness only. The more recent methods of Zachariasen [64] are superior, and represent the current state of the art.

2.4.1 Hwang Topologies

Hwang [27] provided a complete description of the rectilinear FSTs, a result known as *Hwang's theorem*:

Theorem 2.2 (Hwang's theorem) *Every rectilinear full set has a rectilinear Steiner minimal tree having one of four topologies. A* type I topology *consists of a* backbone *formed by two segments (a* long leg *and a* short leg*) meeting at a corner and adjacent to two of the terminals*[1]*. The long leg is incident to segments connecting the other terminals to the backbone. (Assume without loss of generality that the long leg is horizontal.) From left to right, these terminals (and the terminal on the short leg) must appear on alternating sides of the long leg. A* type II topology *is similar to a type I topology, but with a single terminal — the leftmost (or rightmost) — connected to the short leg. A* degenerate type I *(or* straight*) topology is similar to a type I topology, but having a short leg of zero length and therefore no corner. A* cross *topology has exactly 4 terminals connected by one horizontal and one vertical segment that meet at a single Steiner point of degree 4.*

[1]The term *long leg* does not imply greater length geometrically, but rather having a potentially greater number of incident segments.

See Figure 2.12 for examples of all four types of topologies. The straight and cross topologies are degenerate cases that appear only when V contains terminals with duplicate x or y coordinates. Note that in general, type I and type II topolgies can have four different orientations times two reflections each, while straight topologies can be either horizontal or vertical.

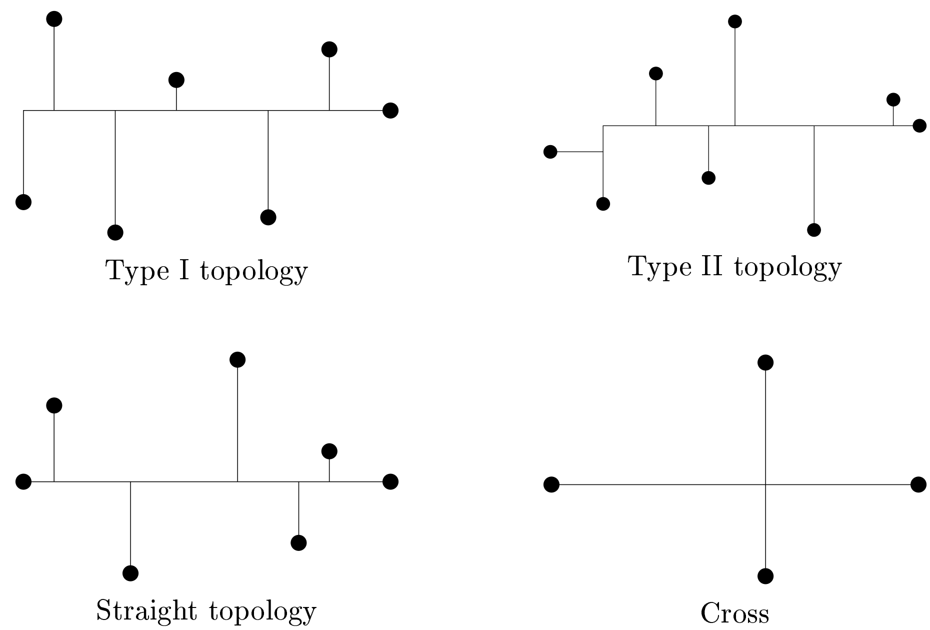


Figure 2.12: The Hwang topologies.

There is some ambiguity in this classification scheme. For example, a non-degenerate FST with 3 terminals could be classified as either a type I or type II topology, depending on which segment is called the long leg. Similarly, in a type II topology with 4 terminals either of two segments can be called the long leg. The classification is unique, however, for 5 terminals or more.

2.4.2 Corner-Flipped Topologies

There are two transforms that can be applied to a Hwang type I or type II topology that do not change its length — the *corner flip* and the *slide*. By iteratively applying these two transformations, the Hwang topology can be converted finally into yet another Hwang topology, having an orientation different from the original. This process is illustrated in Figure 2.13, in which a Hwang type II topology is transformed into a Hwang type I topology of the same length.

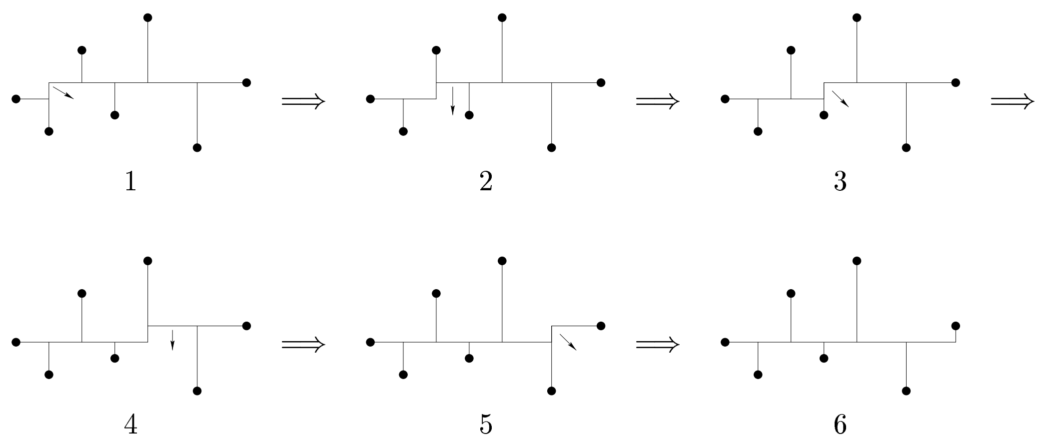


Figure 2.13: The corner-flip and slide transforms.

In general, any Hwang type I or type II topology X can be transformed to another Hwang topology $\hat{X}$ in this way. We say that $\hat{X}$ is the *corner-flipped topology* of X. Of course these transformations work just as well in reverse, so it is also true that X is the corner-flipped topology of $\hat{X}$. Let a Hwang topology be *even* (or *odd*) if it has an even (or *odd*) number of alternating terminals attached to the long leg. Then we can characterize all such transformations as shown in Figure 2.14. If X is a straight topology or a cross then we let $\hat{X} = X$ since there is no corner at which to begin the flip and slide transform.

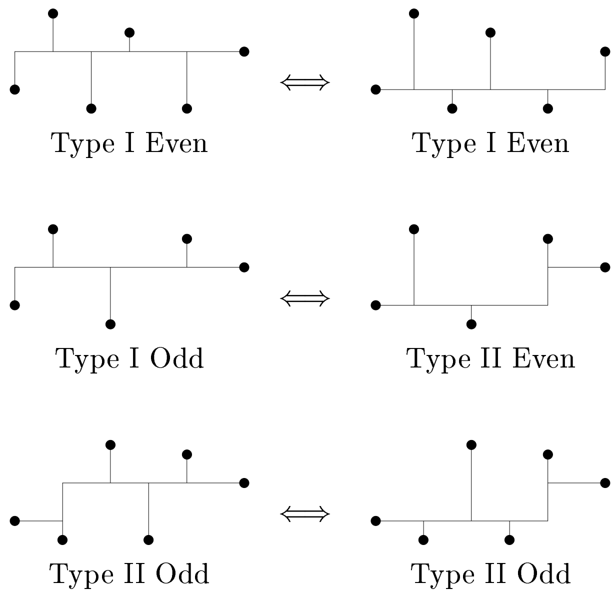


Figure 2.14: The corner-flipped topologies.

Suppose X is a Hwang topology and $\hat{X}$ is its corner-flipped topology. If it can be shown that $\hat{X}$ cannot be an FST, then we can also conclude that X cannot be an FST. Therefore when generating FST X we can usually make our screening tests more effective by applying them to both X and $\hat{X}$.

2.4.3 Empty Regions

Let X be a Hwang topology. The lune property of Section 2.2.1 implies that X cannot be an FST if any of the lunes (i.e., corner lunes or diamonds) defined by its segments are non-empty. Neither can X be an FST if the corner-flipped topology $\hat{X}$ has non-empty diamonds.

It is known (e.g., [5, 49, 64]) and easy to show that certain rectangular regions must also be empty. Let X be a Hwang topology containing segments ab and bc that form a $90°$

angle at point b. Points a and c can be either terminals or Steiner points, but there must not be any terminals or Steiner points in the relative interior of segments ab or bc. Point b can be a terminal, Steiner point or backbone corner. Let d be the point obtained by adding the vector $a - b$ to point c, so that $abcd$ forms a rectangle. If there are terminals in the interior of rectangle $abcd$ then FST X cannot be part of an SMT for V. See Figure 2.15.

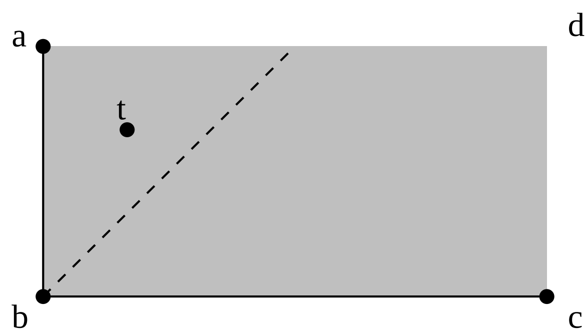


Figure 2.15: Empty rectangles.

Proof: Suppose X is part of an SMT for V, and that a terminal t lies inside rectangle $abcd$ and above the diagonal extending from b into the rectangle as shown in Figure 2.15. Delete segment ab from the tree. If t is in the same connected component as a, then reconnect by adding a vertical segment from t down to segment bc. Otherwise, reconnect by connecting a and t. The tree is shortened in either case, a contradiction. A similar argument applies if t lies below the diagonal line. Now suppose t lies precisely on the diagonal. Since t must be connected to the rest of the tree using only horizontal and vertical segments, there must be some other point u in the tree that lies above or below the diagonal line that we can use to shorten the tree in the same manner. $\square$

As a result, Hwang topologies (and their corner-flipped topologies) must have both empty diamonds, and empty rectangles in order to be an FST. This is illustrated in Figure 2.16. Some additional empty regions are described by Salowe and Warme [49].

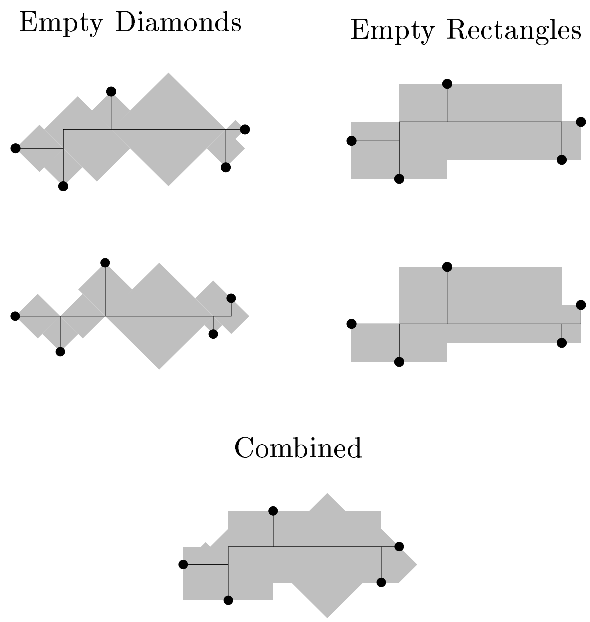


Figure 2.16: Hwang topology empty regions.

2.4.4 Generating Rectilinear FSTs

Hwang's theorem tells us that every valid full set will have at least one FST having one of the four Hwang topologies. We can guarantee, therefore, that by finding all topologies having one of these configurations we will have found all full sets (plus perhaps other subsets that are *not* full sets, which is why they are *candidate* full sets). Perhaps even more importantly, this approach automatically gives us a full Steiner tree for each such candidate full set.

The Salowe-Warme algorithm [49] generates FSTs by considering all pairs (a, b) of terminals as backbones for Hwang topologies. The backbone for (a, b) consists of a vertical line segment incident to a and a horizontal line segment incident to b. These segments meet at a common corner point $c = (a_x, b_y)$. Note that backbone (b, a) represents the corner-flip of backbone (a, b). Because of the symmetry provided by the corner-flip transform, we need consider only pairs (a, b) whose horizontal segment lies to the right of the vertical segment. Each backbone (a, b) is considered twice: once considering the vertical segement to be the *long leg*, and once considering the horizontal segment to be the *long leg*.

Consider a backbone (a, b) with corner c (as shown in Figure 2.17) such that segment cb is the long leg. Consider the set A of all terminals in the shaded region that define an empty diamond when connected to the long leg cb with a vertical line segment. The terminals in A are the candidates for attaching to the long leg of the backbone. Consider also the set B of all terminals in the shaded region of Figure 2.18 that similarly define empty rectangles when connected to short leg ac with a horizontal segment. The terminals in B are the candidates for optionally attaching to the short leg. Each properly alternating combination of attached terminals from A is tried in turn, resulting in a Type I topology. If it survives all of the screening tests it is retained as an FST. Each $t \in B$ is then attached to the short leg in turn, resulting in a Type II topology. Any of these that survive all of the screening tests are then retained as an FST. Recursive enumeration starts at the corner and proceeds down the long leg away from the corner. This makes it easy to guarantee that the candidate nearest the corner is on the proper side of the long leg.

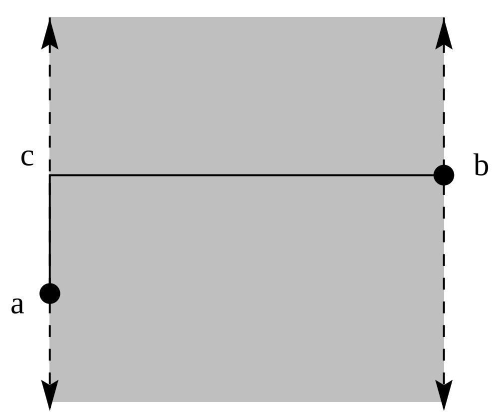


Figure 2.17: Long leg candidate region.

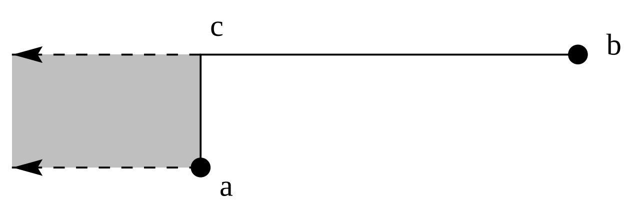


Figure 2.18: Short leg candidate region.

Generating the straight and cross topologies can be done as an easy special case. Sort the terminal set lexically by x/y and by y/x coordinates (major/minor keys). Then $O(n)$ more time is sufficient to find the set of all horizontal and vertical segments (if any) bounded by pairs of terminals with no terminals in the interior. Crosses can be discovered in $O(n^2)$ time by considering each pair of such horizontal and vertical segments. Those that form crosses and pass the screening tests are retained as FSTs.

Each horizontal or vertical segment uv is then considered as the backbone of a straight topology. Recursive enumeration of topologies is then very similar to the type I/type II case, with two exceptions: no short leg candidates are tried since there is no short leg; secondly both alternating directions are valid starting points, since there is no corner.

2.4.5 Screening Tests

Let X be a generated Hwang topology over terminals $U \subset V$, and let $\hat{X}$ be its corner-flipped topology. We check that $\hat{X}$ is a proper Hwang topology (for example, terminals might not properly alternate down the long leg). No terminal may lie in the interior of any segments of X or $\hat{X}$. No terminal may lie in any of the "empty regions" of X or $\hat{X}$. The BSD property must hold for each segment of X and $\hat{X}$. The MST of U computed with bottleneck Steiner distances must not be shorter than X. An SMT for U computed via a heuristic (such as the 1-Steiner heuristic of Kahng and Robins [32]) must not be shorter than X. If any of these conditions are violated, X may be discarded. Otherwise, X is retained as an FST.

Note that some of these checks can be made while recursively enumerating combinations of long leg candidates. For example if two consecutive alternating terminals delimit a segment on the backbone that defines a non-empty diamond or is longer than b_{ab} then the recursive enumeration can be cut off.

2.4.6 Rectilinear Compatibility Tests

Two FSTs F_i and F_j are incompatible if they intersect anywhere other than at a single terminal. This check is repeated for $\hat{F}_i$ and F_j, F_i and $\hat{F}_j$, and for $\hat{F}_i$ and $\hat{F}_j$. Suppose F_i and F_j meet at a single terminal and $S = (F_i \cup F_j) \cap V$ is the union of their terminals. If a heuristic finds an RSMT for S that is shorter than $|F_i| + |F_j|$, then F_i and F_j are incompatible.

Refer to Zachariasen [64] for the state of the art in rectilinear FST generation. His method considers only $O(n)$ backbone roots instead of $O(n^2)$ backbones, uses good bounds to constrain candidate choices for the long and short legs, and uses good sweep-line algorithms (instead of brute force) for checking empty regions. Using these techniques, the rectilinear FSTs for a random 1000 terminal instance are generated in less than a minute on average (compared to 3.5 hours using the Salowe-Warme algorithm).

The rest of the dissertation focuses on FST concatenation, which we solve in the next chapter by reducing it to finding a minimum-weight spanning tree in a hypergraph. For further details on FST generation, incompatibility testing and pruning, refer to [11, 16, 48, 49, 60, 62, 64].

3

The Spanning Tree in Hypergraph Problem

This chapter defines hypergraphs, their notation, and the spanning tree in hypergraph problem. It is shown that the problem of deciding even the existence of a spanning tree in an arbitrary hypergraph is NP-complete. The spanning tree in hypergraph problem is then motivated by showing that FST concatenation can be reduced to that of finding a minimum-weight spanning tree (MST) in a hypergraph. The MST in hypergraph problem is then formulated as an integer program using subtour elimination constraints. The spanning tree in hypergraph polytope $\mathrm{STHGP}(n)$ is introduced and a number of its more important properties are proven. In particular it is shown that all of the constraints used in the integer programming formulation define facets of $\mathrm{STHGP}(n)$. An alternate integer programming formulation using cutset constraints is also presented. It is shown that this formulation is "inferior" to the subtour formulation in the sense that its LP relaxation is weaker. Furthermore, it is shown that cutset constraints do not define facets of $\mathrm{STHGP}(n)$, except in the special case of the one-terminal cutsets. Finally, a simple formula that extends a classic result from graphs to hypergraphs is presented.

3.1 Definitions

The following definitions are adapted from Berge [4]. Let V be a finite set and $E \subseteq 2^V$. Then $H = (V, E)$ is a *hypergraph* if

$$|e| \geq 2 \text{ for all } e \in E \tag{3.1}$$

Normally we require only $e \neq \emptyset$ for all $e \in E$ [4] but, since our present concern is spanning trees, we assume the tighter restriction of (3.1). In keeping with graph theory we will use lower case letters to denote hyperedges — even though they are *sets*, which would normally be denoted with capital letters. We say that e is a *k-edge* of H if $e \in E$ and $|e| = k$. In a hypergraph $H = (V, E)$, a *chain of length q from v_0 to v_q* is defined to be a sequence v_0, e_1, v_1, e_2, v_2, $\ldots$, e_q, v_q such that

1. $v_0, v_1, \ldots, v_q \in V$,
2. $v_0, v_1, \ldots, v_{q-1}$ are distinct,
3. $v_1, v_2, \ldots, v_q$ are distinct,
4. $e_1, e_2, \ldots, e_q \in E$ and are distinct, and
5. $v_{i-1} \in e_i \wedge v_i \in e_i$ for $i = 1, 2, \ldots, q$.

If $q > 1$ and $v_0 = v_q$, then this chain is called a *cycle of length q*. We may omit either or both of the phrases "length q" and "from v_0 to v_q" when they are apparent or arbitrary. Hypergraph $H' = (V', E')$ is a *subhypergraph* of hypergraph $H = (V, E)$ if $V' \subseteq V$, and for every $e' \in E'$ there is an $e \in E$ such that $e' = e \cap V'$ and $|e'| \geq 2$. A hypergraph $H = (V, E)$ is *connected* if for every $s, t \in V$ there is a chain from s to t in H. A hypergraph $H = (V, E)$ is a *tree* if for every $s, t \in V$ there is a unique chain from s to t in H. Hypergraph $H' = (V, E')$ is a *spanning tree* of $H = (V, E)$ if $E' \subseteq E$ and H' is a tree.

If w is a function defined on E then for any subset $F \subseteq E$ we define $w(F) = \sum_{e \in F} w_e$. For any $S, A, B \subseteq V$, define

$$\begin{aligned}
E(S) &\equiv \{e \in E : e \subseteq S\}, \\
E(S)_k &\equiv \{e \in E : e \subseteq S \wedge |e| = k\}, \\
\delta(S) &\equiv \{e \in E : 1 \leq |e \cap S| < |e|\}, \\
\delta(S)_k &\equiv \{e \in E : 1 \leq |e \cap S| < |e| = k\}, \\
(A : B) &\equiv \{e \in E : (e \cap A \neq \emptyset) \wedge (e \cap B \neq \emptyset)\}, \\
(A : B)_k &\equiv \{e \in (A : B) : |e| = k\}.
\end{aligned}$$

We call $(S : V - S)$ a *cut* with shores S and $V - S$.

The following is an example of a hypergraph:

$$\begin{array}{lll}
H = (V, E), & & \\
V = \{a, b, c, d, e, f, g, h, i, j, k, l\}, & & \\
E = \{e_1, e_2, e_3, e_4, e_5, e_6, e_7, e_8, e_9\}, & & \\
e_1 = \{a, b, d\}, & e_4 = \{b, h\}, & e_7 = \{f, h, j\}, \\
e_2 = \{a, c, d\}, & e_5 = \{e, f, g\}, & e_8 = \{j, k, l\}, \\
e_3 = \{d, e\}, & e_6 = \{c, i\}, & e_9 = \{g, i, j\}.
\end{array}$$

This hypergraph is illustrated in Figure 3.1, where hyperedges are denoted by encircling the member vertices. Note that H is connected, but contains cycles (e.g., $f, e_5, g, e_9, j, e_7, f$). Figure 3.2 is a subhypergraph of H and also a tree — making it a spanning tree of H. Figure 3.3 is another subhypergraph of H that is not connected, and contains a cycle a, e_1, d, e_2, a. Note that if for some hypergraph $H' = (V', E')$, there are $e, f \in E'$ such that $|e \cap f| > 1$, then H' has at least one cycle and cannot be a tree.

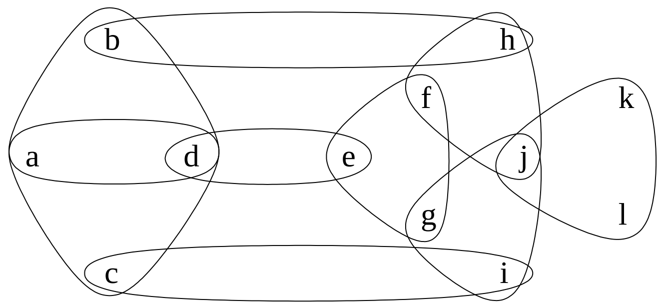


Figure 3.1: Example hypergraph H.

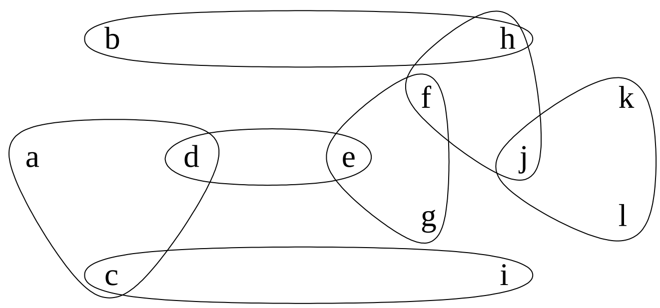


Figure 3.2: Example spanning tree of H.

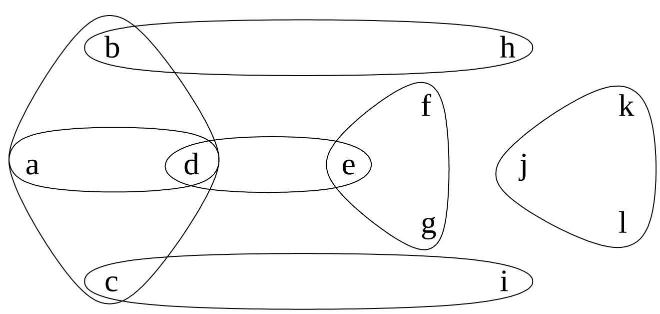


Figure 3.3: Example non-tree subhypergraph of H.

3.2 Spanning Tree in Hypergraph is NP-Complete

This section defines the spanning tree in hypergraph problem and shows that it is strongly NP-complete.

Problem Spanning Tree in Hypergraph (STHG):
Given: A hypergraph $H = (V, E)$.
Question: Is there an $E' \subseteq E$ such that $H' = (V, E')$ is a tree?

Tomescu and Zimand [55] have shown that for every $h \geq 3$, the problem of deciding the existence of a spanning tree in an h-uniform hypergraph (where $|e| = h$ for all $e \in E$) is NP-complete. Their proof uses a rather complicated reduction from 3SAT. Here is a very simple and elegant proof for the $h = 4$ case that was devised by Thomas McCormick [39]. It reduces from *exact 3 cover* which is well-known to be NP-complete [33]:

Problem Exact 3 Cover:
Given: A finite set S with $|S| = 3k$, a family F of 3-element subsets of S.
Question: Is there a subfamily $C \subset F$ that partitions S?

Theorem 3.1 *The spanning tree in hypergraph problem is NP-complete.*

Proof: Given an instance (S, F) of exact 3 cover, construct an instance $H = (S', F')$ of *spanning tree in hypergraph* as follows. Let v be an item such that $v \notin S$, let $S' = S \cup \{v\}$, and let $F' = \{e \cup \{v\} : e \in F\}$.

If C is a partition of S then the corresponding C' defines a spanning tree (S', C') of H. Conversely, let $H' = (S', C')$ be a spanning tree of H. Since every $a', b' \in C'$ both contain v, the corresponding $a, b \in F$ must be disjoint. Therefore, the C that corresponds to C' must partition S, since H' spans S'. $\square$

It follows that the corresponding optimization problem is NP-hard:

Problem Minimum Spanning Tree in Hypergraph (MSTHG):

Given: A hypergraph $H = (V, E)$, edge weights $c_e \in \mathbb{Z}^+$ for all $e \in E$.

Find: $E' \subseteq E$ that minimizes $c(E')$ such that $H' = (V, E')$ is a tree.

3.3 Reduction From FST Concatenation to MST in Hypergraph

In this section we motivate the study of the MST in hypergraph problem by showing that we can use it to solve the Steiner tree problem. A simple reduction from FST concatenation to MST in hypergraph is presented and shown to be correct.

We are given a finite set V of terminals and a corresponding set $\mathcal{F}$ of FSTs. A subset $\mathcal{F}' \subseteq \mathcal{F}$ is *non-overlapping* if $(F \cap G) \subseteq V$ for every $F, G \in \mathcal{F}'$ such that $F \neq G$. Otherwise we say that $\mathcal{F}'$ is *overlapping*. We assume that $\mathcal{F}$ contains at least one non-overlapping subset $\mathcal{F}'$ such that $T' = \cup\mathcal{F}'$ is a Steiner minimal tree for V. For any $F \in \mathcal{F}$ we define $g(F) = F \cap V$: the set of terminals spanned by F. For any $\mathcal{F}' \subseteq \mathcal{F}$ we define $g(\mathcal{F}') = \{g(F) : F \in \mathcal{F}'\}$. Let $E = g(\mathcal{F})$. We assume without loss of generality that $g(F) \neq g(G)$ for all $F, G \in \mathcal{F}$ such that $F \neq G$; if more than one FST exists for a given $S \subseteq V$, then any shortest FST spanning S can be chosen arbitrarily. Therefore $g : \mathcal{F} \mapsto E$ is an isomorphism. Let $g^{-1} : E \mapsto \mathcal{F}$ be the inverse mapping of g. For any $E' \subseteq E$ we define $g^{-1}(E') = \{g^{-1}(e) : e \in E'\}$.

Theorem 3.2 *Let V be a finite set of terminals, and $\mathcal{F}$ be a corresponding set of FSTs for V having at least one non-overlapping subset $\mathcal{F}' \subseteq \mathcal{F}$ such that $T' = \cup\mathcal{F}'$ is a Steiner minimal tree for V. Let $E = g(\mathcal{F})$, hypergraph $H = (V, E)$ and $c \in \mathbb{R}^{|E|}$ such that $c_{g(F)} = |F|$ for all $F \in \mathcal{F}$. Let $H^* = (V, E^*)$ be a spanning tree of H that minimizes $c(E^*)$, $\mathcal{F}^* = g^{-1}(E^*)$ and $T^* = \cup\mathcal{F}^*$. Then T^* is a Steiner minimal tree for V.*

Proof: Let $E' = g(\mathcal{F}')$. The Steiner minimal tree T' corresponds in a clear way to a spanning tree $H' = (V, E')$ of H. We have $|T'| = c(E')$ since the members of $\mathcal{F}'$

intersect only at terminals, which are segments of length zero. Any MST for H will therefore have weight at most $|T'|$. Let $H^* = (V, E^*)$ be any MST for H, $\mathcal{F}^* = g^{-1}(E^*)$ and $T^* = \cup E^*$. If $\mathcal{F}^*$ is overlapping then either $|T^*| < \sum_{F \in \mathcal{F}^*} |F| = c(E^*) \leq c(E') = |T'|$ (due to overlaps of non-zero length), or T^* must have a cycle, implying that $|T'| < |T^*| \leq c(E^*)$ — a contradiction in either case. Therefore, $\mathcal{F}^*$ is non-overlapping, and spanning tree H^* corresponds in a clear way to T^*, which must be a tree connecting V such that $|T^*| \leq |T'|$. Since T' is a shortest such tree, we have $|T^*| = |T'|$. □

If an FST incompatibility relation $C \subset \mathcal{F} \times \mathcal{F}$ is available, we reduce it to the corresponding relation $\hat{C} \subset E \times E$ over the hyperedges in the obvious way: Let

$$\hat{C} = \{(g(F_i), g(F_j)) : (F_i, F_j) \in C\}. \tag{3.2}$$

3.4 Integer Programming Formulation

This section presents an integer programming formulation of the minimum spanning tree in hypergraph problem. Let $H = (V, E)$ be a hypergraph, and $c \in \mathbb{R}^{|E|}$ be a vector such that c_e is the weight of edge e for all $e \in E$. Let $n = |V|$, $m = |E|$ and polytope P be the set of all $x \in \mathbb{R}^m$ that satisfy the following constraints:

$$\sum_{e \in E} (|e| - 1)\, x_e \;=\; |V| - 1, \tag{3.3}$$

$$\sum_{e \in E} \max(|e \cap S| - 1,\, 0)\, x_e \leq |S| - 1 \text{ for all } S \subseteq V \text{ with } 2 \leq |S| < |V|, \tag{3.4}$$

$$x_e \geq 0 \quad \text{for all } e \in E. \tag{3.5}$$

Theorem 3.3 *Let x be a solution to the following integer program:*

$$\min\, \{cx : x \in P \cap \mathbb{Z}^m\} \tag{3.6}$$

and let $E' = \{e \in E : x_e = 1\}$. *Then hypergraph* $H' = (V, E')$ *is a minimum spanning tree of* H.

Proof: Let $e \in E$. We first show that (3.4) and (3.5) imply $x_e \leq 1$. Choose any $S \subseteq e$ such that $|S| = 2$. Then (3.4) becomes $x_e + z \leq 1$, where z are the remaining terms which are all non-negative because of (3.5).

The integrality constraint of (3.6) together with $0 \leq x_e \leq 1$ assure that each $e \in E$ is either included in E' or not. We prove in Section 3.5 that equation (3.3) is satisfied by every spanning tree. It requires exactly the right number and size of hyperedges to guarantee that H' either has a cycle and is disconnected, or is acyclic and connected (i.e., a tree). As we also show in Section 3.5, constraints (3.4) prohibit cycles by forcing the subhypergraph induced by each subset of 2 or more vertices to be acyclic. □

3.5 The Spanning Tree in Hypergraph Polytope: STHGP(n)

This section defines the spanning tree in hypergraph polytope, STHGP(n), and proves a number of its properties. The principal goal here is to show that (3.3) is the affine hull of STHGP(n), and that (3.4) and (3.5) define facets of STHGP(n). This is important because it shows that these constraints are as tight as possible — they cannot be made any more restrictive without eliminating one or more valid spanning trees from consideration.

3.5.1 Definitions

Let $d > 0$ be an integer and $P = \{p_1, p_2, \ldots, p_k\}$ be a finite set of points in $\mathbb{R}^d$. A point x is a *linear combination of* P if

$$x = \sum_{i=1}^{k} \lambda_i \, p_i,$$

where $\lambda_i \in \mathbb{R}$ for $1 \leq i \leq k$. If $\sum_{i=1}^{k} \lambda_i = 1$, then x is also called an *affine combination of* P. If we also have $\lambda_i \geq 0$ for $1 \leq i \leq k$, then x is a *convex combination of* P. The set P

is said to be *linearly* (or *affinely*) *dependent* if there is a $p_i \in P$ that is a linear (or affine) combination of $P \setminus \{p_i\}$. Otherwise, P is *linearly* (or *affinely*) *independent*. The set of all x such that x is an affine, or convex combination of P is called the *affine hull* $\text{aff}(P)$, or *convex hull* $\text{conv}(P)$, respectively. A point $p \in P$ is said to be *extreme* if $\text{conv}(P) \neq \text{conv}(P \setminus \{p\})$.

For $0 \leq k \leq d$, a *k-flat* in $\mathbb{R}^d$ is defined as the affine hull of $k+1$ affinely independent points. A $(d-1)$-flat in $\mathbb{R}^d$ is called a *hyperplane*. A set P is said to be of *dimension* k $\dim(P) = k$ if there is a k-flat that contains P, but no $(k-1)$-flat. A k-flat therefore has dimension k. A hyperplane h may be specified as $h = \{x \in \mathbb{R}^d : a\, x = b\}$, where $a \in \mathbb{R}^d$ is a non-zero vector normal to h, and $b \in \mathbb{R}$. If $||a|| = 1$ then b is the distance (in the direction of a) from the origin to h. The set X formed by the intersection of k hyperplanes in $\mathbb{R}^d$, whose normal vectors are affinely independent is a $(d-k)$-flat. For all $p \in R^d$ and $\epsilon > 0$, define $B(p, \epsilon) = \{x \in \mathbb{R}^d : |x - p| < \epsilon\}$, that is, the open ball of radius ϵ centered at point p.

A *polyhedron* is the intersection of a finite number of linear half-spaces. A *polytope* is a polyhedron that is bounded. Alternatively, a polytope can be defined as the convex hull of a finite set of points. Let P be a polytope of dimension d. A point $p \in P$ is an *interior point* of P if there is an $\epsilon > 0$ such that $B(p, \epsilon) \subset P$. If no such ϵ exists, p is said to be a *boundary point* of P. The set of all such interior (or boundary) points is called the *interior* (or boundary) of P. There is one d-face of P, namely P itself. Let h be a hyperplane and f be the intersection of h with the boundary of P. If $\dim(f) = d-1$, we call f a $d-1$-*face* of P. In general, if f_1 and f_2 are k-faces of P and $g = f_1 \cap f_2$ is of dimension $k-1$, then g is a $k-1$-face of P. A k-face of P is itself a polytope of dimension k. We call the $d-1$-faces of P *facets*, the $d-2$-faces *ridges*, the 1-faces *edges*, and the 0-faces *vertices*, or *extreme points*.

We define hypergraph $\mathcal{K}_n = (V, E)$ such that $|V| = n$ and $E = \{e \subseteq V : |e| \geq 2\}$. Let $m = |E| = 2^n - n - 1$. To every subhypergraph $H' = (V, E')$ of $\mathcal{K}_n = (V, E)$, we associate an incidence vector $x \in \{0, 1\}^m$ defined by $x_e = 1$ if $e \in E'$ and 0 otherwise. Let $\text{ST}_n \subset \{0, 1\}^m$ denote the set of incidence vectors of spanning trees of $\mathcal{K}_n$.

Define $\mathrm{STHGP}(n) = \mathrm{conv}(\mathrm{ST}_n)$.

3.5.2 Dimensionality of STHGP(n)

Theorem 3.4 *Let $n \geq 2$, $(V, E) = \mathcal{K}_n$ and $x \in \mathrm{ST}_n$, then*

$$\sum_{e \in E} (|e| - 1)\, x_e = |V| - 1. \tag{3.7}$$

Proof: Let $T = (V, E')$ be the hypergraph corresponding to x. Then

$$\sum_{e \in E} (|e| - 1) x_e = \sum_{e \in E'} (|e| - 1).$$

From [4] we know that a hypergraph (V, E') is acyclic if, and only if,

$$\sum_{e \in E'} (|e| - 1) = |V| - p, \tag{3.8}$$

where p is the number of connected components. Since spanning trees are both connected and acyclic, we have $p = 1$. $\square$

Remark: Equation (3.8) can be shown directly by simple induction on the hyperedges. The induction step is analogous to a single iterative step of Kruskal's algorithm for the minimum spanning tree [35].

Theorem 3.4 gives a linear equation satisfied by all $x \in \mathrm{ST}_n$. We now show there are no other such linear equations. To do this, we will need two lemmas.

Lemma 3.1 *Let $T^1 = (V, E_1 \cup E_2)$ and $T^2 = (V, E_1 \cup E_3)$ be two spanning trees with corresponding incidence vectors x^1 and x^2, such that E_1 is disjoint from $E_2 \cup E_3$. If x^1 and x^2 both satisfy a linear equation $c\,x = b$, then*

$$\sum_{e \in E_2} c_e = \sum_{e \in E_3} c_e. \tag{3.9}$$

Proof:

$$
\begin{aligned}
& c\,x^1 = b \text{ and } c\,x^2 = b \\
\Longrightarrow \quad & c\,x^1 = c\,x^2 \\
\Longrightarrow \quad & \sum_{e \in E_1 \cup E_2} c_e = \sum_{e \in E_1 \cup E_3} c_e \\
\Longrightarrow \quad & \sum_{e \in E_2} c_e = \sum_{e \in E_3} c_e,
\end{aligned}
$$

since E_1 is disjoint from both E_2 and E_3. □

Lemma 3.2 *Let $n \geq 3$ and $(V, E) = \mathcal{K}_n$. If $c\,x = b$ is any linear equation satisfied by every $x \in \mathrm{ST}_n$ then there is an α such that $c_e = \alpha(|e| - 1)$ for every $e \in E$, and $b = \alpha(n-1)$.*

Proof: Let c and b be as stated. Let $e_1, e_2 \in E(V)_2$. We first show that $c_{e_1} = c_{e_2}$. Let $S \subseteq V$ be a cut of V crossed by both e_1 and e_2 (i.e., $e_1, e_2 \in (S : V - S)$). A suitable cut S always exists when $n \geq 3$. Construct spanning trees $S_1 = (S, E_1)$ and $S_2 = (V - S, E_2)$ for each side of the cut using only 2-edges. We can now construct spanning trees $T^1 = (V, E_1 \cup \{e_1\} \cup E_2)$ and $T^2 = (V, E_1 \cup \{e_2\} \cup E_2)$ for V having incidence vectors x^1 and x^2, respectively. By Lemma 3.1 we must have $c_{e_1} = c_{e_2}$. Let $\alpha = c_{e_1}$. Certainly $c_e = \alpha(|e| - 1)$ holds for every 2-edge $e \in E$. We deduce that $b = \alpha(n-1)$ by noting that we can construct spanning trees for V entirely out of $(n-1)$ 2-edges.

Let $T^3 = (V, E_3)$ be a spanning tree, $e \in E_3$ and let $k = |e|$. Let x^3 be the incidence vector of T^3. We can construct a new spanning tree T^4 by replacing e with any spanning tree constructed using only $(k-1)$ 2-edges from $E(e)_2$. Let x^4 be the incidence vector of T^4, then by Lemma 3.1 we have $c_e = (k-1)\alpha = (|e| - 1)\alpha$, which completes the proof. □

Theorem 3.5

$$\dim(\mathrm{STHGP}(n)) = 2^n - n - 2 \quad \textit{for } n \geq 2. \tag{3.10}$$

Proof: For $n = 2$, $|\mathrm{ST}_2| = 1$ so that STHGP(2) is a single point with dimension 0 and the theorem holds. Now suppose $n \geq 3$. Theorem 3.4 gives one linear equation satisfied by every $x \in \mathrm{ST}_n$ and lemma 3.2 shows that there are no other such linear equations. Therefore $\dim(\mathrm{STHGP}(n)) = m - 1 = 2^n - n - 2$. □

Corollary 3.5.1 *Let h be the hyperplane satisfying (3.7). Then $h = \mathrm{aff}(\mathrm{STHGP}(n))$.*

Theorem 3.6 *Every $x \in \mathrm{ST}_n$ is an extreme point of* $\mathrm{STHGP}(n)$.

This is clearly true of any $x \in X$ and polytope $P = \mathrm{conv}(X)$, where X is any subset of vertices of the hypercube.

Corollary 3.6.1 *If $x \in \mathrm{ST}_n$ then x cannot be expressed as a convex combination of the elements of* $\mathrm{ST}_n \setminus \{x\}$.

3.5.3 Non-Negativity Constraints are Facet-Defining

To prove that the non-negativity constraints (3.5) are facet-defining, we will need two lemmas:

Lemma 3.3 *Let $n \geq 4$, $(V, E) = \mathcal{K}_n$ and let $e, e_1, e_2 \in E$ be distinct edges such that $|e_1| = |e_2| = 2$. Then there is an $S \subset V$ such that $1 \leq |S| < n$ with the following properties:*

1. *$e_1, e_2 \in (S : V - S)$,*

2. *There exist spanning trees $S_1 = (S, E_1)$ and $S_2 = (V - S, E_2)$ such that $E_1 \subseteq E(S)_2$ and $E_2 \subseteq E(V - S)_2$,*

3. *$e \notin E_1$ and $e \notin E_2$.*

We omit the details of the proof, except to note that if $|e| \geq 3$ then property (iii) is automatically satisfied and we can assume without loss of generality that $|e| = 2$. The rest of the proof follows by case analysis for $n = 4$ and by induction for $n \geq 5$.

Lemma 3.4 *Let $n \geq 4$, $(V, E) = \mathcal{K}_n$, $e \in E$ and let $F = \{x \in \mathrm{ST}_n : x_e = 0\}$. If $cx = b$ is any linear equation satisfied by every $x \in F$ then there exists an α such that $b = \alpha(n-1)$ and $c_{e'} = \alpha(|e'| - 1)$ for all $e' \in E$, $e' \neq e$.*

Proof: Let e_1 and e_2 be 2-edges distinct from e. By Lemma 3.3 there is a cut $(S : V - S)$, $1 \leq |S| < |V|$ crossed by both e_1 and e_2. Also by Lemma 3.3 there exist spanning trees $S_1 = (S, E_1)$ and $S_2 = (V - S, E_2)$ for S and $V - S$ respectively, such that $e \notin E_1$ and $e \notin E_2$. Then $T^1 = (V, E_1 \cup \{e_1\} \cup E_2)$ and $T^2 = (V, E_1 \cup \{e_2\} \cup E_2)$ are spanning trees for V that do not contain edge e. Let x^1 and x^2 be the incidence vectors corresponding to T^1 and T^2, respectively. We have $x^1, x^2 \in F$ since $x^1_e = x^2_e = 0$ by construction, and so $cx^1 = b$ and $cx^2 = b$. By Lemma 3.1 we have $c_{e_1} = c_{e_2}$, so every 2-edge $e' \neq e$ therefore has the same coefficient $c_{e'} = \alpha$.

Let $x^3 \in F$ and $T^3 = (V, E_3)$ be its corresponding spanning tree. Let $e' \in E_3$ and $k = |e'|$. Since $x^3 \in F$ we know $e' \neq e$. Construct a new spanning tree T^4 by replacing edge e' with a spanning tree constructed using only 2-edges ($k - 1$ of them) from $E(e')_2 \setminus \{e\}$. Let x^4 be the incidence vector of T^4. We have $x^3 \in F$ and $x^4 \in F$ by construction. By Lemma 3.1 we have $c_{e'} = \alpha(k-1)$. Therefore $c_{e'} = \alpha(|e'| - 1)$ for all $e' \in E$, $e' \neq e$. We must have $b = (n-1)\alpha$, since a spanning tree for V can be always be constructed using exactly $n - 1$ 2-edges from $E(V)_2 \setminus \{e\}$. $\square$

Theorem 3.7 *Let $n \geq 4$, $(V, E) = \mathcal{K}_n$ and let $e \in E$. Then the inequality $x_e \geq 0$ defines a facet of* STHGP(n).

Proof: First note that $x_e \geq 0$ is satisfied by every $x \in \mathrm{ST}_n$ and is therefore a valid inequality. Let $F = \{x \in \mathrm{ST}_n : x_e = 0\}$ and let $cx = b$ be any linear equation that is satisfied by every $x \in F$. By Lemma 3.4 we know that equation $cx = b$ is such that $b = \alpha(n-1)$ and $c_{e'} = \alpha(|e'| - 1)$ for all $e' \in E$, $e' \neq e$. Equation $cx = b$ can therefore be obtained by taking α times Equation (3.7) plus $c_e - \alpha(|e| - 1)$ times equation $x_e = 0$.

The set F therefore has dimension $m-2 = \dim(\mathrm{STHGP}(n)) - 1$, proving that $x_e \geq 0$ is facet-defining. $\square$

3.5.4 Subtour Constraints are Facet-Defining

In order to prove that the subtour elimination constraints (3.4) are facet-defining, we need two lemmas.

Lemma 3.5 *Let $n \geq 3$, $(V,E) = \mathcal{K}_n$ and let $S \subset V$ such that $|S| \geq 2$. Also, let $F = \{x \in \mathrm{ST}_n : \sum_{e \in E} \max(|e \cap S| - 1, 0)x_e = |S| - 1\}$ and $cx = b$ be any linear equation satisfied by every $x \in F$. Then there exist α and β such that:*

1. *$c_e = \alpha$ for all $e \in E(S)_2$,*
2. *$c_e = \beta$ for all $e \in E(V)_2 \setminus E(S)_2$.*

Proof: We note that $E(V)_2 \setminus E(S)_2 = \delta(S)_2 \cup E(V-S)_2$. For part 2 it therefore suffices to show: (a) $e_1, e_2 \in \delta(S)_2 \Longrightarrow c_{e_1} = c_{e_2}$; (b) $e_1, e_2 \in E(V-S)_2 \Longrightarrow c_{e_1} = c_{e_2}$; and (c) that there is a $e_1 \in \delta(S)_2$ and a $e_2 \in E(V-S)_2$ such that $c_{e_1} = c_{e_2}$. Part 2 then follows by transitivity of equality. We prove each of the 4 resulting cases by obtaining trees T^1 and T^2 with corresponding incidence vectors x^1 and x^2 such that $x^1, x^2 \in F$ and that differ only by substituting edge e_1 for e_2 or vice versa. Then by Lemma 3.1 we have $c_{e_1} = c_{e_2}$.

Case 1: Let $e_1, e_2 \in E(S)_2$. The $|S| = 2$ case is trivial since there is only one such edge. Otherwise $|S| \geq 3$ and there is a cut $U \subseteq S$ such that $e_1, e_2 \in (U : S-U)_2$. Let $S^1 = (U, E_1)$, $S^2 = (S-U, E_2)$ and $S^3 = (V-S, E_3)$ be spanning trees such that $E_1 \subseteq E(U)_2$, $E_2 \subseteq E(S-U)_2$ and $E_3 \subseteq E(V-S)_2$. Let $e_3 \in (U : V-S)_2$. Then $T^1 = (V, E_1 \cup \{e_1\} \cup E_2 \cup \{e_3\} \cup E_3)$ and $T^2 = (V, E_1 \cup \{e_2\} \cup E_2 \cup \{e_3\} \cup E_3)$ are spanning trees with the necessary properties. See Figure 3.4.

Case 2a: Let $e_1, e_2 \in \delta(S)_2$ and let $S_1 = (S, E_1)$ and $S^2 = (V-S, E_2)$ be spanning trees such that $E_1 \subseteq E(S)_2$ and $E_2 \subseteq E(V-S)_2$. Then $T^1 = (V, E_1 \cup \{e_1\} \cup E_2)$ and $T^2 = (V, E_1 \cup \{e_2\} \cup E_2)$ are spanning trees with the necessary properties. See Figure 3.5.

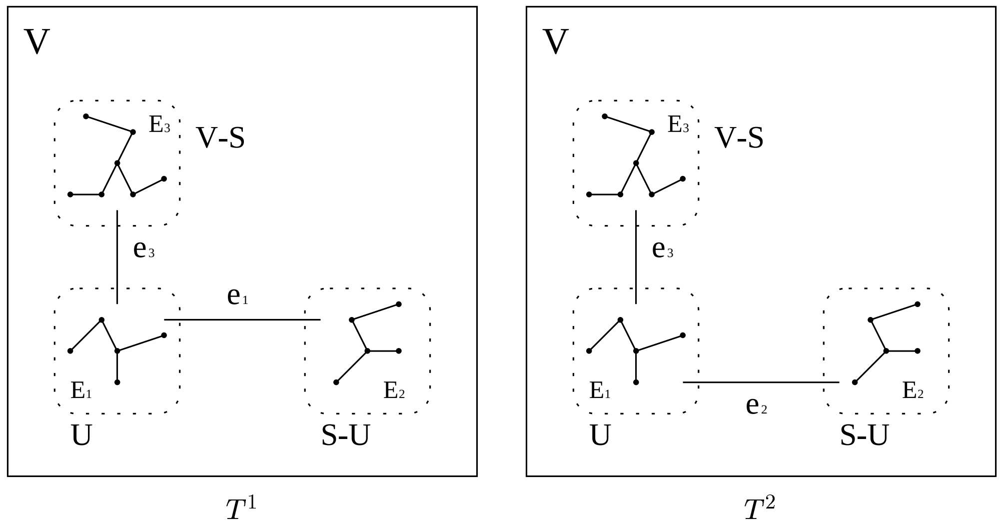


Figure 3.4: Case 1 for proof of Lemma 3.5

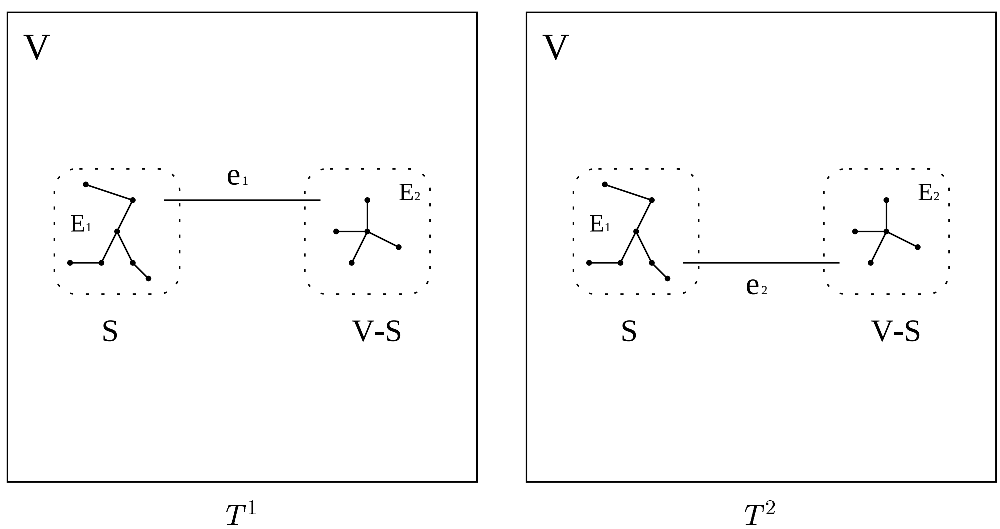
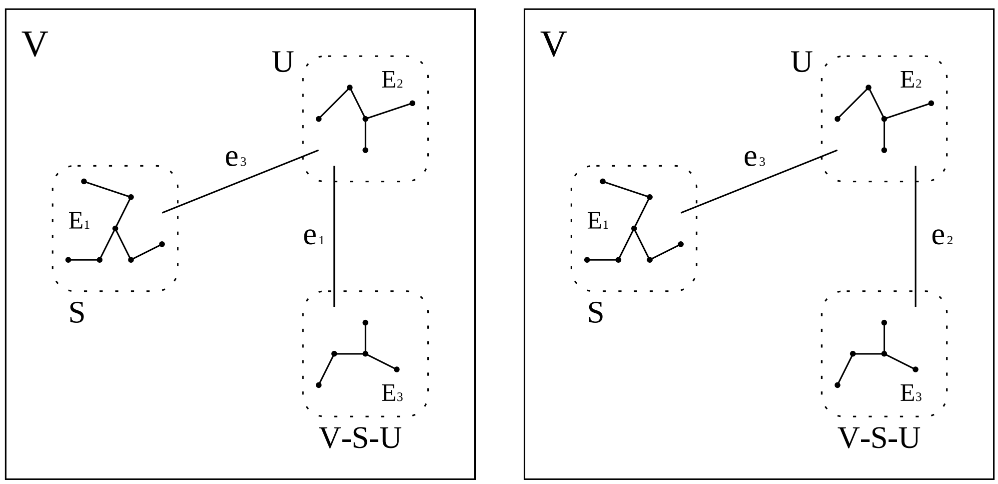


Figure 3.5: Case 2 for proof of Lemma 3.5

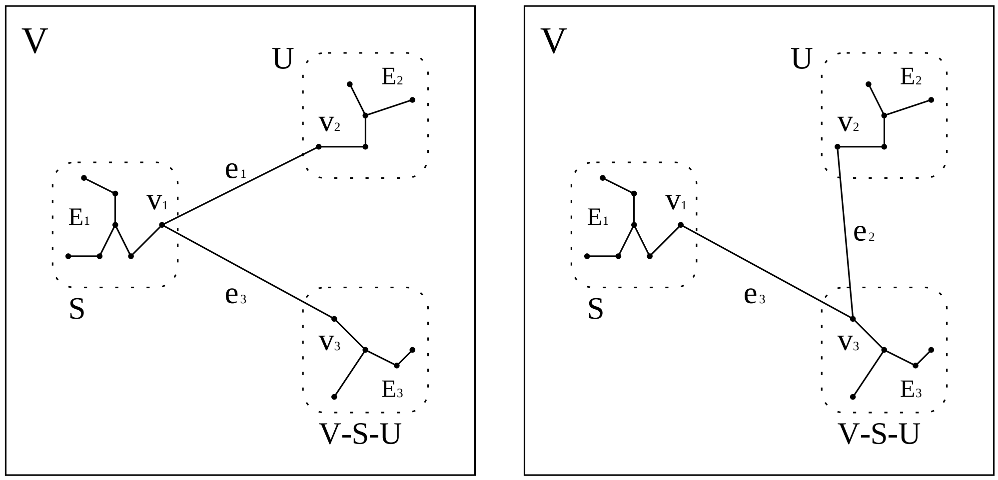
Case 2b: Let $e_1, e_2 \in E(V-S)_2$. The $|V-S| \leq 2$ case is trivial since there is at most one such edge. Otherwise $|V-S| \geq 3$ and there is a cut $U \subseteq (V-S)$ such that $e_1, e_2 \in (U : V-S-U)_2$. Let $S^1 = (S, E_1)$, $S^2 = (U, E_2)$ and $S^3 = (V-S-U, E_3)$ be spanning trees such that $E_1 \subseteq E(S)_2$, $E_2 \subseteq E(U)_2$ and $E_3 \subseteq E(V-S-U)_2$. Let $e_3 \in (S : U)_2$. Then $T^1 = (V, E_1 \cup \{e_3\} \cup E_2 \cup \{e_1\} \cup E_3)$ and $T^2 = (V, E_1 \cup \{e_3\} \cup E_2 \cup \{e_2\} \cup E_3)$ are spanning trees with the necessary properties. See Figure 3.6.



T^1 T^2

Figure 3.6: Case 3 for proof of Lemma 3.5

Case 2c: If $|V-S| \leq 1$ then $E(V-S)_2$ is empty and the theorem is proved. Otherwise let $v_1 \in S$ and let $v_2, v_3 \in V-S$ be distinct vertices. Let $e_1 = \{v_1, v_2\}$, $e_2 = \{v_2, v_3\}$ and $e_3 = \{v_1, v_3\}$. Let $U \subseteq V-S$ by any cut such that $e_2 \in (U : V-S-U)_2$ and $v_2 \in U$. Let $S^1 = (S, E_1)$, $S^2 = (U, E_2)$ and $S^3 = (V-S-U, E_3)$ be spanning trees such that $E_1 \subseteq E(S)_2$, $E_2 \subseteq E(U)_2$ and $E_3 \subseteq E(V-S-U)_2$. Then $T^1 = (V, E_1 \cup E_2 \cup E_3 \cup \{e_3\} \cup \{e_1\})$ and $T^2 = (V, E_1 \cup E_2 \cup E_3 \cup \{e_3\} \cup \{e_2\})$ are spanning trees with the necessary properties. See Figure 3.7. $\square$



T^1 T^2

Figure 3.7: Case 4 for proof of Lemma 3.5

Lemma 3.6 *Let $n \geq 3$, $(V, E) = \mathcal{K}_n$ and let $S \subset V$ such that $|S| \geq 2$. Let*

$$F = \{x \in \mathrm{ST}_n : \sum_{e \in E} \max(|e \cap S| - 1,\, 0)x_e = |S| - 1\}$$

and let $cx = b$ be any linear equation satisfied by every $x \in F$. Then there exist α and β such that

$$b = \alpha(|S| - 1) + \beta(|V| - |S|)$$

and

$$c_e = \alpha \max(|e \cap S| - 1,\, 0) + \beta(|e| - 1 - \max(|e \cap S| - 1,\, 0))$$

for all $e \in E$.

Proof: By Lemma 3.5, $c_e = \alpha$ for every 2-edge $e \in E(S)_2$ and $c_e = \beta$ for every 2-edge $e \in E(V)_2 \setminus E(S)_2$. Let $x^1 \in F$ so that $cx^1 = b$, let $T^1 = (V, E_1)$ be the hypergraph corresponding to x^1 and let $e \in E_1$ be any edge of this tree. Let $k = |e|$ and $j = \max(|e \cap S| - 1, 0)$.

Now e can be replaced with a spanning tree constructed of 2-edges from $E(e)_2$, taking j of these 2-edges from $E(e \cap S)_2$ and the other $k-1-j$ 2-edges from $E(e)_2 \setminus E(S)_2$. The result will be a spanning tree T^2 with incidence vector x^2. We have $x^2 \in F$ by construction so that $cx^2 = b$. Edge e was replaced by j 2-edges of weight α and $k-1-j$ edges of weight β and so by Lemma 3.1 we have $c_e = j\alpha + (k-1-j)\beta$. Substituting j and k back in gives $c_e = \alpha \max(|e \cap S| - 1, 0) + \beta(|e| - 1 - \max(|e| \cap S| - 1, 0))$.

If we reduce all edges to 2-edges in this fashion, we will have exactly $|S|-1$ 2-edges in $E(S)_2$ and exactly $|V| - |S|$ 2-edges in $E(V)_2 \setminus E(S)_2$. We must therefore have

$$b = \alpha(|S| - 1) + \beta(|V| - |S|).$$

$\square$

Theorem 3.8 *Let $n \geq 3$, $(V, E) = \mathcal{K}_n$ and let $S \subseteq V$ such that $2 \leq |S| < n$. Then the inequality*

$$\sum_{e \in E} \max(|e \cap S| - 1,\, 0)\, x_e \leq |S| - 1 \tag{3.11}$$

defines a facet of STHGP(n).

Proof: First note that (3.11) is a valid inequality, since if

$$\sum_{e \in E} \max(|e \cap S| - 1, 0)\, x_e > |S| - 1$$

we have a cycle residing entirely within S, a contradiction since every spanning tree $x \in \mathrm{ST}_n$ is acyclic. Let F be the set of all $x \in \mathrm{ST}_n$ that satisfy the linear equation

$$\sum_{e \in E} \max(|e \cap S| - 1,\, 0)\, x_e = |S| - 1. \tag{3.12}$$

Let $cx = b$ be any linear equation that is satisfied by every $x \in F$. By Lemma 3.6 we know that equation $cx = b$ can be written in the form: $b = \alpha(|S| - 1) + \beta(|V| - |S|)$ and $c_e = \alpha \max(|e \cap S| - 1,\, 0) + \beta(|e| - 1 - \max(|e \cap S| - 1,\, 0))$ for all $e \in E$. We can therefore obtain this equation by taking β times equation (3.7) plus $(\alpha - \beta)$ times equation (3.12). The set F therefore has dimension $m - 2 = \dim(\mathrm{STHGP}(n)) - 1$, proving that inequality (3.11) is facet-defining. $\square$

Remark: All of the preceding proofs remain valid for any hypergraph $H = (V, E)$ containing all 2-edges. In this case $m = |E|$ and the resulting polytope has dimension $m - 1$.

3.5.5 Cutsets are Weaker than Subtours

This section presents an alternate integer programming formulation based on cutset constraints and shows that its LP relaxation is weaker than the formulation based on subtour elimination. It also shows that cutset constraints do not define facets of STHGP(n) except in the special case of single-terminal cutsets — in which case they are equivalent to the $n-1$-terminal subtour constraints.

Let $n \geq 2$ and hypergraph $H = (V, E) = \mathcal{K}_n$. We assume for the sake of concreteness that $V = \{0, 1, \ldots, n-1\}$. If for example $e = \{1, 3, 5\}$ we shall denote x_e concisely as x_{135}. We define STP(n), the subtour polytope, to be those points satisfying (3.3), (3.4) and (3.5). We define CSP(n), the cutset polytope, to be those points satisfied by (3.3), (3.5) and

$$\sum_{e \in (S:V-S)} x_i \geq 1 \quad \text{for all } S \subset V \text{ such that } |S| \geq 1. \tag{3.13}$$

Theorem 3.9 *Let $x \in \mathrm{CSP}(n) \cap \mathbb{Z}^m$ and $E' = \{e \in E : x_e = 1\}$. Then $H' = (V, E')$ is a spanning tree of H.*

Proof: Let $e \in E$. We first show that $x_e \leq 1$ is implied by the other constraints. Let $\bar{E} = \{e \in E : x_e \geq 2\}$. We can subtract $(|e|-1)(x_e-1)$ from both sides of (3.3) for all $e \in \bar{E}$. Comparing the right hand side with (3.8) to conclude that $p \geq 2$ and so there must be at least 2 connected components in H'. This implies that at least one of the constraints (3.13) is violated, a contradiction. So we infer that $x_e \in \{0, 1\}$ and each edge e is either selected in

E' or not. The cutset constraints (3.13) imply that H' is connected. Equation (3.3) implies that H' is also acyclic, since it represents (3.8) with $p = 1$. Since H' is both connected and acyclic, it is a tree. □

For any $S \subseteq V$, define $h(S) = \sum_{e \in E} \max(|e \cap S| - 1, 0) x_e$. We now show that every cutset constraint is a sum of two subtour constraints.

Theorem 3.10 *For all* $S \subseteq V$ *such that* $0 < |S| < |V|$,

$$x(S : V - S) - 1 = [|S| - 1 - h(S)] + [|V - S| - 1 - h(V - S)]\,. \tag{3.14}$$

Proof: From (3.3) we have

$$\begin{aligned} & |V| - 1 = \sum_{e \in E} (|e| - 1) x_e = h(S) + x(S : V - S) + h(V - S) \\ \Longrightarrow \quad & x(S : V - S) = |S| + |V - S| - 1 - h(S) - h(V - S) \\ \Longrightarrow \quad & x(S : V - S) = [|S| - 1 - h(S)] + [|V - S| - 1 - h(V - S)]\,. \end{aligned}$$

□

We are now ready to prove the main result — that the LP relaxation of the cutset formulation is weaker than the LP relaxation of the subtour formulation.

Theorem 3.11 *For* $n \geq 4$,

$$\mathrm{STHGP}(n) \subset \mathrm{STP}(n) \subset \mathrm{CSP}(n) \tag{3.15}$$

Proof: Every constraint of STP(n) is a facet of STHGP(n), implying that

$$\mathrm{STHGP}(n) \subseteq \mathrm{STP}(n).$$

Let $n = 4$, $(V, E) = \mathcal{K}_n$, and consider that point $\bar{x} \in \mathbb{R}^{|E|}$ whose only non-zero components are

$$\bar{x}_{012} = \bar{x}_{013} = \bar{x}_{023} = \bar{x}_{0123} = \frac{1}{3}.$$

Then $\bar{x} \in \text{STP}(4)$ but $\bar{x} \notin \text{STHGP}(4)$ since $x_{012} + x_{013} + x_{023} + x_{123} + x_{0123} \leq 1$ is a valid inequality for STHGP(4) that is violated by $\bar{x}$. (This inequality actually defines a facet of STHGP(4).) By adding additional 2-edges of weight 1, we can embed this example for any $n \geq 4$. Therefore $\text{STHGP}(n) \subset \text{STP}(n)$ for $n \geq 4$.

Suppose that $\emptyset \subset S \subset V$ such that $x(S : V - S) < 1$. Then the left hand side of equation (3.14) is negative, implying at least one of $h(S) > |S| - 1$ or $h(V - S) > |V - S| - 1$ is true. Therefore, any violation of (3.13) implies at least one violation of (3.4). This implies that $\text{STP}(n) \subseteq \text{CSP}(n)$. Let $n = 4$ and consider the solution $\bar{y}$ whose only non-zero components are $\bar{y}_{01} = 1$ and $\bar{y}_{012} = \bar{y}_{13} = \bar{y}_{23} = 1/2$. We have $\bar{y} \in \text{CSP}(4)$ but $\bar{y} \notin \text{STP}(4)$ since subtour $S = \{0, 1\}$ is violated by $\bar{y}$. This implies $\text{STP}(4) \subset \text{CSP}(4)$. By adding additional 2-edges of weight 1, we can embed this example for any $n \geq 4$. Therefore $\text{STP}(n) \subset \text{CSP}(n)$ for $n \geq 4$. □

Finally, we show that the cutset constraints do *not* define facets of STHGP(n), except in one special case.

Theorem 3.12 *Let $n \geq 3$, and $S \subset V$ such that $0 < |S| < n$. Then the cutset constraint*

$$\sum_{e \in (S:V-S)} x_e \geq 1 \tag{3.16}$$

defines a facet of STHGP(n) *if and only if $|S| = 1$ or $|V - S| = 1$.*

Proof: From (3.14) we deduce that

$$x(S : V - S) - 1$$

is non-negative if and only if

$$[|S| - 1 - h(S)] + [|V - S| - 1 - h(V - S)]$$

is non-negative. If $|S| = 1$ then $|S| - 1 - h(S) = 0$ and (3.16) is equivalent to subtour $V - S$. Similarly, if $|V - S| = 1$, then $|V - S| - 1 - h(V - S) = 0$ and (3.16) is equivalent to subtour S. In all other cases, however, constraint (3.16) is the sum of two inequalities that define distinct facets. Consider the hyperplanes H_S and H_{V-S} consisting of those points that satisfy $h(S) = |S| - 1$ and $h(V - S) = |V - S| - 1$, respectively. A convex combination of H_S and H_{V-S} is equivalent to a rotation of H_S about its intersection with H_{V-S}. This intersection is a *ridge* of STHGP(n), not a facet — its dimensionality is too small by 1 to be a facet. □

3.6 Counting the Spanning Trees of $\mathcal{K}_n$

We now turn to the question of how many distinct labeled spanning trees there are in $\mathcal{K}_n$, the complete hypergraph on n vertices. This is equivalent to the number of extreme points of STHGP(n). The results of this section are part of work done in collaboration with W. D. Smith. A forthcoming paper by Smith and Warme will present these and other enumeration results for hypertrees, including simple combinatorial proofs of Theorem 3.15 and Corollary 3.15.1 based on a generalization of the Prüfer code [37, 44].

For the analogous problem in conventional graphs the classical result is n^{n-2}, and is usually attributed to Cayley in 1889 [8]. Cayley's own paper, however, references an earlier proof of this formula by Borchardt [6] in 1860. We now present the analogous result for spanning trees in the complete hypergraph (i.e., hypertrees).

For $n \geq 1$, let h_n be the number of *rooted* hypertrees spanning n labeled vertices. A rooted hypertree is a hypertree in which one particular vertex is identified as being the *root*. The desired result for unrooted hypertrees is then h_n/n. Considering rooted hypertrees breaks up the symmetry of the problem and avoids various automorphisms that would otherwise result.

Let $\left\{ {n \atop k} \right\}$ denote the Stirling numbers of the second kind (i.e., the number of ways of partitioning n items into k non-empty subsets). They can be defined by the following recurrence:

$$\begin{aligned}
\left\{ {0 \atop 0} \right\} &= 1 \\
\left\{ {n \atop 0} \right\} &= 0 \quad \text{for } n \geq 1 \\
\left\{ {n \atop k} \right\} &= 0 \quad \text{for } n < k \\
\left\{ {n \atop k} \right\} &= k \left\{ {n-1 \atop k} \right\} + \left\{ {n-1 \atop k-1} \right\} \quad \text{for } 1 \leq k \leq n
\end{aligned}$$

For $k > 0$, let Bell(k) be the kth Bell number (Bell(k) is the number of ways of partitioning k items into non-empty subsets). The Bell numbers can be expressed in terms of the Stirling numbers:

$$\mathrm{Bell}(n) = \sum_{k=1}^{n} \left\{ {n \atop k} \right\}.$$

Recently, W. D. Smith [51] obtained the following recurrence and generating function for h_n:

Theorem 3.13 (W. D. Smith [51]) *Let h_n be the number of rooted hypertrees spanning n labeled vertices. Then $h_1 = 1$, and for $n > 1$*

$$h_n = n \sum_{k>0} \frac{\mathrm{Bell}(k)}{k!} \sum_{\substack{a_j > 0 \\ \sum_{j=1}^{k} a_j = n-1}} \binom{n-1}{a_1, a_2, \ldots, a_k} \prod_{j=1}^{k} h_{a_j}. \tag{3.17}$$

Proof: The base case is obvious, so assume $n > 1$. Select a unique root vertex (there are n possible choices). Now delete the root vertex and every hyperedge incident to the root. All that remains are the individual subhypertrees of the root node, containing a total of $n-1$ vertices. Each of these subhypertrees is itself a rooted hypertree, the root vertex being the one that was incident to a deleted hyperedge. Suppose there are k of these rooted subhypertrees. The vector $a_1, a_2, \ldots, a_k$ indicates how many vertices are in each of the k subhypertrees. We divide by $k!$ since the particular ordering of the subhypertrees does not matter. For each such vector there are

$$\binom{n-1}{a_1, a_2, \ldots, a_k}$$

ways of partitioning the $n-1$ vertices into k non-empty subsets of sizes $a_1, a_2, \ldots, a_k$. For each subset j of a_j vertices, there are h_{a_j} distinct rooted subhypertrees. Each of the Bell(k) partitions of the k subhypertrees represents a distinct way of hooking the subhypertrees to the root using hyperedges. Let $S_1, S_2, \ldots, S_j$ be such a partition. Then the k subhypertrees are connected to the root using j hyperedges. The hyperedge for S_i consists of *the* root together with the root vertices of each subhypertree in S_i. $\square$

Remark: Replacing Bell(k) with 1 in (3.17) gives a recurrence for conventional rooted trees.

Let $f(z)$ be a series in powers of z. Then $[z^n]\ f(z)$ denotes the coefficient of z^n in the series $f(z)$. If λ is any non-zero real number, then $[z^n/\lambda]\ f(z)$ denotes $\lambda\ [z^n]\ f(z)$. Let

$$H(z) = \sum_{n \geq 1} h_n \frac{z^n}{n!} \tag{3.18}$$

be the exponential generating function for h_n.

Theorem 3.14 (W. D. Smith [51])

$$H(z) = z\ e^{e^{H(z)}-1}. \tag{3.19}$$

Proof: It just so happens that

$$\sum_{\substack{a_j > 0 \\ \sum_{j=1}^{k} a_j = n-1}} \binom{n-1}{a_1, a_2, \ldots, a_k} \prod_{j=1}^{k} h_{a_j} = \left[\frac{z^{n-1}}{(n-1)!}\right]\ H(z)^k. \tag{3.20}$$

Therefore, if $n > 1$ we have

$$h_n = \left[\frac{z^{n-1}}{(n-1)!}\right]\ n \sum_{k>0} \text{Bell}(k) \frac{H(z)^k}{k!} \tag{3.21}$$

Note that

$$1 + \sum_{k \geq 1} \mathrm{Bell}(k) \frac{z^k}{k!} = e^{e^z - 1} \tag{3.22}$$

is known (e.g., equation 24f, page 34 of [53]). Substituting into (3.21) yields:

$$\frac{h_n}{n!} = [z^{n-1}]\ (1 + \sum_{k \geq 1} \mathrm{Bell}(k) \frac{H(z)^k}{k!}) = [z^{n-1}]\ e^{e^{H(z)} - 1} = [z^n]\ z\ e^{e^{H(z)} - 1} \tag{3.23}$$

which happens to hold at $n = 1$ as well as for $n > 1$. Therefore

$$H(z) = \sum_{n > 0} h_n\ \frac{z^n}{n!} = z\ e^{e^{H(z)} - 1}. \tag{3.24}$$

□

To obtain a closed form for h_n, we employ the Lagrange inversion formula [59], a weak form of which (sufficient for our purposes) is

Lemma 3.7 (Lagrange inversion formula) *Let $\theta(u)$ be a formal power series in u, such that $\theta(0) = 1$. Then there is a unique formal power series $u(z)$ (about $z = 0$) satisfying*

$$u(z) = z\ \theta(u(z)). \tag{3.25}$$

This formal power series satisfies

$$[z^n]\ u(z) = \frac{1}{n}\ [u^{n-1}]\ \{\theta(u)^n\}. \tag{3.26}$$

A proof can be found in [59].

Theorem 3.15 (Warme) *Let h_n be the number of rooted hypertrees spanning n labeled vertices. Then for every $n \geq 1$*

$$h_n = \sum_{i=0}^{n-1} \left\{ {n-1 \atop i} \right\} n^i. \tag{3.27}$$

Proof: Apply the Lagrange inversion formula (Lemma 3.7) to (3.19) with $\theta(u) = e^{e^u - 1}$:

$$
\begin{aligned}
\frac{h_n}{n!} &= [z^n]\ H(z) \\
&= \frac{1}{n}\ [u^{n-1}]\ \theta(u)^n \\
&= \frac{1}{n}\ [u^{n-1}]\ e^{n(e^u-1)} \\
&= \frac{1}{n}\ [u^{n-1}]\ \sum_{i\geq 0} \frac{n^i\ (e^u-1)^i}{i!} \\
&= \frac{1}{n}\ [u^{n-1}]\ \sum_{i\geq 0} \frac{n^i}{i!}\ \sum_{j=0}^{i} \binom{i}{j} e^{ju}\ (-1)^{i-j} \\
&= \frac{1}{n}\ [u^{n-1}]\ \sum_{i\geq 0} \frac{n^i}{i!}\ \sum_{j=0}^{i} \binom{i}{j} (-1)^{i-j} \sum_{k\geq 0} \frac{j^k\ u^k}{k!} \\
&= \frac{1}{n}\ [u^{n-1}]\ \sum_{k\geq 0} \left[\sum_{i\geq 0} \frac{n^i}{i!}\ \sum_{j=0}^{i} \binom{i}{j} (-1)^{i-j} \frac{j^k}{k!} \right] u^k \\
&= \frac{1}{n} \sum_{i\geq 0} \frac{n^i}{i!}\ \sum_{j=0}^{i} \binom{i}{j} (-1)^{i-j} \frac{j^{n-1}}{(n-1)!} \\
&= \frac{1}{n!} \sum_{i\geq 0} \frac{n^i}{i!}\ \sum_{j=0}^{i} \binom{i}{j} j^{n-1} (-1)^{i-j}
\end{aligned}
$$

It is known that

$$
i! \left\{ {n \atop i} \right\} = \sum_{j} \binom{i}{j}\ j^n\ (-1)^{i-j}
$$

(See, for example equation (6.19) from [22]). Performing this substitution yields

$$
\frac{h_n}{n!} = \frac{1}{n!} \sum_{i\geq 0} \left\{ {n-1 \atop i} \right\} n^i.
$$

Since $\left\{ {n-1 \atop i} \right\} = 0$ for all $i > n-1$, we can stop summing at $i = n-1$ which yields:

$$
h_n = \sum_{i=0}^{n-1} \left\{ {n-1 \atop i} \right\} n^i.
$$

□

Corollary 3.15.1 (Warme) *The number of distinct (unrooted) hypertrees spanning n labeled vertices is*

$$\sum_{i=0}^{n-1} \left\{ {n-1 \atop i} \right\} n^{i-1}.$$

4

The Algorithm

This chapter presents a branch-and-cut algorithm for solving the MST in hypergraph problem — and by reduction the FST concatenation problem. First the algorithm is presented and its more important pieces are shown to be correct. A number of important implementation details are highlighted. Finally, empirical results are presented from a computational study containing a large number of problem instances, both randomly generated and from well known problem libraries. Each instance is solved as both a Euclidean and a rectilinear problem. The results indicate that these methods yield by far the fastest exact Steiner tree algorithm in existence.

4.1 Branch-and-Cut Procedure

This section presents a branch-and-cut algorithm that solves integer program (3.6). Lower bounds for the branch-and-cut are provided by the linear program relaxation of (3.6):

$$\min \{c\,x : x \in P\} \tag{4.1}$$

This lower bound has been extremely tight in practice. For most problems in the computational study (Section 4.2 below), the optimal solution to (4.1) is integral.

Unfortunately, there are an exponential number of constraints (3.4), making it impractical to solve (4.1) directly. Instead an iterative method is used that avoids dealing with so many constraints.

Let $\mathcal{C}$ be any finite collection of linear equations and inequalities. Let $\mathcal{P}$ be the polyhedron defined as those x satisfying every constraint in $\mathcal{C}$. Let $\mathcal{C}_0$ be some small subset of $\mathcal{C}$. For all $i \geq 0$ let $\mathcal{P}_i$ be the polyhedron defined as those x satisfying every constraint in $\mathcal{C}_i$.

The iteration begins with $i = 0$. At step i, let $\vec{x_i}$ be an optimal solution to the following linear program:

$$\min\{c\,x : x \in \mathcal{P}_i\}. \tag{4.2}$$

Let $\mathcal{V}_i \subset \mathcal{C}$ be any non-empty subset of constraints that are violated by $\vec{x_i}$. If no such subset $\mathcal{V}_i$ exists, then the iteration terminates and $\vec{x_i}$ is an optimal solution to linear program

$$\min\{c\,x : x \in \mathcal{P}\} \tag{4.3}$$

If such a $\mathcal{V}_i$ exists, however, define $\mathcal{C}_{i+1} = \mathcal{C}_i \cup \mathcal{V}_i$, increment i, and repeat.

In a landmark result, Grötschel, Lovász and Schrijver [23, 24] showed that this process always terminates, and that the number of iterations required is at most a polynomial function of the number of variables. In particular, the number of constraints is irrelevant — but must be finite.

Given an $\vec{x_i}$, we must either find a non-empty set $\mathcal{V}_i \subset \mathcal{C}$ of constraints that are violated by $\vec{x_i}$ *or show that every constraint in* $\mathcal{C}$ *is satisfied.* This sub-problem is known as *the separation problem* for constraints $\mathcal{C}$, since violated inequalities represent hyperplanes that *separate* $\vec{x_i}$ from polytope $\mathcal{P}$. The constraints are sometimes called *cutting-planes*, and the iterative process is often called *constraint generation*, or *cutting-plane generation*. This is the *cut* portion of a branch-and-cut algorithm.

If $\mathcal{C}$ contains an exponential (or even larger) number of constraints, it is not at all clear that the separation problem can be solved in polynomial time. But if it *can*, then the entire iteration can be solved in polynomial time.

For the particular case at hand, let P_0 be the polyhedron defined by (3.3), (3.5), all (3.4) for which $|S| = 2$, plus the following constraints:

$$x(\{t\} : V - \{t\}) \geq 1 \quad \text{for all } t \in V, \tag{4.4}$$

$$x_e + x_f \leq 1 \quad \text{for all } (e, f) \in \hat{C}, \tag{4.5}$$

where $\hat{C} \subset E \times E$ is an incompatibility relation, which may be empty. Constraints (4.4) are the 1-terminal cutset constraints. In Section 3.5.5 we showed these are equivalent to the $n-1$-terminal subtours. The cutset form normally yields constraint rows that are much more sparse than the equivalent subtour constraints. Constraints (4.5) introduce the optional incompatibility information to improve the initial LP. We use all (4.5) that are not dominated by 2-terminal subtours (i.e., the subtour constraint $x_1 + x_2 + x_3 + x_4 \leq 1$ dominates the incompatibility constraint $x_2 + x_4 \leq 1$). We then solve (4.1) by iterations of optimization (i.e., LP solving) followed by separation of constraints (3.4).

Figure 4.1 presents pseudo-code for the overall `branch_and_cut` algorithm. Each node η is a tuple containing three members: η_z is the node's objective value; η_x is the node's LP solution vector; and η_b is the set of all constraints that the node imposes due to branch variables. Figure 4.2 presents pseudo-code for the `process_node` subroutine. It iterates optimization and separation until either the node is infeasible, cut off, integral, or preempted. Node preemption is discussed in Section 4.1.3.5.

4.1.1 Branch-and-Cut Example

We now consider an example of how the branch-and-cut algorithm might behave when solving a rectilinear FST concatenation problem. Note that problem instances requiring several branch-and-cut nodes are invariably too large to serve usefully as detailed examples. The following computational example, therefore, is entirely hypothetical — we illustrate computational behavior and results without specifying the precise input data that produce them.

```
branch_and_cut (𝓕)
{
      lp = initial_LP (𝓕); η = new_node (); η_b = ∅
      node_set = ∅; UB = ∞; preempt_z = ∞

      loop
            status = process_node (lp, η, UB, preempt_z)

            case status in
            INFEASIBLE, CUTOFF:
                  destroy_node (η)

            INTEGRAL:
                  BEST = η_x; UB = η_z
                  destroy_node (η)
                  node_set = {η' ∈ node_set : η'_z < UB}

            FRACTIONAL:
                  (e, z_0, z_1) = choose_branching_variable (η_x)

                  η^0 = new_node (); η^0_z = z_0; η^0_b = η_b ∪ {x_e = 0}
                  η^1 = new_node (); η^1_z = z_1; η^1_b = η_b ∪ {x_e = 1}
                  node_set = node_set ∪ {η^0, η^1}
                  destroy_node (η)

            PREEMPTED:
                  node_set = node_set ∪ {η}
            endcase

            if node_set = ∅ then return (BEST)

            η = select_next_node (node_set)
            node_set = node_set \ {η}

            preempt_z = ∞
            for every η' ∈ node_set do
                  preempt_z = min (preempt_z, η'_z)
            end
      endloop
}
```

Figure 4.1: Algorithm 1 — `branch_and_cut`.

```
process_node (lp, η, UB, preempt_z)
{
     loop
          (status, η_z , η_x) = solve_LP (lp ∪ η_b)

          if status = INFEASIBLE then return (INFEASIBLE)

          /* status = OPTIMAL */
          if η_z ≥ UB then return (CUTOFF)

          if integer_feasible_solution (η_x) then return (INTEGRAL)

          if η_z > preempt_z then return (PREEMPTED)

          C = perform_separations (η_x)

          if C = ∅ then return (FRACTIONAL)

          add_constraints (lp, C)
     endloop
}
```

Figure 4.2: Algorithm 2 — `process_node`.

The algorithm is given the set $\mathcal{F}$ of FSTs, and it constructs the initial LP tableaux as described above. For this example the LP solver yields an optimal solution η_x having objective value of $Z = 1.2$. The separation algorithm finds a number of subtour constraints that η_x violates. These constraints are added to the LP tableaux which is re-optimized yielding a new optimal solution η_x having objective value $Z = 1.41$. After 58 more separate/optimize iterations, the separation procedure declares that $x = \eta_x$ violates none of the subtour constraints (3.4), and has objective value $Z = 1.6$.

It may happen that $x_e \in \{0, 1\}$ for all $e \in E$, in which case x is the incidence vector of the Steiner minimal tree. Unfortunately in this example there are a number of x_e that have fractional values. Although we do not yet have a Steiner minimal tree, we do have a lower bound — no SMT for the given point set can be shorter than 1.6. One of the fractional variables is $x_{14} = 1/2$. We must have either $x_{14} = 0$ or $x_{14} = 1$ in any valid Steiner tree incidence vector, so we break the initial problem into two subproblems as shown in Figure 4.3. Node 0 represents the initial problem with objective value $Z = 1.6$. Node 1 represents the subproblem obtained by appending the constraint $x_{14} = 0$ to those of node 0. Similarly, node 2 represents the subproblem obtained by appending the constraint $x_{14} = 1$ to those of node 0. We say that node 0 *branches* into nodes 1 and 2, and variable x_{14} is called the *branch variable*.

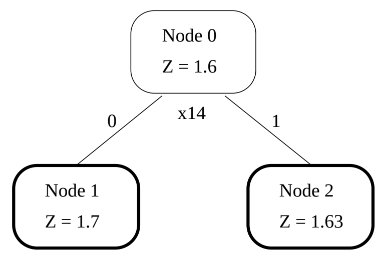


Figure 4.3: Example branch-and-cut tree 1.

Note that adding these constraints cannot cause the objective value Z to decrease — Z can only stay the same or increase. Since we want the lower bound to be as high as possible, it pays to choose a fractional variable (x_{14} in this case) for which the objective increases significantly in both subproblems. In this case, the objective has risen to 1.7 and 1.63 for nodes 1 and 2, respectively. Node 0 is now retired, since nodes 1 and 2 now collectively represent node 0's problem.

Node 2 is selected for processing (it has the lowest objective value). After 3 constraint generation cycles, the objective value for node 2 has risen to $Z = 1.65$. Although no subtour constraints are violated, the solution x is again fractional and variable $x_9 = 1/2$ is chosen as the branch variable. Node 2 therefore retires, being replaced by nodes 3 and 4 having objective values 1.8 and 1.75, respectively. Figure 4.4 illustrates the current state of the *branch-and-cut tree*.

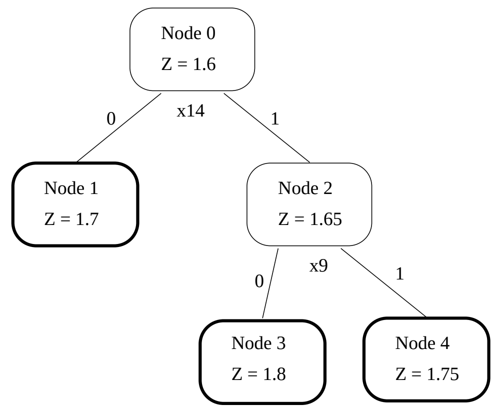


Figure 4.4: Example branch-and-cut tree 2.

Node 1 is now selected, and after 2 constraint generation cycles, its objective value has risen to $Z = 1.73$. The LP solution vector x, however, is fractional and $x_{23} = 3/8$ is chosen as the branch variable. Node 1 therefore retires, being replaced by nodes 5 and 6, having objective values 1.81 and 1.92, respectively. See Figure 4.5.

Node 4 is now selected, and after 87 iterations of constraint generation, its objective value has risen to $Z = 1.9$. The solution is fractional, however, and x_{23} is the chosen branch variable. Node 4 retires and is replaced by nodes 7 and 8 having objective values 1.91 and 1.92, respectively. See Figure 4.6.

Node 3 is selected next, and after 5 iterations of constraint generation, a solution is obtained that is integral and has objective value 1.84. This is the incidence vector of a valid Steiner tree (which may or may not be optimal). But we do know that nodes 6, 7 and 8 are now *suboptimal*, so we can retire them. Such nodes are said to be *cut off*. See Figure 4.7.

Node 5 is now selected, since it is the only remaining node to process. Its objective value rises to 1.82 after constraint generation, and fractional variable x_{12} is chosen for branching. Node 5 retires and is replaced by nodes 9 and 10, having objective values 1.87 and 1.83,

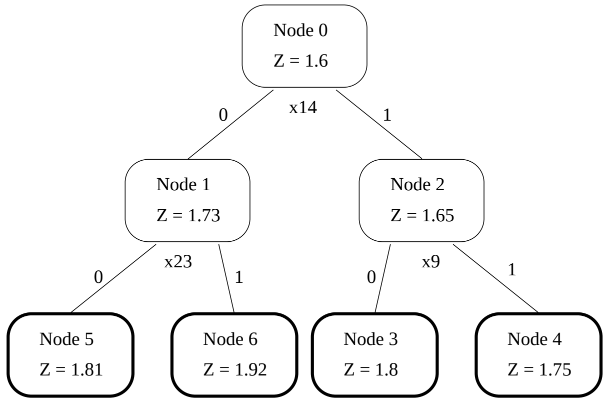


Figure 4.5: Example branch-and-cut tree 3.

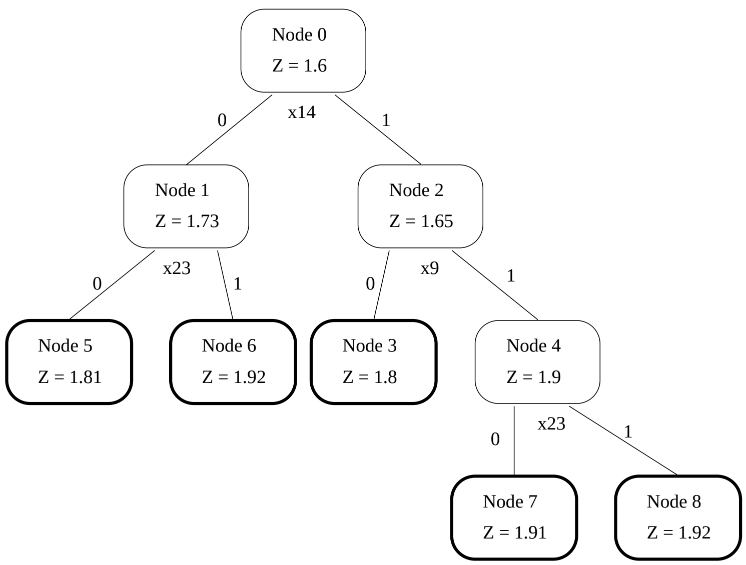


Figure 4.6: Example branch-and-cut tree 4.

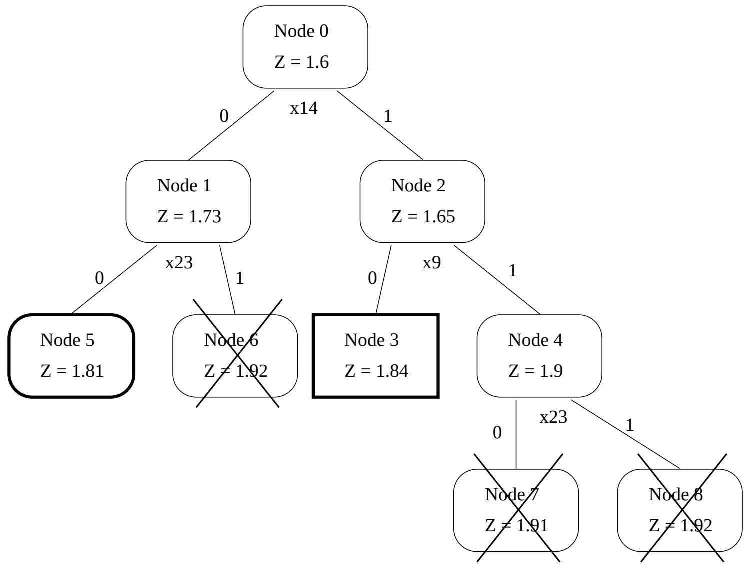


Figure 4.7: Example branch-and-cut tree 5.

respectively. Node 9 is immediately cut off, since its objective value already exceeds the upper bound of 1.84 established by node 3.

Node 10 now remains, and its solution is integral with objective $Z = 1.83$. This causes node 3 to be cut off, leaving node 10 as the optimal solution to the integer program as shown in Figure 4.8.

A number of design parameters must be specified for any branch-and-cut algorithm. Some procedure must be specified for selecting the next pending node to process. Separation procedures must be provided for constraint classes large enough to require them. Finally, some method of choosing branch variables must be specified. The most complex of these components are normally the separation procedures.

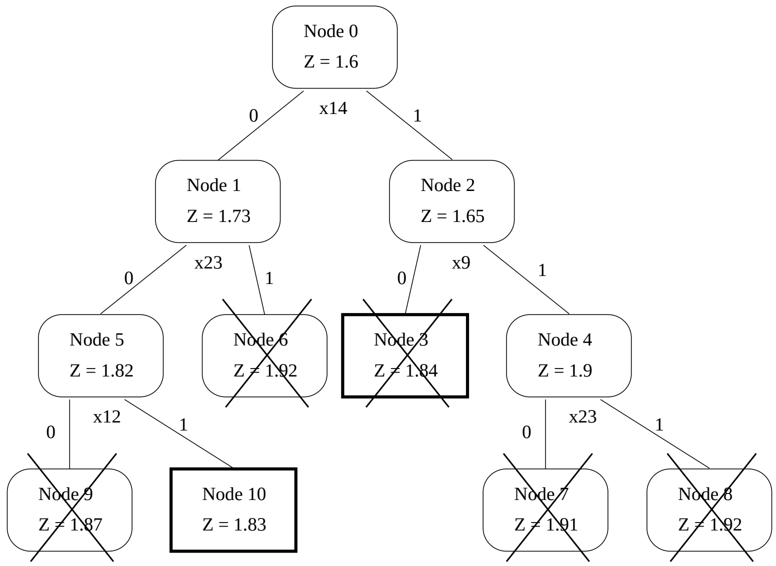


Figure 4.8: Final example branch-and-cut tree.

4.1.2 Separation of Subtour Elimination Constraints

We are given an LP solution x and we need to find an $S \subset V$ with $S \neq \emptyset$ that violates (3.4), or show that no such S exists. This section presents a flow formulation that solves this separation problem in polynomial time. We define the following function

$$f(S) = |S| - \sum_{e \in E} \max(|e \cap S| - 1, 0) x_e. \tag{4.6}$$

Then separating constraints (3.4) is equivalent to finding an $S \subset V$ such that $S \neq \emptyset$ and $f(S) < 1$.

We note that $f(S)$ is submodular. A function $f : 2^V \mapsto \mathbb{R}$ is *submodular* if and only if $f(A) + f(B) \geq f(A \cup B) + f(A \cap B)$ for all $A, B \subset V$.

4.1.2.1 Deterministic Flow Formulation

The first polynomial time deterministic algorithm for separating inequalities (3.4) was to find a minimum of the submodular function (4.6) using the "ellipsoid" method of Grötschel, Lovász, and Schrijver [23, 24]. Although a major improvement over heuristics alone, this method was exceedingly slow on separation subproblems larger than about 80 terminals.

Queyranne [45] noticed that minimizing $f(S)$ can be reduced to an instance of the "selection problem," as defined by Rhys [47] and Balinski [1]. These are equivalent to finding a "maximal closure of a graph," as defined by Picard [42]. These problems reduce to finding a minimum cut on a simple bipartite directed graph.

The flow network $G = (N, A)$ for this separation problem is constructed as follows: Let the set of distinct vertices be $N = \{s\} \cup Y \cup Z \cup \{t\}$ and the set of arcs be $A = A_1 \cup A_2 \cup A_3$, where

$$\begin{aligned} Y &= \{f_e : e \in E\}, \\ Z &= \{g_j : j \in V\}, \\ A_1 &= \{(s, f_e) : e \in E\}, \\ A_2 &= \{(f_e, g_j) : e \in E \wedge j \in e\}, \\ A_3 &= \{(g_j, t) : j \in V\}. \end{aligned}$$

For all $j \in V$, define

$$b_j = x(\delta(\{j\})) = \sum_{e \in E : j \in e} x_e.$$

We call b_j the "congestion level" of terminal j. Let arc $(s, f_e) \in A_1$ have capacity x_e, arc $(g_j, t) \in A_3$ have capacity $b_j - 1$, and let all arcs in A_2 have infinite capacity. See Figure 4.9 for an illustration of this flow network.

We define an $s - t$ cut of G to be a subset $W \subset N$ such that $s \in W$ and $t \notin W$. The weight $c(W)$ of $s - t$ cut W is the total capacity of all arcs $(u, v) \in A$ such that $u \in W$ and $v \notin W$.

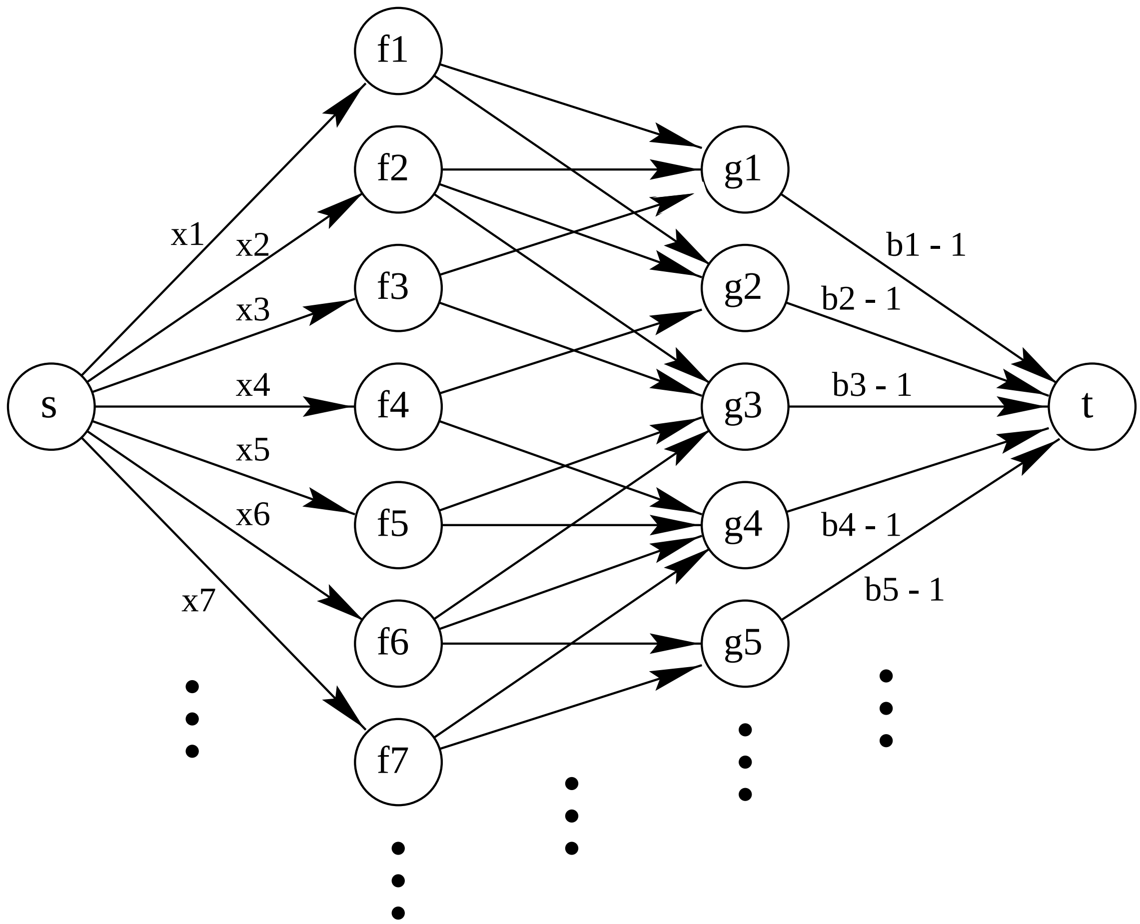


Figure 4.9: Flow network for subtour separation problem.

Theorem 4.1 *Let $W \subset N$ be an $s-t$ cut of G that minimizes $c(W)$. Let*

$$S_W = \{j \in V : g_j \notin W\}.$$

Then S_W is a minimum of $f(S)$.

Proof: Let W be such a minimum cut. We can write $W = \{s\} \cup F \cup G$, where $F \subseteq Y$ and $G \subseteq Z$. We note as follows that F is completely determined by G. Let $f_e \in Y$. Suppose there is an arc $(f_e, g_j) \in A_2$ such that $g_j \notin W$. Then we must have $f_e \notin W$ or else arc (f_e, g_j) of infinite capacity would span the cut, contradicting $c(W)$ being a minimum. The remaining case is where $g_j \in W$ for every g_j such that $(f_e, g_j) \in A_2$. We claim in this case that $f_e \in W$, since a search for an augmenting path from s to t would always label node f_e: if arc (s, f_e) has zero flow, then node f_e would be labeled directly from s; if arc (s, f_e) has positive flow, then there is at least one arc $(f_e, g_j) \in A_2$ with flow that can be returned to f_e. Node f_e would be labeled from such a g_j since all of them are in W.

Let $w_j = 1$ if $g_j \in W$ and $w_j = 0$ otherwise. Then $c(W)$ can be written in terms of the w_j as

$$\begin{aligned} c(W) &= \sum_{e \in E} \left[1 - \prod_{j \in e} w_j \right] x_e + \sum_{j \in V} (b_j - 1) w_j \\ &= \sum_{e \in E} -x_e \prod_{j \in e} w_j + \sum_{j \in V} (b_j - 1) w_j + \sum_{e \in E} x_e \end{aligned} \tag{4.7}$$

The last summation is a constant that does not depend on the w_j.

Now consider the function $f(S)$. Let $s_j = 1$ if $j \in S$ and $s_j = 0$ otherwise. Let $\bar{s}_j = 1 - s_j$ be the complementary $0-1$ variables. We write $f(S)$ in terms of the $\bar{s}_j$ as follows:

$$\begin{aligned} f(S) &= |S| - \sum_{e \in E} \max(|e \cap S| - 1, 0)\, x_e \\ &= \sum_{j \in V} s_j - \sum_{e \in E} \left[\left(\sum_{j \in e} s_j \right) - 1 + \prod_{j \in e} (1 - s_j) \right] x_e \\ &= \sum_{j \in V} (1 - \bar{s}_j) - \sum_{e \in E} \left[\left(\sum_{j \in e} (1 - \bar{s}_j) \right) - 1 + \prod_{j \in e} \bar{s}_j \right] x_e \\ &= |V| - \sum_{j \in V} \bar{s}_j - \sum_{e \in E} \left[|e| - \sum_{j \in e} \bar{s}_j - 1 + \prod_{j \in e} \bar{s}_j \right] x_e \\ &= |V| - \sum_{j \in V} \bar{s}_j - \sum_{e \in E} (|e| - 1)\, x_e + \sum_{e \in E} \left(x_e \sum_{j \in e} \bar{s}_j \right) - \sum_{e \in E} \left(x_e \prod_{j \in e} \bar{s}_j \right) \\ &= |V| - \sum_{j \in V} \bar{s}_j - \sum_{e \in E} (|e| - 1)\, x_e + \sum_{j \in V} \left(\bar{s}_j \sum_{e : j \in e} x_e \right) - \sum_{e \in E} \left(x_e \prod_{j \in e} \bar{s}_j \right) \end{aligned}$$

$$= \quad |V| - \sum_{j \in V} \bar{s}_j - \sum_{e \in E} (|e| - 1)\, x_e + \sum_{j \in V} b_j \bar{s}_j - \sum_{e \in E} \left(x_e \prod_{j \in e} \bar{s}_j \right)$$

$$= \quad \sum_{e \in E} \left(-x_e \prod_{j \in e} \bar{s}_j \right) + \sum_{j \in V} (b_j - 1) \bar{s}_j - \sum_{e \in E} (|e| - 1)\, x_e + |V| \tag{4.8}$$

The last two terms do not depend upon the $\bar{s}_j$ and are therefore constants that can be ignored. If we set $\bar{s}_j = w_j$ we see that the cut capacity (4.7) differs by a constant from the function (4.8) being minimized.

Suppose on the other hand that S is a minimum of $f(S)$. Then the corresponding W is seen to be a minimum of $c(W)$ because (4.7) and (4.8) differ by a constant, and because of the correspondence between S, G and F. □

Remark: Minimizing $f(S)$ is equivalent to finding the minimum of the following non-linear polynomial over $0 - 1$ variables

$$\sum_{e \in E} \left(-x_e \prod_{j \in e} \bar{s}_j \right) + \sum_{j \in V} (b_j - 1) \bar{s}_j \tag{4.9}$$

where all non-linear coefficients are negative[1]. Note that if the linear term coefficient of $\bar{s}_j$ isn't positive, then $\bar{s}_j = 1$ in any optimal solution. This is a problem reduction criteria $b_j \leq 1 \Longrightarrow s_j = 0 \Longrightarrow j \notin S$ that will be discussed further in Section 4.1.2.2.

To satisfy the $S \neq \emptyset$ constraint, let $t \in V$. Define a new function $f_t : (V - \{t\}) \mapsto \mathbb{R}$ as: $f_t(S) = f(S \cup \{t\})$. Let S_t^* be a minimum of $f_t(S)$. Then $S = S_t^* \cup \{t\}$ is a minimum of $f(S)$ satisfying $t \in S$. Repeating this for every $t \in V$ guarantees finding the minimum of $f(S)$ subject to the side constraint that $S \neq \emptyset$.

Finding a minimum of $f_t(S)$ corresponds to forcing $\bar{s}_t = 0$ in equation (4.9). When setting up the flow network for this problem, simply eliminate vertex g_t, vertices f_e such that $t \in e$ and the associated arcs when setting up the flow network. When the minimum

[1] Picard and Queyranne [43] showed that such problems are equivalent to the selection problem.

of $f_t(S)$ is obtained, delete terminal t from the separation problem, choose another t and iterate. Our implementation chooses a t that minimizes b_t on each iteration.

The deterministic flow formulation can be costly. To speed up the separation process, a suite of problem reductions and heuristics are used.

4.1.2.2 Reductions and Decompositions

Following Padberg and Wolsey [41], we can eliminate many terminals from consideration using the following idea, which is adapted from their proposition 2 (i). Recall from Section 4.1.2.1 that for all $t \in V$, we define

$$b_t = x(\delta(\{t\})) = \sum_{e \in E : t \in e} x_e. \tag{4.10}$$

We call b_t the "congestion level" of terminal t.

Lemma 4.1 *If $b_t \leq 1$ and $f(S \cup \{t\}) < 1$ then $f(S) \leq f(S \cup \{t\}) < 1$.*

Proof: If $t \in S$ then there is nothing to prove, so assume $t \notin S$. Let

$$A = \{e \in E : |e \cap S| \geq 1 \text{ and } t \in e\}$$

and

$$B = \{e \in E : |e \cap S| \geq 1 \text{ and } t \notin e\}.$$

Then

$$\begin{aligned}
f(S \cup \{t\}) - f(S) &= |S \cup \{t\}| - \sum_{e \in A} |e \cap S| x_e - \sum_{e \in B} (|e \cap S| - 1) x_e \\
&\quad - |S| + \sum_{e \in A} (|e \cap S| - 1) x_e + \sum_{e \in B} (|e \cap S| - 1) x_e \\
&= |S| + 1 - \sum_{e \in A} |e \cap S| x_e - |S| + \sum_{e \in A} |e \cap S| x_e - \sum_{e \in A} x_e \\
&= 1 - \sum_{e \in A} x_e \geq 1 - b_t \geq 0
\end{aligned}$$

□

We say that a terminal t such that $b_t \leq 1$ is *uncongested*, or is *congestion-free*. By iteratively eliminating all uncongested terminals, we are left with a core set $\hat{V}$ of *congested* terminals. We need only consider the *congested subhypergraph* $\hat{H} = (\hat{V}, \hat{E})$ (i.e., the subhypergraph induced by vertices $\hat{V}$). Appendix A presents a simple stack-based algorithm for computing $\hat{H}$ in linear time.

For every hypergraph $H = (V, E)$ having edge weights x_e for all $e \in E$, we define $\bar{H} = (V, \bar{E})$ such that $\bar{E} = \{e \in E : x_e > 0\}$. We call $\bar{H}$ the *support hypergraph* of H.

Lemma 4.2 *Let $H = (V, E)$ be a hypergraph with weights x_e for all $e \in E$ to separate. Let $\bar{H} = (V, \bar{E})$ be the support hypergraph of H. Let the connected components of $\bar{H}$ be $H_1 = (V_1, E_1)$, $H_2 = (V_2, E_2)$, $\ldots$, $H_k = (V_k, E_k)$. Let $S \subseteq V$ and $S_j = S \cap V_j$ for all $1 \leq j \leq k$. If $f(S) < 1$ then there is some j such that $f(S_j) < 1$.*

Proof: We assume that $k \geq 2$, since if $k = 1$ we have $S_1 = S$ and the theorem holds. Now assume to the contrary that $f(S_j) \geq 1$ for all $1 \leq j \leq k$. Then

$$\begin{aligned} f(S) &= |S| - \sum_{e \in E} \max(|e \cap S| - 1, 0) x_e \\ &= \sum_{j=1}^{k} \left[|S_j| - \sum_{e \in E_j} \max(|e \cap S_j| - 1, 0) x_e \right] \\ &= \sum_{j=1}^{k} f(S_j) \geq k \geq 2, \end{aligned}$$

a contradiction. □

Thus we may further confine our search to within single connected components. This is just a generalization of proposition 1 of [41] to hypergraphs.

Lemma 4.3 *Let $H = (V, E)$ be a hypergraph with weights x_e for all $e \in E$ to separate. Let $\bar{H} = (V, \bar{E})$ be the support hypergraph of H. Let A, B, C be a partition of V and E_A, E_B be a partition of $\bar{E}$ such that $|C| = 1$, $E_A = \{e \in \bar{E} : e \subseteq (A \cup C)\}$ and*

$E_B = \{e \in \bar{E} : e \subseteq (B \cup C)\}$. *If* $S \subseteq V$ *such that* $f(S) < 1$ *then* $f(S \cap (A \cup C)) < 1$ *or* $f(S \cap (B \cup C)) < 1$.

Proof: Assume $f(S \cap (A \cup C)) \geq 1$ and $f(S \cap (B \cup C)) \geq 1$. Then

$$
\begin{aligned}
f(S) &= |S| - \sum_{e \in E} \max(|e \cap S| - 1, 0)x_e \\
&= |S \cap (A \cup C)| + |S \cap (B \cup C)| - |S \cap C| \\
&\quad - \sum_{e \in E_A} \max(|e \cap S| - 1, 0)x_e - \sum_{e \in E_B} \max(|e \cap S) - 1, 0)x_e \\
&= f(S \cap (A \cup C)) + f(S \cap (B \cup C)) - |S \cap C| \\
&\geq f(S \cap (A \cup C)) + f(S \cap (B \cup C)) - 1 \geq 1
\end{aligned}
$$

a contradiction. □

One may therefore separately consider the subhypergraphs $(A \cup C, E_A)$ and $(B \cup C, E_B)$. By simple induction it may be shown that the search for violations may be confined to the biconnected components of $\bar{H}$. Suppose $t \in C$ (i.e., t is an articulation point). Then t can be congested initially, but uncongested within $(A \cup C, E_A)$ and/or $(B \cup C, E_B)$. If so, the reduction steps can be applied recursively. The subhypergraphs that remain after all reductions have been performed are called *congested components*. Without loss of generality, we will assume in the sequel that we are solving the separation problem on a single congested component $H_j = (V_j, E_j)$. Appendix A presents an algorithm that finds the biconnected components of a hypergraph in linear time.

These reductions are repeated every time the deterministic flow formulation deletes a terminal t. Deleting one or more terminals can produce opportunities for further reduction of the component.

4.1.2.3 Heuristics

We use two very quick heuristics that locate cycles that are *integral* as well as cycles that are *nearly integral* (i.e., integral except for a single fractional edge). The first procedure uses depth-first traversal over all edges $e \in E$ for which $x_e = 1$. Any terminal that is visited more than once implies a cycle that can be read off the stack. During this walk, the integral edges traversed are recorded, yielding the "integrally connected components". Although enumerating all cycles in this way could take exponential time, this seldom happens in practice due to the combined effects of constraints (3.3), (4.4) and fractional edges. This problem is avoided by terminating the traversal after some limited number of cycles have been discovered. Nearly integral cycles are discovered by checking each fractional edge $e \in E$ (i.e., $0 < x_e < 1$) against each integrally connected component. Any fractional edge having two or more terminals in common with a single integrally connected component represents a violated subtour that is "nearly integral."

The reductions are then applied, yielding a set of congested components $H_j = (V_j, E_j)$. If a congested component is small (e.g. $|V_j| \leq 10$), then it is reasonable to completely enumerate all subsets of V_j. On larger components we enumerate small-cardinality subsets. The maximum cardinality checked is a decreasing function of $|V_j|$.

If no violations have yet been discovered within $H_j = (V_j, E_j)$, we apply a method that heuristically reduces the hypergraph H_j to an undirected graph $\bar{H}_j$ and then apply Padberg and Wolsey's method [41] directly. The reduction is as follows: let $e \in E_j$. Let $k_e = |e| - 1$. Let $\bar{T}_e$ be any set of k_e edges from $\{(s,t) \in e \times e\}$ that forms a spanning tree for e. Assign each of these edges weight x_e. Taking the union of the $\bar{T}_e$ for all $e \in E_j$ we obtain a weighted multigraph. By merging equivalent edges and summing their weights we obtain a weighted graph to which we can apply the method [41]. This method is heuristic in that violations will be detected or not based upon the particular choices of spanning tree for each full set. Lacking a better way to proceed, we arbitrarily choose minimum spanning trees.

Finally, the deterministic flow formulation is applied to each congested component for which no violations have been found.

4.1.2.4 Constraint Strengthening

To obtain stronger constraints we *clean up* every subtour violation S by performing all of the reduction steps of Section 4.1.2.2 (removal of uncongested terminals, connected components, biconnected components, etc.) on the subhypergraph induced by S. Occasionally this will split a single "maximally violated" subtour into 2 or more subtours that are lesser violations but stronger constraints. This is done only for constraints discovered by the deterministic flow formulation — constraints discovered by the various heuristics seldom change during this process.

4.1.3 Implementation Details

This section presents some implementation details of the branch-and-cut procedure.

4.1.3.1 Constraint Pool

Constraints for the problem are kept in a *constraint pool.* Conceptually, every LP problem is solved over all of the constraints in the pool. For efficiency, however, the LP solver works with only a subset of these constraints at one time. Whenever a new LP solution x is obtained, the pool is scanned for constraints that x violates. All such constraints are added to the LP and the process iterates until all constraints in the pool are satisfied. Nothing is deleted from the LP tableaux until this happens. We count this as one LP in the empirical data — even though the LP solver may be invoked several times. There are two reasons for this: What we really want to count is optimize/separate iterations. Also the LP tableaux itself could serve as the pool, although less efficiently.

When a suitable fraction of the constraints in the LP tableaux have become slack, they are deleted from the LP but remain in the constraint pool for some time. Subsequent LP solutions may cause such constraints to be reused if violated again. Keeping only binding constraints in the LP tableaux decreases total LP solution time (including pool overhead) by about ten fold on most medium to large problems.

Constraints are deleted from the pool based on a measure of their *effectiveness*, which is inversely proportional to the product of a constraint's size and the number of iterations over which it has remained slack. The least effective constraints are deleted until sufficient space has been reclaimed for newly generated constraints. New constraints are given several iterations of grace time before they become elligible for deletion. The total size of the pool is maintained at a level that is proportional to the largest LP tableaux seen so far. This level is only a target, and may be exceeded if necessary.

A hash table permits duplicate constraints to be discovered quickly as new constraints are added. Each constraint in the pool also has a reference count indicating the number of inactive nodes for which the constraint is binding. Constraints with non-zero counts are never deleted. Without this protection, processing for the current node could undo previous progress made on inactive nodes and termination of the algorithm would no longer be guaranteed.

4.1.3.2 Node Processing

Processing of a node j involves iterating the LP solver and the separation algorithms. This iteration terminates when any of the following conditions is achieved:

1. The LP is infeasible.
2. The LP objective meets or exceeds the upper bound.
3. The LP objective exceeds that of some other node k.
4. The separation algorithms find no violated constraints.

In the first two cases the node is discarded. In the third case the node is set aside and processing is begun (or resumed) on node k instead. Section 4.1.3.5 below explains this in more detail. In the final case the LP solution x is either integral or fractional. If x is integral we record x as the best integer feasible solution seen so far, and discard node j. If x is fractional, we must choose a fractional variable x_e to branch on and replace node j with the two new nodes that result from further restricting node j's problem with constraints $x_e = 0$ and $x_e = 1$, respectively. Note that either (or both) of these new nodes might actually be discarded immediately if they are already known to be infeasible or suboptimal.

4.1.3.3 Selection of Branch Variables

When node processing terminates with a fractional solution, one of the variables x_e having a fractional value must be chosen for branching. The node is then replaced with two new nodes: one restricting $x_e = 0$, the other restricting $x_e = 1$. Since the number of nodes and number of fractional variables are typically both small, brute force is used to choose the best fractional variable to branch on.

Let $e \in E$ such that x_e is fractional. Let Z_e^0 and Z_e^1 be the LP objectives obtained by adding the constraints $x_e = 0$ and $x_e = 1$ correspondingly, to the current problem. The variable x_e that maximizes $Z_e = \min(Z_e^0, Z_e^1)$ is used as the branch variable.

Let Z_{max} be the best Z_e seen so far. If $Z_e^0 \leq Z_{max}$ then there is no need to compute Z_e^1. Similarly, if $Z_e^1 \leq Z_{max}$ then there is no need to compute Z_e^0. Since small changes in x_e tend to correlate well with small increases in the objective, some advantage can be gained by computing Z_e^0 first if $0 < x_e \leq 1/2$, and otherwise computing Z_e^1 first.

Suppose Z_e^0 is infeasible or exceeds the curent upper bound. Then it is possible to *fix* $x_e = 1$ in the current node and continue checking the remaining fractional variables. In similar fashion we can fix $x_e = 0$ if Z_e^1 is infeasible or exceeds the upper bound. If for some fractional x_e both Z_e^0 and Z_e^1 are either infeasible or exceed the upper bound, then no further variables need be tested — the node can be discarded.

It costs virtually nothing to check if any of these LP solutions is an integer feasible solution that improves upon the current upper bound. If so, the solution is recorded and the upper bound updated.

4.1.3.4 Node Selection

When a new node must be selected for processing, the *best node first* strategy is used. That is, the pending node having the lowest objective value is chosen. Although this strategy can require an exponential amount of memory in the worst case, this has not happened in practice due to the quality of the lower bound.

We do not record an LP basis for each node, only two bit vectors indicating which variables are fixed, and if so the value to which they are fixed — 0 or 1. Saving an entire LP basis for each node consumes substantially more memory. The loss of speed caused by beginning (or resuming) the processing of each node with a suboptimal basis appears to be a negligible percentage of the run time.

4.1.3.5 Node Preemption

Problems requiring several nodes sometimes trigger a severe inefficiency in naive branch-and-cut algorithms. Consider what happened to node 4 in the example problem of Section 4.1.1. A large number of expensive separate/optimize iterations (87 in this small example) were executed on node 4, raising its objective from 1.75 to 1.9. Unfortunately, most of this effort was for naught since node 4 was eventually cut off by node 3 at an objective value of 1.84. In fact, all iterations beyond those needed to achieve a node 4 objective value of 1.83 (the optimal solution) are wasted effort.

This happens quite often unless steps are taken to prevent it. When constraint generation for the current node j has increased its objective value Z_j to the point where it is no longer the best node, then we *preempt* the processing of node j. That is, whenever some other node k has $Z_k < Z_j$, we preempt processing of node j and begin (or resume) processing of node k. This keeps the computational effort focused on improving the global lower bound.

After generating some good constraints it is not unusual to process several nodes in turn (each preempted by the next) before encountering a node that resumes constraint generation. The effect is to re-solve the LP for each of these nodes using the recently discovered constraints.

4.2 Empirical Results

A large number of problem instances were attempted (1501). Optimal Euclidean and rectilinear Steiner minimal trees were obtained for every instance. All computations reported here were performed on a 125 MHz Sparc 20 with 256 megabytes of memory. All CPU times are reported in seconds. All LPs were solved using CPLEX version 4.0. All rectilinear FSTs were generated using the Salowe-Warme algorithm [49]. All Euclidean FSTs were generated using the Winter-Zachariasen algorithm [62].

We solved problem sets from the literature, including those of Soukup and Chow [52], and all of the problems from Beasley's OR-library [3, 2] having 1000 or fewer terminals. Because the OR-library problems jump directly from 100 points to 250 we included 15 random problem instances each of 110, 120, ..., 240 points to fill in the gaps in our plots. We also included a more thorough study of random instances including both medium sized problems (50 instances each at $100, 200, 300, 400$ and 500 terminals), and smaller problems (100 problems each of sizes $15, 20, 25, 30, 35, 40, 45$ and 50 points. In all the study contains 1501 problems ranging in size from 3 to 1000 terminals — all of which were solved to proven optimality as both Euclidean and rectilinear instances. Solving all 3002 problems required almost 63 CPU days of computation.

Figures 4.10 through 4.13 plot various execution statistics for FST generation: Figure 4.10 gives a scatter plot of Euclidean and rectilinear FST generation time versus number of terminals. Figure 4.11 plots average EFST and RFST generation time versus number of terminals, with minimum and maximum ranges shown. Note that the Winter-Zachariasen EFST generator is significantly more costly than the Salowe-Warme RFST generator, at least for problem sizes up to 1000 terminals. The plots suggest that these roles might reverse beyond about 1500 terminals. Figure 4.12 gives a scatter plot of both EFST and

RFST generation times versus the number of FSTs generated. Figure 4.13 plots the number of FSTs generated versus the number of terminals. For the uniformly distributed random data in this computational study, these appear to be essentially linear functions, with rectilinear averaging about $4n$ FSTs, and Euclidean averaging about $2.7n$ FSTs. For both the Euclidean and rectilinear problems, point sets are known that give rise to much larger numbers of FSTs.

Figures 4.14 through 4.19 plot various execution statistics for FST concatenation: Figures 4.14 and 4.15 give scatter plots of the FST concatenation times versus number of terminals for the Euclidean and rectilinear cases, respectively. Figures 4.16 and 4.17 scatter plot the same data, but instead as a function of the number of FSTs. Figure 4.18 overlays both plots. There appears to be very little difference in the way that Euclidean and rectilinear concatenation times are distributed when viewed this way. This suggests that the sole explanation for EFST concatenation being easier might be that fewer FSTs are normally obtained in Euclidean problems. Figure 4.19 plots the average EFST and RFST concatenation times as a function of the number of terminals.

Figures 4.20 through 4.25 plot various execution statistics for total SMT computation time: Figures 4.20 and 4.21 give scatter plots of total SMT computation time versus number of terminals for the Euclidean and rectilinear cases, respectively. Figures 4.22 and 4.23 scatter plot the same data, but instead as a function of the number of FSTs. Figure 4.24 overlays both plots. Finally, Figure 4.25 plots average SMT computation time with minimum and maximum ranges for both the Euclidean and rectilinear problems. Although Euclidean SMTs are more expensive for small numbers of points, they appear to become less costly above about 900 terminals.

See Appendix B for a tabulation of the specific computational details of each OR-library problem instance solved.

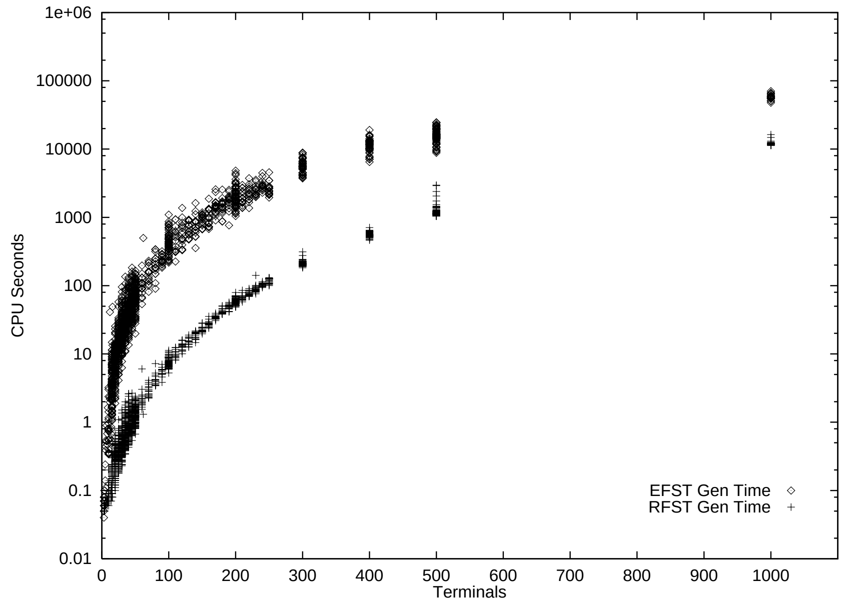


Figure 4.10: Scatter plot of FST generation time vs. number of terminals.

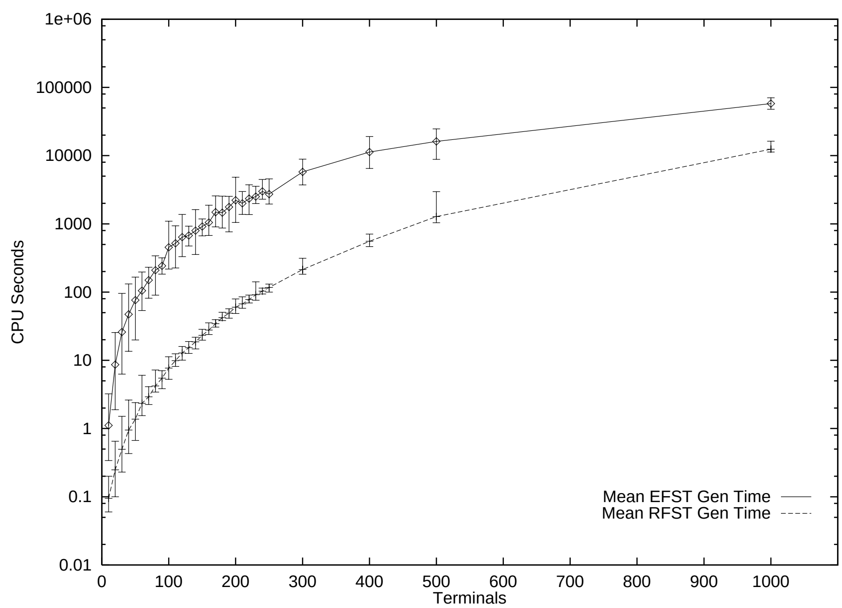


Figure 4.11: Plot of min/avg/max FST generation time vs. number of terminals.

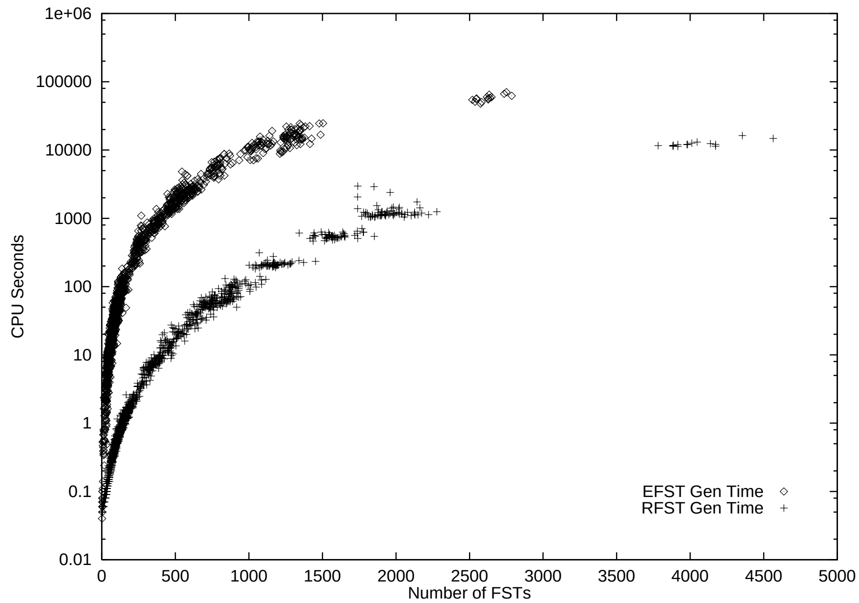


Figure 4.12: Scatter plot of FST generation time vs. number of FSTs generated.

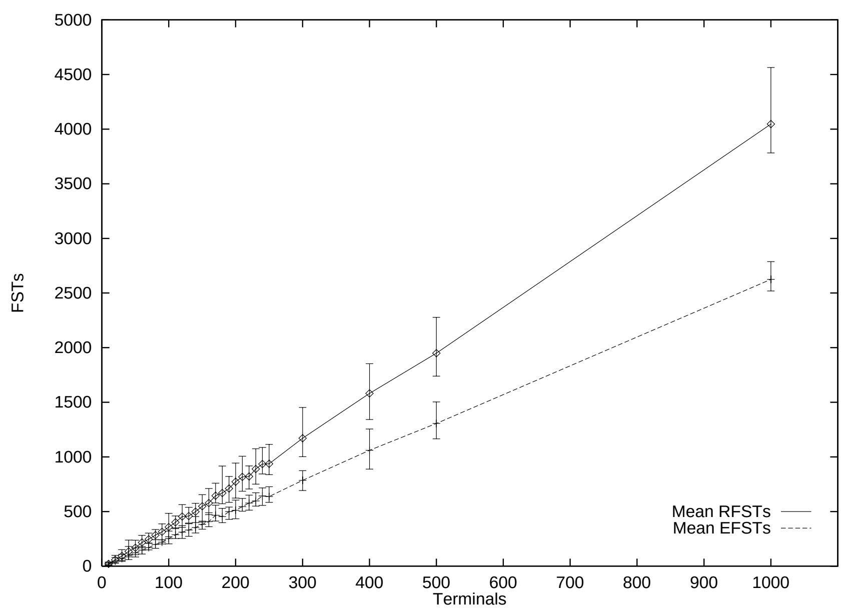


Figure 4.13: Plot of number of FSTs vs. number of terminals.

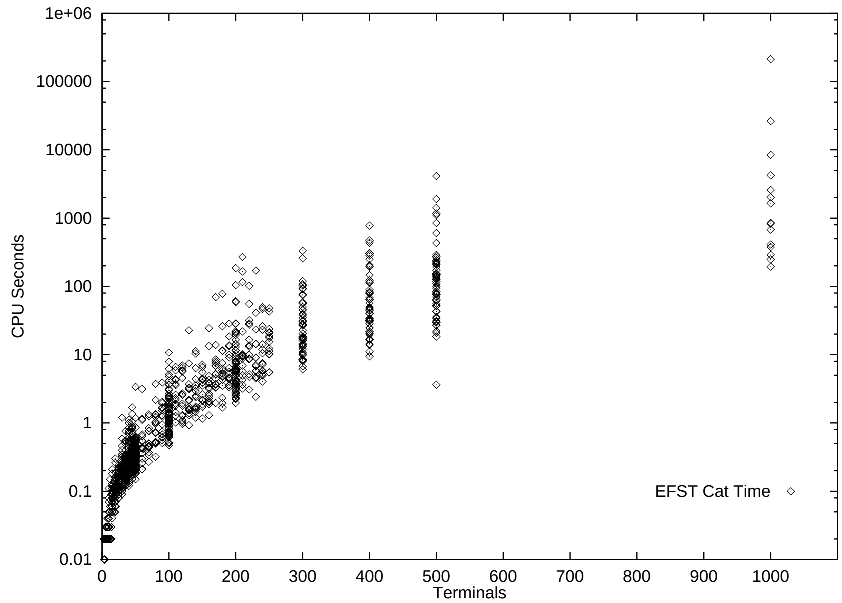


Figure 4.14: Scatter plot of Euclidean FST concatenation time vs. number of terminals.

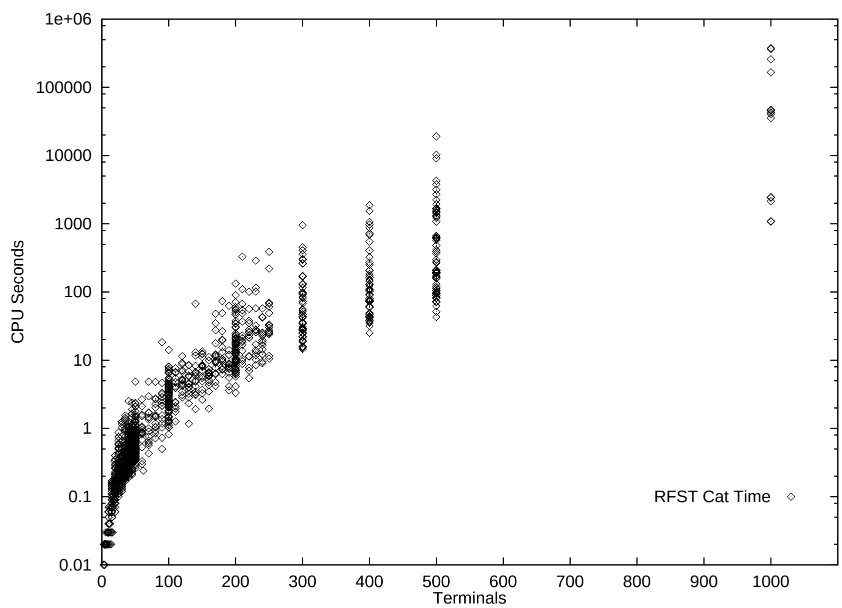


Figure 4.15: Scatter plot of rectilinear FST concatenation time vs. number of terminals.

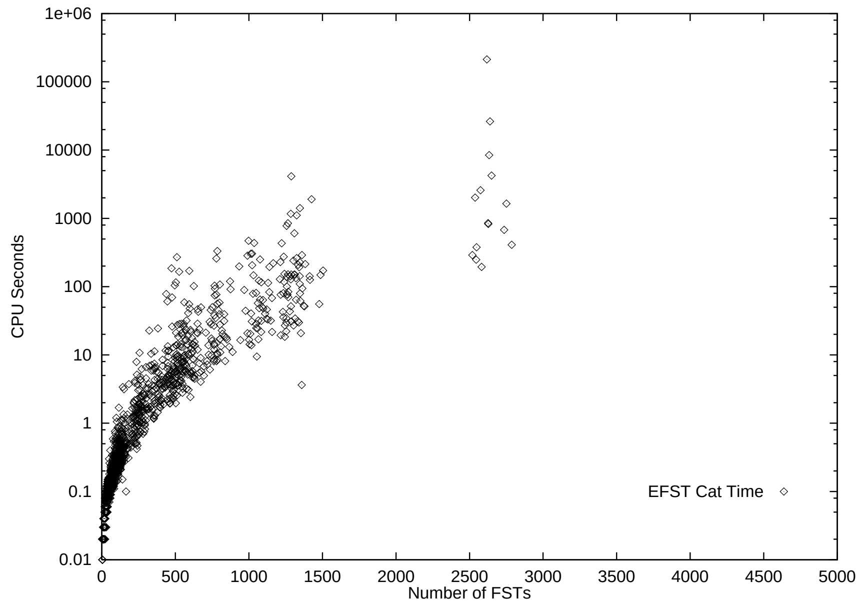


Figure 4.16: Scatter plot of Euclidean FST concatenation time vs. number of FSTs.

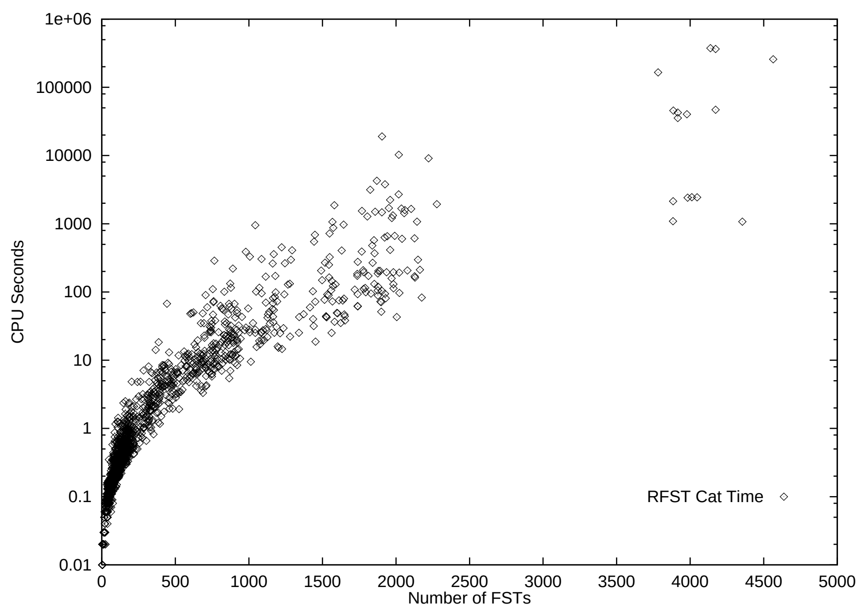


Figure 4.17: Scatter plot of rectilinear FST concatenation time vs. number of FSTs.

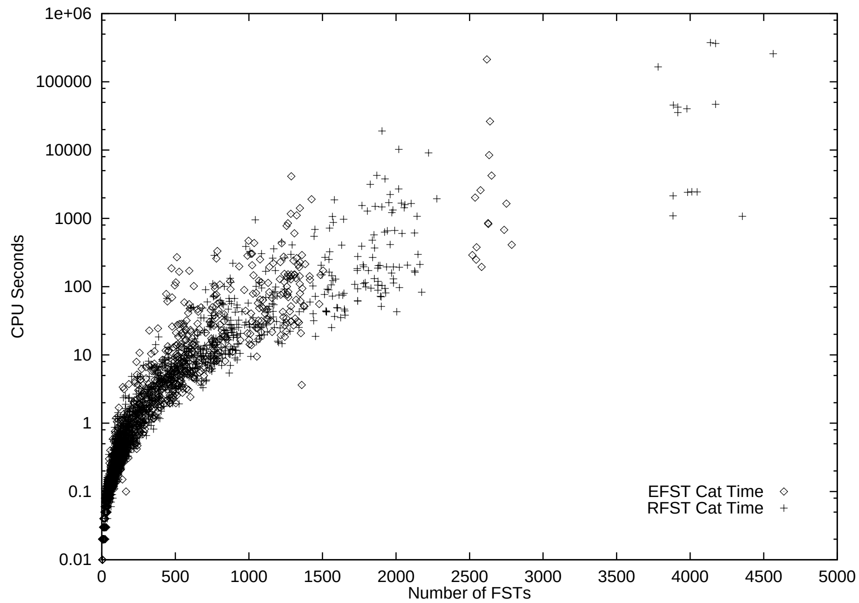


Figure 4.18: Scatter plot of EFST and RFST concatenation time vs. number of FSTs.

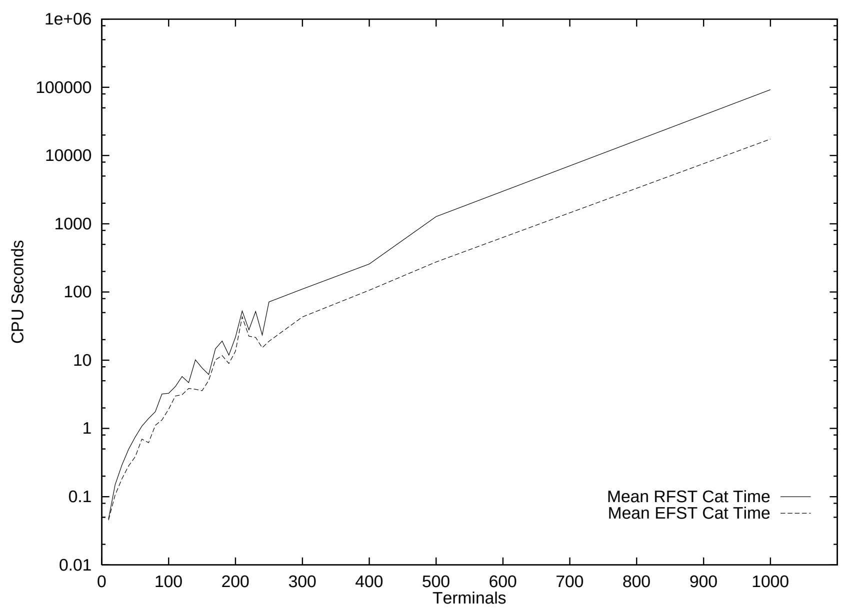


Figure 4.19: Plot of FST min/avg/max concatenation time vs. number of terminals.

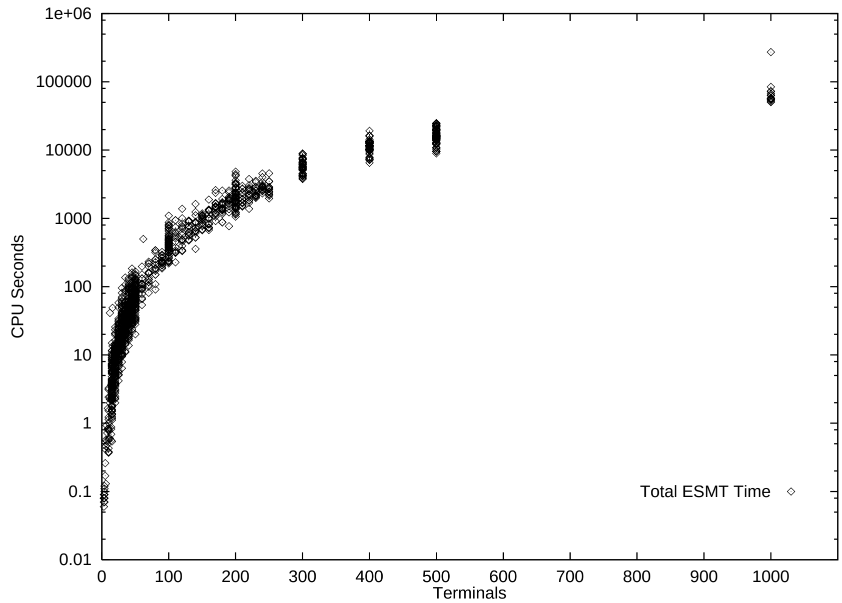


Figure 4.20: Scatter plot of Euclidean SMT total CPU time vs. number of terminals.

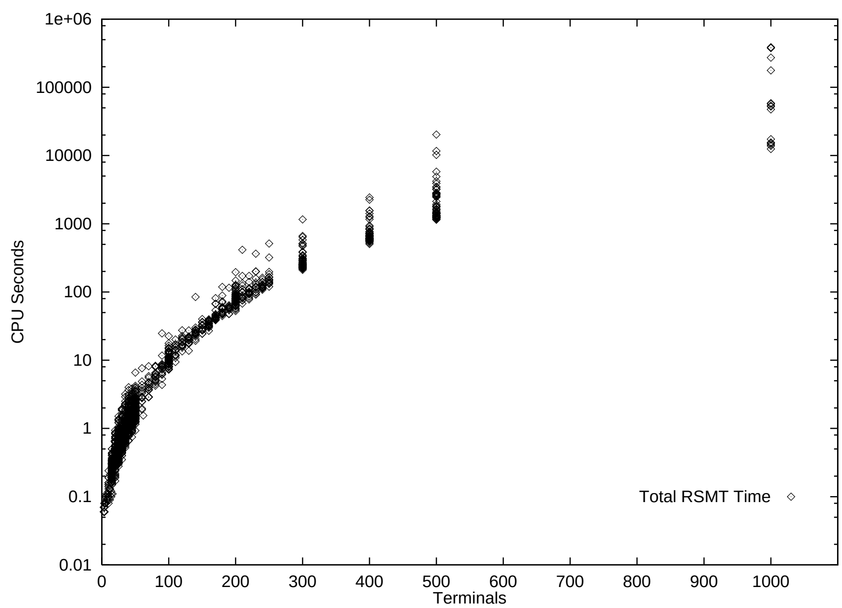


Figure 4.21: Scatter plot of rectilinear SMT total CPU time vs. number of terminals.

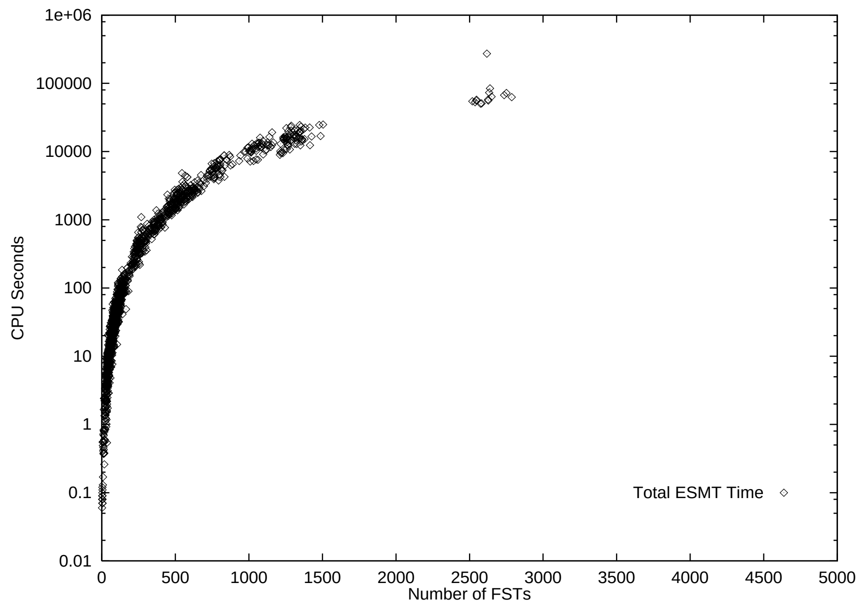


Figure 4.22: Scatter plot of Euclidean SMT total CPU time vs. number of FSTs.

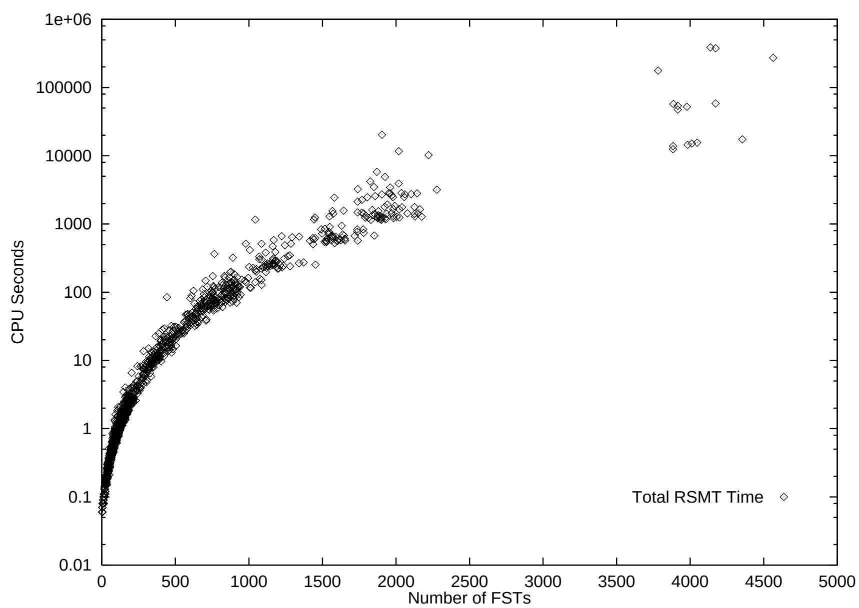


Figure 4.23: Scatter plot of rectilinear SMT total CPU time vs. number of FSTs.

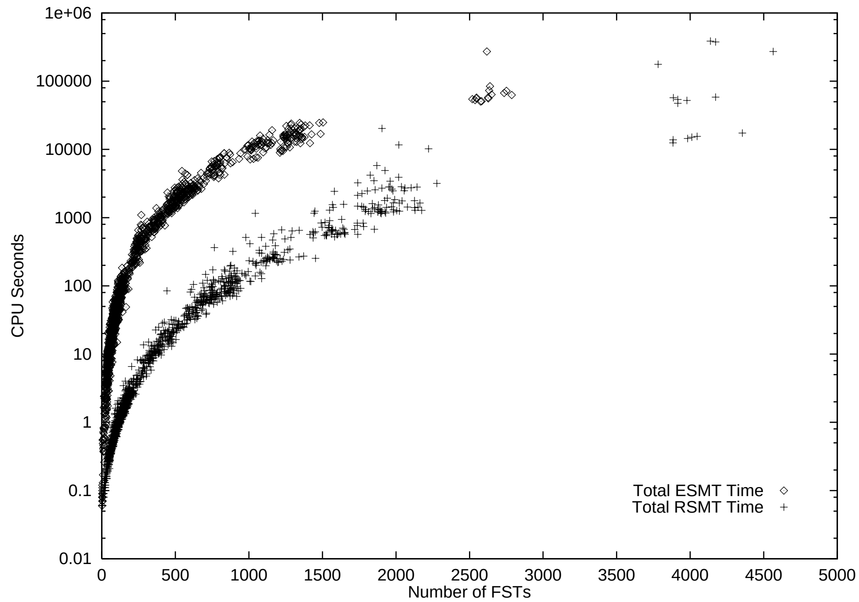


Figure 4.24: Scatter plot of ESMT and RSMT total CPU time vs. number of FSTs.

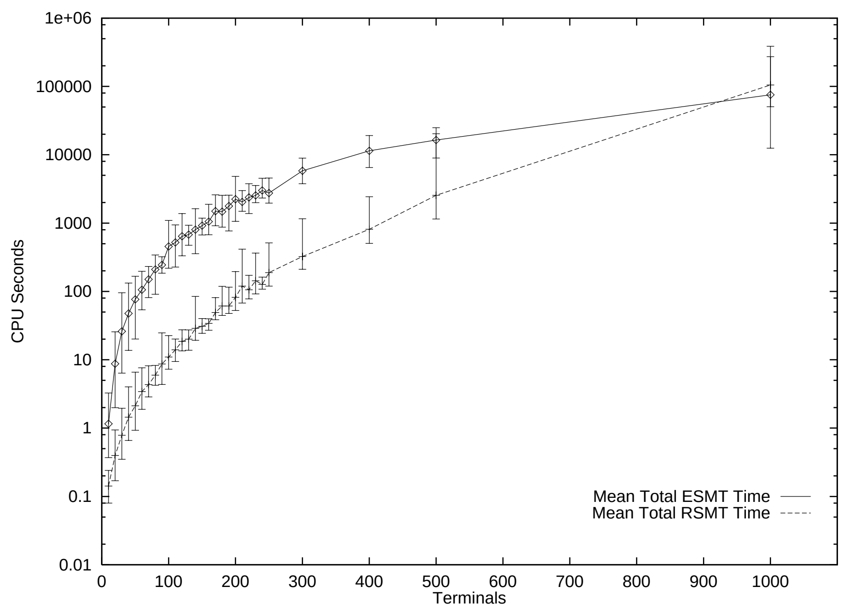


Figure 4.25: Plot of min/avg/max total CPU time vs. number of terminals.

Two problems required 19 branch-and-cut nodes, eight problems needed 9 to 15 nodes. All other problems required less than 8 branch-and-cut nodes. Over 92% of the problems obtained the optimal solution at the root node, with no branching.

The lower bound computed at the root node is extremely tight. Only 35 of the problems exceeded a gap of 0.1% — of these 35 problems only three had 100 terminals or more. The worst was a 25 terminal problem with a gap of 1.12257%. The gap was zero for almost 88% of the problems.

The problems that consume the most CPU time for a given number of terminals generally spend that time doing large numbers of constraint generation iterations that improve the objective value only minutely — improvements of less than one part in 10^9 per iteration are common in such circumstances. In Section 5.2 we propose a method that should greatly speed solution when convergence becomes this slow.

Warme, Winter and Zachariasen [58] present additional computational experience that combines the new FST concatenation algorithm presented here with state-of-the-art Euclidean [62] and rectilinear [64] FST generators. The computational study presented there includes instances from the `TSPLIB` problem set [46], as well as some pathological Euclidean and rectilinear instances. In that study, optimal Euclidean and rectilinear solutions were obtained for instances as large as 2392 points (`TSPLIB` instance `pr2392`).

Figure 4.26 presents the solution for a 1000 point problem (instance 1 from the OR-library `estein1000.txt` file).

Finally, to show that the method can handle even larger problems, we also solved a single 2000 terminal Euclidean instance obtained by combining problems 1 and 2 from the 1000 point OR-library problem set. See Figure 4.27 for a plot of the optimal solution.

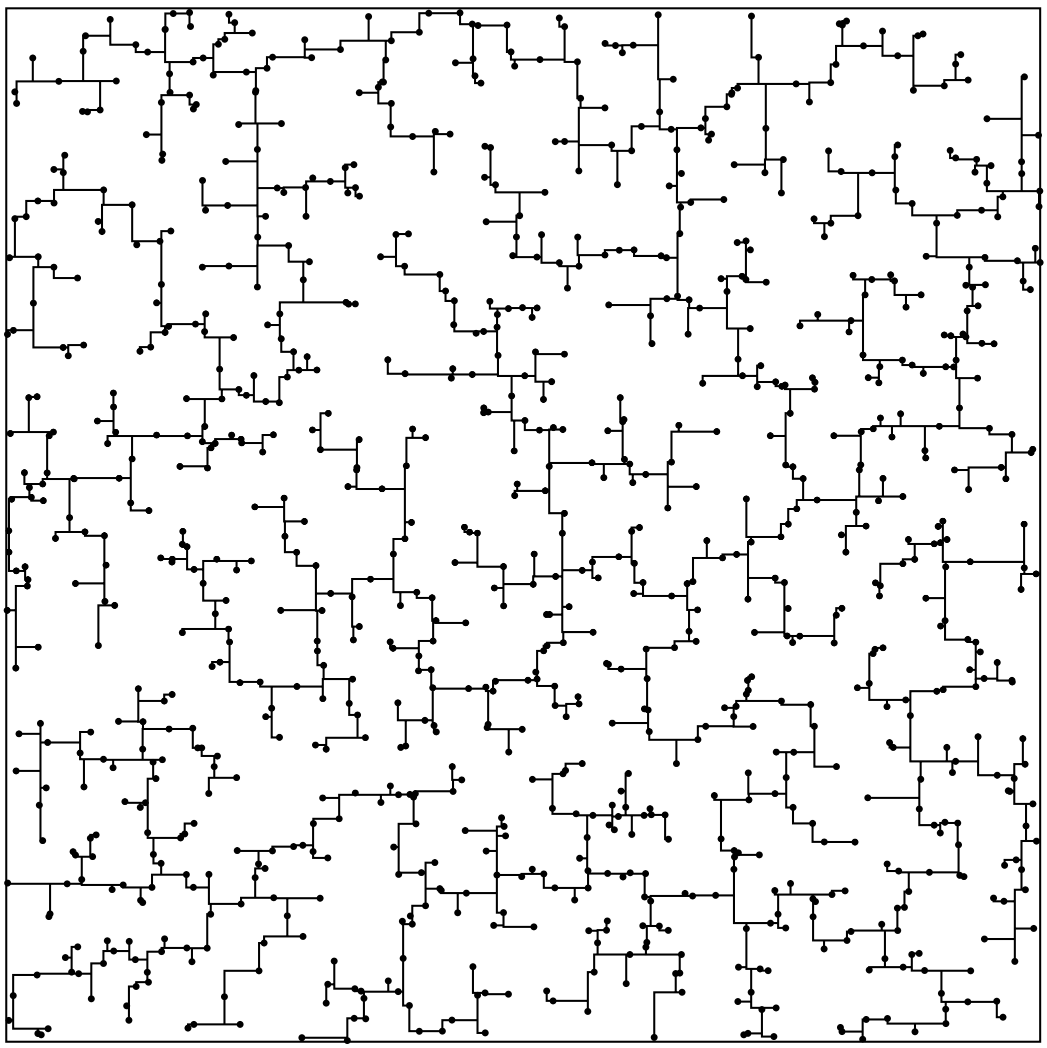


Figure 4.26: A rectilinear Steiner minimal tree for 1000 terminals. (Problem 1 from OR-library `estein1000.txt` file.)

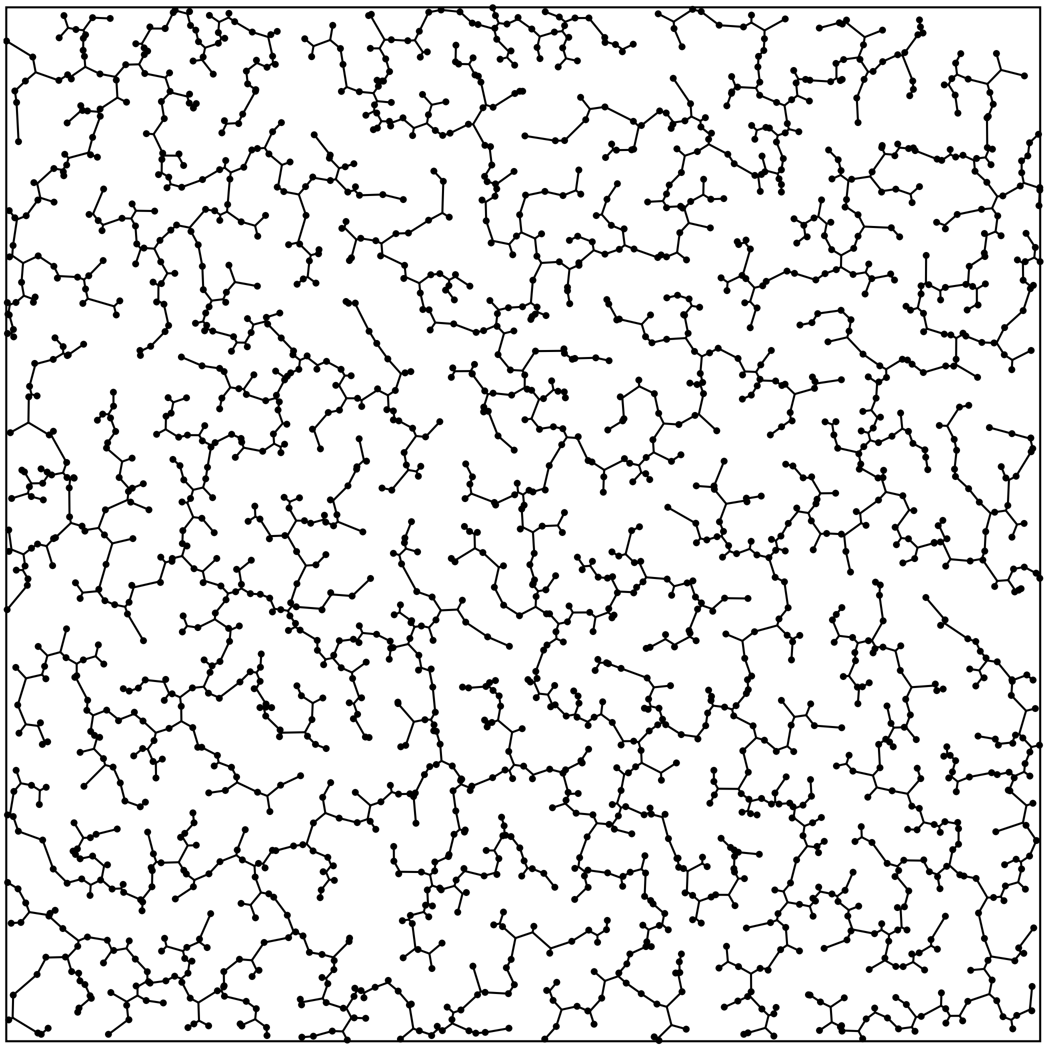


Figure 4.27: A Euclidean Steiner minimal tree for 2000 terminals. (Problems 1 and 2 combined from OR-library `estein1000.txt` file.)

5

Future Work

This chapter presents some ideas for future research that may improve upon the results presented here.

5.1 New Facet Classes

One of the best ways of improving a branch-and-cut method such as this is to identify major new classes of facet-defining inequalities. There are a number of ways to achieve this.

- Analyze numerous fractional solutions until a pattern is discovered.
- Obtain all facets of the polytope for small n. Analyze those that are unrecognized until a pattern is discovered.

Significant work has already been directed at the second method, resulting in complete lists of all facets of STHGP(n) for $2 \leq n \leq 5$. Enumeration of ST_n was done using a simple recursive `C` program. All facet enumeration computations were done using Christof and Loebel's `porta` code, which uses Fourier-Motzkin elimination [14] to obtain the convex hull as a set of linear equations and inequalities. We assume for the sake of concreteness that $V = \{0, 1, \ldots, n-1\}$. Suppose edge $e = \{1, 3, 5\}$. For conciseness we write x_e as x_{135}.

There are a large number of facet-defining inequalities. To conserve space we partition them into equivalence classes. For each class we present only the member count and one representative member inequality. Two inequalities are members of the same class if and only if they are identical under some permutation of the vertices.

STHGP(2) consists of the single point $x_{01} = 1$. There are no facets.

STHGP(3) has 4 hyperedges, 4 extreme points, and 4 facets. The facet classes are:

- (3) two-terminal subtours,
- (1) $x_{012} \geq 0$.

STHGP(4) has 11 hyperedges, 29 extreme points, and 22 facets. The facet classes are:

- (6) two-terminal subtours,
- (4) three-terminal subtours,
- (6) $x_{01} \geq 0$,
- (4) $x_{012} \geq 0$,
- (1) $x_{0123} \geq 0$,
- (1) $x_{012} + x_{013} + x_{023} + x_{123} + x_{0123} \leq 1$.

The first two classes are subtours, the next three classes are non-negativity constraints, and the final class is a single clique constraint.

STHGP(5) has 26 hyperedges and 311 extreme points and 172 facets. The facet classes are:

- (10) two-terminal subtours,
- (10) three-terminal subtours,
- (5) four-terminal subtours,

- (10) $x_{01} \geq 0$,
- (10) $x_{012} \geq 0$,
- (5) $x_{0123} \geq 0$,
- (1) $x_{01234} \geq 0$,
- (30) $x_{01} + x_{04} + x_{14} + x_{012} + x_{013} + 2x_{014} + x_{023} + x_{024} + x_{034} + x_{123}$
  $+2x_{124} + 2x_{134} + x_{234} + 2x_{0123} + 3x_{0124} + 3x_{0134} + 2x_{0234} + 2x_{1234}$
  $+3x_{01234} \leq 3$,
- (20) $x_{01} + x_{04} + x_{14} + x_{012} + 2x_{013} + 2x_{014} + x_{023} + x_{024} + 2x_{034} + x_{123}$
  $+x_{124} + 2x_{134} + x_{234} + 2x_{0123} + 3x_{0124} + 3x_{0134} + 2x_{0234} + 2x_{1234}$
  $+3x_{01234} \leq 3$,
- (10) $x_{01} + x_{02} + x_{03} + x_{04} + x_{12} + x_{13} + x_{14} \geq x_{0234} + x_{1234}$,
- (5) $x_{01} + x_{02} + x_{03} + x_{04} \geq x_{1234}$,
- (5) $x_{012} + x_{013} + x_{023} + x_{123} + x_{0123} + x_{0124} + x_{0134} + x_{0234} + x_{1234} + x_{01234} \leq 1$,
- (10) $x_{012} + x_{013} + x_{014} + x_{0123} + x_{0124} + x_{0134} + x_{0234} + x_{1234} + x_{01234} \leq 1$,
- (1) $x_{012} + x_{013} + x_{014} + x_{023} + x_{024} + x_{034} + x_{123} + x_{124} + x_{134} + x_{234}$
  $+2x_{0123} + 2x_{0124} + 2x_{0134} + 2x_{0234} + 2x_{1234} + 2x_{01234} \leq 2$,
- (30) $x_{01} + x_{012} + x_{013} + 2x_{014} + x_{024} + x_{034} + x_{124} + x_{134}$
  $+2x_{0123} + 2x_{0124} + 2x_{0134} + x_{0234} + x_{1234} + 2x_{01234} \leq 2$,
- (10) $x_{01} + x_{012} + x_{013} + x_{014} + x_{023} + x_{024} + x_{034} + x_{123} + x_{124} + x_{134}$
  $+x_{234} + 2x_{0123} + 2x_{0124} + 2x_{0134} + x_{0234} + x_{1234} + 2x_{01234} \leq 2$.

The $n = 6$ case poses an enormous computational effort, which is underway. As of this writing, 415 classes representing 311738 facets have been identified.

In Table 5.1 we summarize the basic properties of STHGP(n) for small n.

n	m	Extreme Points	Facets
2	1	1	0
3	4	4	4
4	11	29	22
5	26	311	172
6	57	4447	$\geq$ 311738
7	120	79745	
8	247	1722681	
9	502	43578820	
10	1013	1264185051	

Table 5.1: Properties of STHGP(n).

5.2 Early Branching

Problems that take excessive time to solve do not usually need an extraordinary number of branch-and-bound nodes. Normally it is the constraint generation process that takes so long to converge. When this happens it is possible to terminate constraint generation for the node and branch instead. This usually achieves a dramatic decrease in total solution time. The danger, however, is that the number of nodes can explode if branching is begun too soon. Good heuristics are needed for monitoring the convergence rate and deciding when to branch.

During periods of slow convergence it is also possible to begin testing the branching behaviour of each of the variables. One variable per iteration could be tested, in some most-promising-first heuristic order until a good variable is found or the convergence becomes extremely slow.

This is an obvious candidate for parallel execution. While one processor is optimizing the main LP, several others can be optimizing various slightly different subproblems. In each case the LP tableaux is identical — only the variable bounds are changed. Synchronization would be needed only once per iteration when the variable branching results would be gathered from the other processors, and newly generated constraints distributed to the other processors.

5.3 Steiner Problem in Graphs

A number of researchers have expressed interest in the problem of generating FSTs for the Steiner problem in graphs. It is not yet known whether this great advance in the geometric problems will transfer to the graph problem. There is also interest in the Steiner problem in directed graphs, since this problem is of considerable importance to the design of large communication networks.

5.4 New Formulations

For the Steiner problem in graphs it is known that a tighter formulation is obtained by using directed edges and identifying a unique terminal as the root vertex, although this doubles the number of problem variables. It is likely that a directed formulation of MST in hypergraph would also be tighter, although in the FST concatenation application this would more than triple the number of solution variables on average.

6

Conclusions

The method of computing Steiner minimal trees via FST generation and concatenation is currently the most efficient approach in practice. The FST generation processes for both the Euclidean and rectilinear metric were reviewed in substantial detail. The FST concatenation phase, however, has been the major bottleneck with this approach.

A new algorithm for FST concatenation was presented that significantly reduces this bottleneck. The new algorithm reduces FST concatenation to the problem of finding a minimum weight spanning tree in a hypergraph — which was shown to be strongly NP-complete. The MST in hypergraph problem was formulated as an integer program and the polyhedral theory of this problem was developed sufficiently to prove that all of the constraints in this integer programming formulation are facet-defining. The integer program is solved using a new branch-and-cut algorithm whose significant details were presented.

Empirical results show that on both rectilinear and Euclidean Steiner minimal tree problems the new FST concatenation algorithm vastly out-performs all other algorithms in existence. Its nearest rectlinear competitors seem to be Martin and Koch [34] (up to 40 terminals), and Fößmeier and Kaufmann [16] (70 terminals, but at least one instance of 100 terminals). For the Euclidean problem, Winter and Zachariasen [62] is the closest competitor at 150 terminals.

Provided a suitable FST generator is available, this method is applicable to other distance metrics and arbitrary dimensions — even the Steiner problem in graphs. In light of its great success on the rectilinear and Euclidean problems, it will be interesting to see how well the method works on the graph problem.

Despite the advance achieved in the computation of Steiner trees, it is likely that the MST in hypergraph results presented here will be the more important and lasting contribution. This is due to the inherent generality of hypergraphs and hypertrees as compared to Steiner trees.

A

Reduction Algorithms

This appendix presents two algorithms used by the problem reductions of Section 4.1.2.2. Both algorithms operate on a hypergraph $H = (V, E)$, and assume that for every $t \in V$, the set $E_t = \{e \in E : t \in e\}$ has been precomputed. Note that this is easily done in $O(|V| + |E| + k)$ time, where $k = \sum_{e \in E} |e|$.

Given a hypergraph $H = (V, E)$ with edge weights x_e for all $e \in E$, Algorithm A.1 computes the *congested subhypergraph* $\hat{H} = (\hat{V}, \hat{E})$. See Figure A.1. The first loop requires $O(|E|)$ time, and the second requires $O(|V| + k)$ time. The final loop runs at most $|V|$ times since each terminal is pushed onto the stack once at most. The loop for every $e \in E$ runs at most k times. The variable k_e is one less than the number of undeleted vertices in e, and k_e decrements to zero when only one vertex of e remains. This happens at most once per edge. When this happens, edge e is removed by the statement $DE = DE \cup \{e\}$, which runs at most $|E|$ times. The innermost loop runs at most k times total: it decreases the congestion level of the sole remaining undeleted vertex $v \in e$. Therefore this algorithm runs in $O(|V| + |E| + k)$ time and space. Correctness follows from two facts: that $t \in V$ is deleted at most once (and only after $b_t \leq 1$); and that edges are only deleted when they have one vertex left.

```
DV = DE = ∅;  S = emptystack
/* DV = vertices to discard, DE = edges to discard.  */
for every e ∈ E do
      if x_e > 0 then
            k_e = |e| − 1
      else
            k_e = 0
      endif
end
for every t ∈ V do
      b_t = ∑_{e∈E_t} x_e
      if b_t ≤ 1 then
            push t onto stack S;  DV = DV ∪ {t}
      endif
end
while stack S is not empty do
      pop t from stack S;  b_t = 0
      for every e ∈ E_t do
            if k_e > 0 then
                  k_e = k_e − 1
                  if k_e ≤ 0 then
                        DE = DE ∪ {e}
                        for every v ∈ e such that b_v > 0 do   /* only one such v.  */
                              b_v = b_v − x_e
                              if b_v ≤ 1 and v ∉ DV then
                                    push v onto stack S;  DV = DV ∪ {v}
                              endif
                        end
                  endif
            endif
      end
end
V̂ = V \ DV;  Ê = {e ∩ V̂ : e ∈ E \ DE and |e ∩ V̂| ≥ 2};  Ĥ = (V̂, Ê)
```

Figure A.1: Algorithm A.1 — compute congested subgraph.

Cockayne and Hewgill [10] propose to solve a problem equivalent to finding the biconnected components of a hypergraph by constructing a conventional graph G containing edge (i, j) if there is some hyperedge containing both vertices i and j. Finding the biconnected components of G then yields the biconnected components of the original hypergraph in a direct way.

Algorithm A.2 in Figures A.2 and A.3 is a slight modification of the standard biconnected components algorithm for conventional graphs. The modification permits it to operate directly on a hypergraph, however, which is superior in that it does not require the construction of a separate graph data structure. Algorithm A.2 is easily shown to run in $O(|V| + |E| + k)$ time and space. Its correctness is shown using the same argument as for the standard algorithm for conventional graphs, by simply considering chains instead of paths.

```
bcc(V,E)
    for every t ∈ V do
        DFS_t = 0; BACK_t = 0
    end
    S = emptystack; DE = ∅; j = 0
    /* DE = edges traversed */
    for every t ∈ V do
        if DFS_t ≤ 0 then
            traverse (t)
        endif
    end
end bcc
```

Figure A.2: Algorithm A.2 — biconnected components of hypergraph.

```
traverse (v)
    j = j + 1; DFS_v = j; BACK_v = j
    for every e ∈ E_v do
        if e ∉ DE then
            push e onto stack S; DE = DE ∪ {e}
        endif
        for every w ∈ e do
            if DFS_w ≤ 0 then
                traverse (w)
                if BACK_w ≥ DFS_v then
                    BE = ∅; BV = ∅
                    repeat
                        pop e_2 from stack S; BE = BE ∪ {e_2}; BV = BV ∪ e_2
                    until e_2 = e
                    output component (BV, BE)
                else if BACK_w < BACK_v then
                    BACK_v = BACK_w
                endif
            else if BACK_w < BACK_v then
                BACK_v = BACK_w
            endif
        end
    end
end traverse
```

Figure A.3: Subroutine `traverse` of Algorithm A.2.

B

Tabulated OR-Library Results

This appendix presents a complete tabulation of the computational details for each OR-library problem instance solved.

In tables B.1 through B.9, N is the number of terminals (and problem instance), M is the number of FSTs. Z is the length of the optimal RSMT. The "Z Root" column is the final LP objective value of the root node. The "% Gap" column is: $100(\mathrm{Z} - \mathrm{Z\ Root})/\mathrm{Z}$. "Nds" is the number of branch-and-bound nodes required — 1 node indicates that optimality was proven at the root node (without branching). "LPs" is the total number of optimize/separate iterations that were required. The "IRow" column is the initial number of constraints. The "RTight" column is the number of binding constraints in the final LP tableaux for the root node.

N		M	Z	Z	%	Nds	LPs	Constraints		CPU seconds		
				Root	Gap			IRow	RTight	FST Gen	FST Cat	Total
5	(1)	8	1.6643993	1.664399	0.00000	1	1	12	9	0.14	0.03	0.17
6	(2)	11	1.5004998	1.500500	0.00000	1	1	17	9	0.44	0.02	0.46
7	(3)	6	2.0776711	2.077671	0.00000	1	1	8	8	0.53	0.02	0.55
8	(4)	7	2.1387890	2.138789	0.00000	1	1	9	9	0.47	0.02	0.49
6	(5)	10	2.0440525	2.044052	0.00000	1	1	14	11	0.39	0.02	0.41
12	(6)	20	2.1842047	2.184205	0.00000	1	5	27	26	2.36	0.06	2.42
12	(7)	23	2.2052928	2.205293	0.00000	1	1	29	15	1.96	0.02	1.98
12	(8)	19	2.1777945	2.177795	0.00000	1	2	27	25	2.18	0.05	2.23
7	(9)	30	1.5594229	1.559423	0.00000	1	1	29	8	0.90	0.03	0.93
6	(10)	24	1.5987517	1.598752	0.00000	1	1	22	10	0.54	0.03	0.57
6	(11)	7	1.2741137	1.274114	0.00000	1	1	11	9	0.11	0.02	0.13
9	(12)	14	1.6483376	1.648338	0.00000	1	1	19	12	1.63	0.02	1.65
9	(13)	12	1.2733761	1.273376	0.00000	1	1	15	14	0.80	0.03	0.83
12	(14)	16	2.2049159	2.204916	0.00000	1	1	19	13	0.58	0.02	0.60
14	(15)	15	1.2304077	1.230408	0.00000	1	1	18	17	0.54	0.02	0.56
3	(16)	2	1.1667809	1.166781	0.00000	1	1	4	4	0.04	0.02	0.06
10	(17)	9	1.6427922	1.642792	0.00000	1	1	11	11	0.54	0.02	0.56
62	(18)	237	3.8176188	3.817619	0.00000	1	5	242	147	498.95	0.42	499.37
14	(19)	37	1.7064572	1.706457	0.00000	1	4	41	35	3.54	0.09	3.63
3	(20)	3	1.0396152	1.039615	0.00000	1	1	6	4	0.06	0.02	0.08
5	(21)	17	1.8181793	1.818179	0.00000	1	1	16	6	0.24	0.02	0.26
4	(22)	4	0.5032862	0.503286	0.00000	1	1	7	5	0.10	0.02	0.12
4	(23)	5	0.5130289	0.513029	0.00000	1	1	8	5	0.10	0.01	0.11
4	(24)	5	0.2528201	0.252820	0.00000	1	1	8	5	0.07	0.02	0.09
3	(25)	3	0.1989685	0.198968	0.00000	1	1	6	4	0.07	0.01	0.08
3	(26)	3	0.1243470	0.124347	0.00000	1	1	6	4	0.08	0.01	0.09
4	(27)	4	1.1781697	1.178170	0.00000	1	1	7	6	0.08	0.02	0.10
4	(28)	5	0.2044153	0.204415	0.00000	1	1	8	5	0.06	0.02	0.08
3	(29)	3	1.4659774	1.465977	0.00000	1	1	6	4	0.05	0.02	0.07
12	(30)	140	1.0198307	1.018917	0.08958	1	1	79	14	41.03	0.15	41.18
14	(31)	21	2.3321736	2.332174	0.00000	1	1	28	21	0.85	0.03	0.88
19	(32)	84	2.8142361	2.814236	0.00000	1	3	87	50	13.67	0.11	13.78
18	(33)	39	2.2258049	2.225805	0.00000	1	3	49	33	10.84	0.07	10.91
19	(34)	38	2.1381261	2.138126	0.00000	1	2	46	37	10.97	0.08	11.05
18	(35)	39	1.3554457	1.355446	0.00000	1	1	51	35	8.34	0.05	8.39
4	(36)	6	0.8789125	0.878912	0.00000	1	1	10	7	0.06	0.01	0.07
8	(37)	11	0.7660261	0.766026	0.00000	1	2	14	12	0.68	0.04	0.72
14	(38)	18	1.4248159	1.424816	0.00000	1	1	21	17	0.79	0.03	0.82
14	(39)	13	1.4312456	1.431246	0.00000	1	1	15	15	0.67	0.02	0.69
10	(40)	29	1.4179883	1.417988	0.00000	1	3	34	37	3.03	0.08	3.11
20	(41)	28	1.9767196	1.976720	0.00000	1	1	37	34	3.78	0.05	3.83
15	(42)	35	1.3152909	1.315291	0.00000	1	1	47	26	0.49	0.05	0.54
16	(43)	62	2.3307646	2.330765	0.00000	1	3	65	44	13.41	0.09	13.50
17	(44)	25	2.1869241	2.186924	0.00000	1	2	29	63	5.52	0.08	5.60
19	(45)	48	1.9309954	1.930995	0.00000	1	3	56	66	12.57	0.16	12.73
16	(46)	165	1.3660254	1.366025	0.00000	1	1	127	17	48.90	0.10	49.00

Table B.1: Euclidean results for Soukup and Chow problems.

N		M	Z	Z Root	% Gap	Nds	LPs	Constraints		CPU seconds		
								IRow	RTight	FST Gen	FST Cat	Total
5	(1)	8	1.87	1.870000	0.00000	1	1	19	8	0.06	0.02	0.08
6	(2)	10	1.64	1.640000	0.00000	1	1	18	7	0.06	0.02	0.08
7	(3)	9	2.36	2.360000	0.00000	1	1	15	8	0.07	0.02	0.09
8	(4)	12	2.54	2.540000	0.00000	1	1	21	13	0.06	0.03	0.09
6	(5)	10	2.26	2.260000	0.00000	1	1	18	7	0.06	0.02	0.08
12	(6)	22	2.42	2.420000	0.00000	1	2	35	15	0.09	0.04	0.13
12	(7)	22	2.48	2.480000	0.00000	1	1	35	13	0.10	0.03	0.13
12	(8)	21	2.36	2.360000	0.00000	1	3	35	17	0.09	0.04	0.13
7	(9)	24	1.64	1.640000	0.00000	1	1	84	8	0.09	0.02	0.11
6	(10)	16	1.77	1.770000	0.00000	1	1	45	10	0.07	0.03	0.10
6	(11)	8	1.44	1.440000	0.00000	1	1	16	9	0.06	0.02	0.08
9	(12)	19	1.80	1.800000	0.00000	1	1	42	10	0.07	0.03	0.10
9	(13)	14	1.50	1.500000	0.00000	1	1	28	10	0.08	0.03	0.11
12	(14)	12	2.60	2.600000	0.00000	1	1	13	13	0.07	0.02	0.09
14	(15)	22	1.48	1.480000	0.00000	1	2	40	31	0.10	0.06	0.16
3	(16)	2	1.60	1.600000	0.00000	1	1	4	4	0.05	0.01	0.06
10	(17)	11	2.00	2.000000	0.00000	1	1	15	13	0.07	0.02	0.09
62	(18)	126	4.04	4.040000	0.00000	1	4	223	149	1.31	0.24	1.55
14	(19)	35	1.88	1.880000	0.00000	1	2	120	43	0.12	0.07	0.19
3	(20)	4	1.12	1.120000	0.00000	1	1	10	4	0.06	0.01	0.07
5	(21)	11	1.92	1.920000	0.00000	1	1	26	6	0.08	0.02	0.10
4	(22)	5	.63	0.630000	0.00000	1	1	10	5	0.06	0.02	0.08
4	(23)	5	.65	0.650000	0.00000	1	1	10	5	0.05	0.01	0.06
4	(24)	6	.30	0.300000	0.00000	1	1	14	5	0.06	0.02	0.08
3	(25)	4	.23	0.230000	0.00000	1	1	10	4	0.05	0.01	0.06
3	(26)	3	.15	0.150000	0.00000	1	1	7	4	0.05	0.02	0.07
4	(27)	4	1.33	1.330000	0.00000	1	1	8	6	0.05	0.02	0.07
4	(28)	6	.24	0.240000	0.00000	1	1	12	5	0.06	0.02	0.08
3	(29)	4	2.00	2.000000	0.00000	1	1	10	4	0.05	0.01	0.06
12	(30)	52	1.10	1.100000	0.00000	1	4	219	25	0.14	0.07	0.21
14	(31)	25	2.59	2.590000	0.00000	1	1	49	15	0.10	0.02	0.12
19	(32)	64	3.13	3.130000	0.00000	1	2	215	78	0.26	0.09	0.35
18	(33)	51	2.68	2.680000	0.00000	1	3	141	37	0.17	0.09	0.26
19	(34)	75	2.41	2.410000	0.00000	1	2	241	39	0.42	0.08	0.50
18	(35)	72	1.51	1.510000	0.00000	1	2	244	61	0.38	0.09	0.47
4	(36)	3	.90	0.900000	0.00000	1	1	5	5	0.05	0.01	0.06
8	(37)	9	.90	0.900000	0.00000	1	1	15	13	0.07	0.02	0.09
14	(38)	14	1.66	1.660000	0.00000	1	1	18	18	0.08	0.03	0.11
14	(39)	14	1.66	1.660000	0.00000	1	1	18	18	0.07	0.03	0.10
10	(40)	18	1.55	1.550000	0.00000	1	2	37	22	0.09	0.05	0.14
20	(41)	30	2.24	2.240000	0.00000	1	2	51	37	0.11	0.06	0.17
15	(42)	39	1.53	1.530000	0.00000	1	1	215	31	0.15	0.05	0.20
16	(43)	68	2.55	2.550000	0.00000	1	3	302	41	0.30	0.11	0.41
17	(44)	31	2.52	2.520000	0.00000	1	2	57	58	0.14	0.07	0.21
19	(45)	80	2.20	2.200000	0.00000	1	4	380	59	0.32	0.33	0.65
16	(46)	24	1.50	1.500000	0.00000	1	1	17	17	0.08	0.03	0.11

Table B.2: Rectilinear results for Soukup and Chow problems.

N	M	Z	Z Root	% Gap	Nds	LPs	Constraints IRow	Constraints RTight	CPU seconds FST Gen	CPU seconds FST Cat	CPU seconds Total
10 (1)	23	2.0206738	2.020674	0.00000	1	1	26	16	0.81	0.03	0.84
10 (2)	12	1.6068682	1.606868	0.00000	1	1	16	15	0.34	0.03	0.37
10 (3)	20	2.2280743	2.228074	0.00000	1	2	27	17	0.55	0.05	0.60
10 (4)	14	1.7985963	1.798596	0.00000	1	2	19	17	0.76	0.04	0.80
10 (5)	13	1.6944333	1.694433	0.00000	1	1	18	17	0.40	0.03	0.43
10 (6)	23	2.3096026	2.309603	0.00000	1	1	29	17	3.22	0.04	3.26
10 (7)	23	2.2338586	2.233859	0.00000	1	2	29	19	0.80	0.04	0.84
10 (8)	15	2.1776829	2.177683	0.00000	1	2	21	19	2.37	0.04	2.41
10 (9)	34	1.9684782	1.968478	0.00000	1	9	34	26	1.42	0.11	1.53
10 (10)	17	2.0593317	2.059332	0.00000	1	1	23	18	0.74	0.04	0.78
10 (11)	32	1.9473221	1.947322	0.00000	1	2	33	22	0.96	0.05	1.01
10 (12)	12	1.7531237	1.753124	0.00000	1	1	16	14	0.36	0.02	0.38
10 (13)	17	1.7138867	1.713887	0.00000	1	1	24	17	0.35	0.03	0.38
10 (14)	27	1.9496522	1.949652	0.00000	1	4	34	26	1.16	0.07	1.23
10 (15)	26	1.6716456	1.671646	0.00000	1	1	29	20	1.09	0.05	1.14
20 (1)	51	3.0716427	3.071643	0.00000	1	2	57	38	13.88	0.07	13.95
20 (2)	40	2.8546314	2.854631	0.00000	1	2	49	46	4.90	0.08	4.98
20 (3)	53	2.4530918	2.453092	0.00000	1	2	54	54	9.37	0.08	9.45
20 (4)	46	2.4661165	2.466117	0.00000	1	4	56	46	12.18	0.10	12.28
20 (5)	51	2.9535470	2.953547	0.00000	1	5	55	51	12.41	0.11	12.52
20 (6)	33	3.1315695	3.131570	0.00000	1	2	41	40	3.50	0.07	3.57
20 (7)	39	3.0593002	3.059300	0.00000	1	2	47	41	9.14	0.07	9.21
20 (8)	51	3.3169861	3.316986	0.00000	1	4	58	43	10.58	0.09	10.67
20 (9)	31	3.1336342	3.133634	0.00000	1	2	40	37	2.20	0.06	2.26
20 (10)	55	3.0118726	3.011873	0.00000	1	4	67	80	6.24	0.15	6.39
20 (11)	59	2.3180526	2.318053	0.00000	1	3	64	47	4.72	0.08	4.80
20 (12)	47	2.6537453	2.653745	0.00000	1	2	58	44	11.31	0.09	11.40
20 (13)	38	3.0228482	3.022848	0.00000	1	2	49	41	2.40	0.08	2.48
20 (14)	39	2.9330086	2.933009	0.00000	1	1	49	37	7.43	0.06	7.49
20 (15)	34	2.7914795	2.791479	0.00000	1	2	42	34	4.73	0.07	4.80

Euclidean

N	M	Z	Z Root	% Gap	Nds	LPs	Constraints IRow	Constraints RTight	CPU seconds FST Gen	CPU seconds FST Cat	CPU seconds Total
10 (1)	27	2.2920745	2.292075	0.00000	1	4	67	22	0.10	0.06	0.16
10 (2)	17	1.9134104	1.913410	0.00000	1	1	33	17	0.08	0.03	0.11
10 (3)	16	2.6003678	2.600368	0.00000	1	2	32	20	0.08	0.05	0.13
10 (4)	19	2.0461116	2.046112	0.00000	1	2	42	30	0.08	0.06	0.14
10 (5)	13	1.8818916	1.881892	0.00000	1	1	22	13	0.07	0.03	0.10
10 (6)	38	2.6540768	2.654077	0.00000	1	1	149	19	0.20	0.04	0.24
10 (7)	25	2.6025072	2.602507	0.00000	1	2	63	23	0.10	0.06	0.16
10 (8)	24	2.5056214	2.505621	0.00000	1	2	65	24	0.09	0.06	0.15
10 (9)	22	2.2062355	2.206236	0.00000	1	2	54	23	0.09	0.06	0.15
10 (10)	15	2.3936095	2.393610	0.00000	1	1	28	18	0.07	0.03	0.10
10 (11)	31	2.2239535	2.223953	0.00000	1	3	102	23	0.13	0.06	0.19
10 (12)	15	1.9626318	1.962632	0.00000	1	1	26	14	0.06	0.02	0.08
10 (13)	22	1.9483914	1.948391	0.00000	1	1	61	17	0.08	0.04	0.12
10 (14)	30	2.1856128	2.185613	0.00000	1	2	88	26	0.12	0.07	0.19
10 (15)	21	1.8641924	1.864192	0.00000	1	2	54	21	0.10	0.06	0.16
20 (1)	64	3.3703886	3.370389	0.00000	1	1	212	33	0.31	0.07	0.38
20 (2)	58	3.2639486	3.263949	0.00000	1	2	162	61	0.25	0.12	0.37
20 (3)	45	2.7847417	2.784742	0.00000	1	2	123	57	0.15	0.09	0.24
20 (4)	88	2.7624394	2.750218	0.44241	1	2	469	93	0.65	0.29	0.94
20 (5)	81	3.4033163	3.392034	0.33152	1	14	368	63	0.45	0.40	0.85
20 (6)	55	3.6014241	3.601424	0.00000	1	2	169	42	0.19	0.08	0.27
20 (7)	68	3.4934874	3.493487	0.00000	1	2	209	106	0.49	0.15	0.64
20 (8)	63	3.8016346	3.788557	0.34401	1	5	178	50	0.24	0.15	0.39
20 (9)	35	3.6739939	3.673994	0.00000	1	2	63	43	0.11	0.08	0.19
20 (10)	56	3.4024740	3.402474	0.00000	1	4	179	88	0.22	0.21	0.43
20 (11)	50	2.7123908	2.712391	0.00000	1	2	129	40	0.20	0.08	0.28
20 (12)	89	3.0451397	3.045140	0.00000	1	2	415	48	0.52	0.13	0.65
20 (13)	50	3.4438673	3.443867	0.00000	1	2	139	47	0.16	0.08	0.24
20 (14)	55	3.4062374	3.406237	0.00000	1	2	150	43	0.19	0.09	0.28
20 (15)	55	3.2303746	3.230375	0.00000	1	2	150	48	0.19	0.09	0.28

Rectilinear

Table B.3: Results for OR-library problems 10–20 points.

N	M	Z	Z Root	% Gap	Nds	LPs	Constraints IRow	Constraints RTight	CPU seconds FST Gen	CPU seconds FST Cat	CPU seconds Total
30 (1)	65	3.5787601	3.578760	0.00000	1	5	75	106	21.66	0.18	21.84
30 (2)	62	3.5766544	3.576654	0.00000	1	8	73	92	26.33	0.21	26.54
30 (3)	62	3.6568972	3.656897	0.00000	1	10	72	104	26.29	0.24	26.53
30 (4)	79	3.7114129	3.711413	0.00000	1	8	90	69	27.84	0.16	28.00
30 (5)	73	3.6138448	3.613845	0.00000	1	4	80	67	27.42	0.13	27.55
30 (6)	80	3.4974427	3.497443	0.00000	1	3	83	66	48.18	0.12	48.30
30 (7)	67	3.8136810	3.813681	0.00000	1	4	78	97	17.80	0.18	17.98
30 (8)	76	3.6858000	3.685800	0.00000	1	9	87	88	30.10	0.29	30.39
30 (9)	63	3.1809772	3.180977	0.00000	1	4	77	67	12.62	0.12	12.74
30 (10)	53	3.7189924	3.718992	0.00000	1	2	64	71	10.48	0.09	10.57
30 (11)	57	3.5901878	3.590188	0.00000	1	3	64	87	14.08	0.11	14.19
30 (12)	76	3.4239470	3.423947	0.00000	1	3	88	66	20.99	0.12	21.11
30 (13)	48	3.2224452	3.222445	0.00000	1	3	63	67	9.79	0.10	9.89
30 (14)	87	3.8532497	3.853250	0.00000	1	2	92	87	36.25	0.15	36.40
30 (15)	87	3.7718083	3.771808	0.00000	1	8	96	84	41.04	0.32	41.36
40 (1)	113	3.9283544	3.928354	0.00000	1	7	124	99	47.67	0.24	47.91
40 (2)	89	4.0668744	4.066874	0.00000	1	9	107	103	36.69	0.43	37.12
40 (3)	88	4.3845457	4.384546	0.00000	1	4	100	90	51.78	0.17	51.95
40 (4)	74	3.8531666	3.853167	0.00000	1	4	91	79	13.56	0.14	13.70
40 (5)	97	4.5432520	4.543252	0.00000	1	5	107	101	50.30	0.18	50.48
40 (6)	87	4.4151983	4.415198	0.00000	1	4	98	88	35.34	0.14	35.48
40 (7)	81	4.0319228	4.031923	0.00000	1	3	93	109	31.25	0.16	31.41
40 (8)	103	4.2734870	4.273487	0.00000	1	2	112	87	60.37	0.14	60.51
40 (9)	139	4.6224129	4.622413	0.00000	1	5	155	126	91.18	0.27	91.45
40 (10)	119	5.0832060	5.083206	0.00000	1	5	130	100	103.35	0.23	103.58
40 (11)	85	4.1399269	4.139927	0.00000	1	8	104	107	39.39	0.24	39.63
40 (12)	96	3.9078624	3.907862	0.00000	1	31	110	127	48.79	0.55	49.34
40 (13)	102	4.5604964	4.560496	0.00000	1	3	113	139	49.04	0.26	49.30
40 (14)	122	4.3578080	4.357808	0.00000	1	2	131	184	44.70	0.27	44.97
40 (15)	137	4.5075847	4.507585	0.00000	1	6	139	101	94.93	0.28	95.21

Euclidean

N	M	Z	Z Root	% Gap	Nds	LPs	Constraints IRow	Constraints RTight	CPU seconds FST Gen	CPU seconds FST Cat	CPU seconds Total
30 (1)	106	4.0692993	4.069299	0.00000	1	3	402	80	0.53	0.20	0.73
30 (2)	112	4.0900061	4.089173	0.02037	1	21	470	85	0.66	1.25	1.91
30 (3)	98	4.3120444	4.312044	0.00000	1	3	363	75	0.68	0.18	0.86
30 (4)	94	4.2150958	4.215096	0.00000	1	7	350	93	0.46	0.23	0.69
30 (5)	76	4.1739748	4.173975	0.00000	1	4	213	95	0.37	0.22	0.59
30 (6)	128	3.9955139	3.995514	0.00000	1	4	615	93	0.76	0.26	1.02
30 (7)	94	4.3761391	4.376139	0.00000	1	3	442	96	0.60	0.18	0.78
30 (8)	100	4.1691217	4.169122	0.00000	1	4	441	122	0.83	0.29	1.12
30 (9)	70	3.7133658	3.713366	0.00000	1	3	174	132	0.24	0.20	0.44
30 (10)	68	4.2686610	4.268661	0.00000	1	2	171	95	0.32	0.12	0.44
30 (11)	107	4.1647993	4.164799	0.00000	1	5	502	127	0.79	0.39	1.18
30 (12)	79	3.8416720	3.841672	0.00000	1	2	224	87	0.32	0.13	0.45
30 (13)	92	3.7406646	3.740665	0.00000	1	6	337	74	0.41	0.24	0.65
30 (14)	140	4.2897025	4.289702	0.00000	1	2	703	251	1.36	0.54	1.90
30 (15)	128	4.3035576	4.303558	0.00000	1	12	864	83	0.75	0.77	1.52
40 (1)	122	4.4841522	4.484152	0.00000	1	3	384	120	0.75	0.22	0.97
40 (2)	128	4.6811310	4.681131	0.00000	1	6	510	118	0.76	0.35	1.11
40 (3)	117	4.9974157	4.997416	0.00000	1	3	354	139	0.80	0.25	1.05
40 (4)	90	4.5289864	4.528986	0.00000	1	3	220	113	0.47	0.20	0.67
40 (5)	160	5.1940413	5.181185	0.24752	4	26	860	109	1.51	2.51	4.02
40 (6)	123	4.9753385	4.975339	0.00000	1	3	467	143	0.70	0.32	1.02
40 (7)	126	4.5639009	4.563901	0.00000	1	5	494	112	0.76	0.28	1.04
40 (8)	122	4.8745996	4.874600	0.00000	1	7	412	100	0.83	0.36	1.19
40 (9)	166	5.1761789	5.176179	0.00000	1	4	716	201	1.70	0.81	2.51
40 (10)	163	5.7136852	5.713685	0.00000	1	4	850	114	1.33	0.30	1.63
40 (11)	126	4.6734214	4.673421	0.00000	1	11	358	242	0.69	0.67	1.36
40 (12)	115	4.3843378	4.384338	0.00000	1	9	383	106	0.69	0.54	1.23
40 (13)	125	5.1884545	5.188454	0.00000	1	3	424	113	0.68	0.26	0.94
40 (14)	141	4.9166952	4.916695	0.00000	1	2	456	237	0.87	0.38	1.25
40 (15)	158	5.0828067	5.082807	0.00000	1	3	635	185	1.01	0.41	1.42

Rectilinear

Table B.4: Results for OR-library problems 30–40 points.

N		M	Z	Z	%	Nds	LPs	Constraints		CPU seconds		
				Root	Gap			IRow	RTight	FST Gen	FST Cat	Total
50	(1)	104	4.8366014	4.836601	0.00000	1	3	122	105	32.66	0.19	32.85
50	(2)	154	4.9484046	4.948405	0.00000	1	5	168	138	83.40	0.36	83.76
50	(3)	113	4.7471702	4.747170	0.00000	1	5	134	145	49.47	0.28	49.75
50	(4)	115	4.4690747	4.469075	0.00000	1	3	131	123	31.49	0.23	31.72
50	(5)	121	4.8648257	4.864826	0.00000	1	14	138	133	59.74	0.37	60.11
50	(6)	112	4.9234586	4.923459	0.00000	1	5	125	120	115.26	0.34	115.60
50	(7)	126	4.3613187	4.361319	0.00000	1	21	145	167	64.76	0.57	65.33
50	(8)	116	4.7027470	4.702747	0.00000	1	3	136	105	50.45	0.22	50.67
50	(9)	142	4.6760739	4.676074	0.00000	1	11	154	162	107.40	0.46	107.86
50	(10)	126	4.6277910	4.627791	0.00000	1	3	141	133	92.54	0.33	92.87
50	(11)	119	4.6693857	4.669386	0.00000	1	5	139	116	67.03	0.25	67.28
50	(12)	126	4.6732215	4.673222	0.00000	1	7	140	127	75.53	0.45	75.98
50	(13)	112	4.6564710	4.656471	0.00000	1	3	128	99	37.98	0.18	38.16
50	(14)	109	4.7098685	4.709869	0.00000	1	2	127	106	61.15	0.15	61.30
50	(15)	128	4.6079909	4.607991	0.00000	1	15	139	137	59.23	0.46	59.69
60	(1)	143	4.7740453	4.774045	0.00000	1	19	163	191	91.46	1.11	92.57
60	(2)	140	4.8129870	4.812987	0.00000	1	5	159	176	110.97	0.68	111.65
60	(3)	148	4.9458783	4.945878	0.00000	1	9	161	163	100.00	0.44	100.44
60	(4)	131	4.8461805	4.846181	0.00000	1	11	151	168	106.14	0.36	106.50
60	(5)	127	4.8355513	4.835551	0.00000	1	15	153	177	77.98	0.54	78.52
60	(6)	147	5.2504575	5.250458	0.00000	1	9	163	252	131.36	0.64	132.00
60	(7)	173	5.2142524	5.214252	0.00000	1	9	195	152	196.85	0.42	197.27
60	(8)	111	5.1173207	5.117321	0.00000	1	4	132	141	65.34	0.21	65.55
60	(9)	161	4.9086808	4.908681	0.00000	1	12	172	161	109.27	0.55	109.82
60	(10)	151	5.0587019	5.058702	0.00000	1	4	170	130	89.49	0.30	89.79
60	(11)	130	4.9327588	4.932759	0.00000	1	3	155	127	53.68	0.21	53.89
60	(12)	180	5.2924352	5.292435	0.00000	1	4	191	229	150.42	0.43	150.85
60	(13)	151	5.2663823	5.266382	0.00000	1	27	163	183	91.91	3.15	95.06
60	(14)	176	5.0235502	5.023550	0.00000	1	15	190	161	130.06	1.15	131.21
60	(15)	136	4.9670958	4.967096	0.00000	1	3	156	156	67.68	0.26	67.94

Euclidean

N		M	Z	Z	%	Nds	LPs	Constraints		CPU seconds		
				Root	Gap			IRow	RTight	FST Gen	FST Cat	Total
50	(1)	172	5.4948660	5.494866	0.00000	1	10	728	164	1.69	0.99	2.68
50	(2)	186	5.5484245	5.548425	0.00000	1	16	797	151	1.48	0.89	2.37
50	(3)	190	5.4691035	5.469104	0.00000	1	4	926	165	1.75	0.86	2.61
50	(4)	141	5.1535766	5.153577	0.00000	1	7	403	147	0.93	0.35	1.28
50	(5)	158	5.5186015	5.518601	0.00000	1	9	591	136	1.33	0.46	1.79
50	(6)	183	5.5804287	5.580429	0.00000	1	16	874	154	1.54	1.59	3.13
50	(7)	190	4.9961178	4.996118	0.00000	1	19	832	139	1.72	1.47	3.19
50	(8)	121	5.3754708	5.375471	0.00000	1	4	338	123	0.86	0.30	1.16
50	(9)	167	5.3456773	5.343995	0.03146	1	6	689	304	1.28	1.23	2.51
50	(10)	181	5.4037963	5.403796	0.00000	1	10	828	151	1.39	0.89	2.28
50	(11)	155	5.2532923	5.253292	0.00000	1	6	482	143	0.99	0.42	1.41
50	(12)	146	5.3409291	5.325255	0.29347	5	15	503	157	1.10	0.77	1.87
50	(13)	129	5.3891019	5.389102	0.00000	1	4	449	132	0.85	0.30	1.15
50	(14)	160	5.3551419	5.355142	0.00000	1	3	718	185	1.19	0.63	1.82
50	(15)	171	5.2180862	5.218086	0.00000	1	6	623	130	1.19	0.44	1.63
60	(1)	219	5.3761423	5.376142	0.00000	1	23	900	176	2.25	2.10	4.35
60	(2)	282	5.5367804	5.530190	0.11902	2	18	1690	174	6.04	1.59	7.63
60	(3)	206	5.6566797	5.656680	0.00000	1	5	821	248	1.96	0.82	2.78
60	(4)	234	5.5371042	5.537104	0.00000	1	19	1306	217	2.22	2.67	4.89
60	(5)	195	5.4704991	5.462873	0.13941	1	7	650	178	1.81	0.68	2.49
60	(6)	201	6.0421961	6.042196	0.00000	1	14	779	194	1.88	1.03	2.91
60	(7)	259	5.8978041	5.897804	0.00000	1	7	1411	215	3.04	0.88	3.92
60	(8)	233	5.8138178	5.813818	0.00000	1	9	1210	225	2.51	0.96	3.47
60	(9)	210	5.5877112	5.587711	0.00000	1	4	824	217	1.79	1.07	2.86
60	(10)	208	5.7624488	5.762449	0.00000	1	8	881	182	2.03	0.85	2.88
60	(11)	169	5.6141666	5.614167	0.00000	1	3	522	156	1.66	0.30	1.96
60	(12)	243	5.9791362	5.979136	0.00000	1	13	1152	163	2.40	0.89	3.29
60	(13)	214	6.1213533	6.121353	0.00000	1	12	841	197	2.12	1.52	3.64
60	(14)	215	5.6035528	5.603553	0.00000	1	3	880	148	1.95	0.53	2.48
60	(15)	179	5.6622581	5.662258	0.00000	1	4	510	175	1.55	0.33	1.88

Rectilinear

Table B.5: Results for OR-library problems 50–60 points.

N		M	Z	Z	%	Nds	LPs	Constraints		CPU seconds		
				Root	Gap			IRow	RTight	FST Gen	FST Cat	Total
70	(1)	158	5.4303745	5.430375	0.00000	1	5	184	178	99.67	0.34	100.01
70	(2)	176	5.3275902	5.327590	0.00000	1	11	194	219	125.75	0.76	126.51
70	(3)	197	5.3911607	5.391161	0.00000	1	5	212	158	231.29	0.50	231.79
70	(4)	164	5.4989330	5.498933	0.00000	1	5	185	193	155.13	0.84	155.97
70	(5)	165	5.4766967	5.476697	0.00000	1	11	187	166	134.25	0.50	134.75
70	(6)	187	5.5335963	5.533596	0.00000	1	4	217	278	157.10	0.50	157.60
70	(7)	209	5.5028315	5.502831	0.00000	1	12	224	221	215.82	1.26	217.08
70	(8)	174	5.4806493	5.480649	0.00000	1	17	196	203	195.90	1.34	197.24
70	(9)	154	5.4721643	5.472164	0.00000	1	7	173	204	116.34	0.46	116.80
70	(10)	155	5.5203690	5.520369	0.00000	1	5	180	170	161.96	0.37	162.33
70	(11)	161	5.7173389	5.717339	0.00000	1	14	190	198	126.10	0.77	126.87
70	(12)	149	5.5228303	5.522830	0.00000	1	8	173	207	104.58	0.45	105.03
70	(13)	151	5.4444504	5.444450	0.00000	1	7	176	238	116.42	0.46	116.88
70	(14)	151	5.3521113	5.352111	0.00000	1	5	169	152	81.15	0.27	81.42
70	(15)	197	5.5198241	5.519824	0.00000	1	4	219	198	218.90	0.44	219.34
80	(1)	224	6.2574180	6.257418	0.00000	1	21	248	239	246.09	2.17	248.26
80	(2)	189	5.6953971	5.695397	0.00000	1	21	220	222	148.93	1.02	149.95
80	(3)	214	5.8724801	5.872201	0.00476	1	6	243	206	250.08	1.23	251.31
80	(4)	208	5.6241641	5.624164	0.00000	1	6	233	209	185.26	0.97	186.23
80	(5)	163	5.7545116	5.754512	0.00000	1	3	190	208	90.21	0.52	90.73
80	(6)	163	6.1632528	6.163253	0.00000	1	3	190	165	108.56	0.32	108.88
80	(7)	209	6.0308500	6.030850	0.00000	1	8	231	228	227.23	0.71	227.94
80	(8)	219	5.9528555	5.952855	0.00000	1	10	239	206	316.42	1.04	317.46
80	(9)	242	6.1076729	6.107673	0.00000	1	18	263	264	339.62	1.35	340.97
80	(10)	186	5.7147350	5.714735	0.00000	1	5	213	213	179.76	0.50	180.26
80	(11)	220	5.7648361	5.764836	0.00000	1	16	243	214	207.76	1.32	209.08
80	(12)	171	5.6731388	5.673139	0.00000	1	13	201	192	149.09	0.51	149.60
80	(13)	184	5.9683681	5.968368	0.00000	1	57	204	242	166.99	3.73	170.72
80	(14)	217	6.1178198	6.117820	0.00000	1	6	235	188	338.83	0.53	339.36
80	(15)	183	6.1433837	6.143384	0.00000	1	10	208	259	179.01	0.73	179.74

Euclidean

N		M	Z	Z	%	Nds	LPs	Constraints		CPU seconds		
				Root	Gap			IRow	RTight	FST Gen	FST Cat	Total
70	(1)	257	6.2058863	6.205886	0.00000	1	3	1047	208	3.02	0.61	3.63
70	(2)	213	6.0928488	6.092849	0.00000	1	12	673	262	2.58	1.44	4.02
70	(3)	256	6.1934664	6.193466	0.00000	1	13	1190	198	2.48	1.45	3.93
70	(4)	215	6.2938583	6.293858	0.00000	1	16	705	312	2.60	1.69	4.29
70	(5)	251	6.2256993	6.225699	0.00000	1	30	1024	219	2.88	2.97	5.85
70	(6)	284	6.2124528	6.212453	0.00000	1	8	1504	263	3.60	1.00	4.60
70	(7)	265	6.2223666	6.222367	0.00000	1	5	1263	189	3.21	0.76	3.97
70	(8)	263	6.1872849	6.187285	0.00000	1	39	1124	223	3.32	4.84	8.16
70	(9)	237	6.2986133	6.297066	0.02457	1	4	900	319	2.82	0.90	3.72
70	(10)	214	6.2511830	6.249459	0.02757	1	7	711	203	2.25	0.61	2.86
70	(11)	277	6.6455760	6.643072	0.03768	1	14	1140	262	3.83	1.72	5.55
70	(12)	232	6.3047132	6.304713	0.00000	1	17	945	259	2.36	1.35	3.71
70	(13)	212	6.2912258	6.291226	0.00000	1	5	822	177	2.31	0.57	2.88
70	(14)	219	6.0411124	6.041112	0.00000	1	3	751	202	2.48	0.43	2.91
70	(15)	303	6.2318458	6.231846	0.00000	1	3	1420	182	4.11	0.66	4.77
80	(1)	278	7.0927442	7.092744	0.00000	1	12	1130	222	4.03	1.57	5.60
80	(2)	272	6.5273810	6.527381	0.00000	1	6	952	242	3.91	0.85	4.76
80	(3)	286	6.5332546	6.533255	0.00000	1	14	1266	394	3.66	2.67	6.33
80	(4)	305	6.4193446	6.419345	0.00000	1	15	1337	247	3.65	1.48	5.13
80	(5)	260	6.6350529	6.634241	0.01224	1	7	980	231	3.42	1.05	4.47
80	(6)	247	7.1007444	7.100744	0.00000	1	5	817	255	3.51	0.72	4.23
80	(7)	335	6.8228475	6.822847	0.00000	1	3	1717	237	4.90	0.93	5.83
80	(8)	324	6.7452377	6.745238	0.00000	1	6	1529	197	7.23	1.04	8.27
80	(9)	328	6.9825651	6.977550	0.07183	1	21	1712	242	4.25	2.28	6.53
80	(10)	242	6.5497988	6.549799	0.00000	1	9	841	272	3.47	1.48	4.95
80	(11)	269	6.6283099	6.628310	0.00000	1	10	1073	277	3.66	1.51	5.17
80	(12)	242	6.5070089	6.507009	0.00000	1	46	940	251	3.43	4.80	8.23
80	(13)	315	6.8022647	6.802265	0.00000	1	23	1500	220	5.24	2.75	7.99
80	(14)	293	7.0077902	7.007790	0.00000	1	8	1300	223	4.67	1.18	5.85
80	(15)	280	6.9939071	6.984967	0.12783	2	7	1244	364	4.16	2.02	6.18

Rectilinear

Table B.6: Results for OR-library problems 70–80 points.

N		M	Z	Z	%	Nds	LPs	Constraints		CPU seconds		
				Root	Gap			IRow	RTight	FST Gen	FST Cat	Total
90	(1)	215	6.0561870	6.056187	0.00000	1	5	238	232	229.27	0.56	229.83
90	(2)	218	6.2213509	6.221351	0.00000	1	50	255	250	231.13	1.95	233.08
90	(3)	233	6.4605693	6.460569	0.00000	1	7	259	272	291.68	1.30	292.98
90	(4)	241	6.2576814	6.257681	0.00000	1	7	263	254	261.48	0.75	262.23
90	(5)	223	6.3891591	6.389159	0.00000	1	5	250	200	204.19	0.51	204.70
90	(6)	208	6.0465321	6.046532	0.00000	1	9	247	216	223.02	0.62	223.64
90	(7)	203	6.2494171	6.249417	0.00000	1	7	231	274	218.59	0.68	219.27
90	(8)	225	6.3064565	6.306456	0.00000	1	9	248	237	283.29	0.61	283.90
90	(9)	207	5.9415312	5.941531	0.00000	1	20	241	288	183.17	1.65	184.82
90	(10)	221	6.3200640	6.320064	0.00000	1	7	245	259	266.79	1.24	268.03
90	(11)	224	6.2808067	6.280807	0.00000	1	67	249	257	317.91	3.91	321.82
90	(12)	231	6.0821854	6.082185	0.00000	1	8	257	430	205.88	1.53	207.41
90	(13)	217	6.3056722	6.305672	0.00000	1	19	241	280	239.27	2.03	241.30
90	(14)	204	6.0941398	6.094140	0.00000	1	17	233	269	191.60	0.95	192.55
90	(15)	223	6.2496530	6.249653	0.00000	1	21	252	227	291.25	1.70	292.95
100	(1)	273	6.3942560	6.394256	0.00000	1	43	303	302	443.33	2.47	445.80
100	(2)	274	6.5948121	6.594812	0.00000	1	21	301	326	564.79	1.55	566.34
100	(3)	257	6.5313471	6.531347	0.00000	1	96	289	333	450.29	10.76	461.05
100	(4)	262	6.5769774	6.576977	0.00000	1	6	297	280	417.27	0.74	418.01
100	(5)	242	6.6746878	6.674688	0.00000	1	3	265	212	351.03	0.47	351.50
100	(6)	259	6.4663684	6.466368	0.00000	1	26	291	308	483.24	1.90	485.14
100	(7)	281	6.9878635	6.987863	0.00000	1	27	310	453	721.15	2.69	723.84
100	(8)	242	6.3949711	6.394971	0.00000	1	12	271	288	363.95	0.99	364.94
100	(9)	270	6.9143211	6.914321	0.00000	1	34	297	330	425.46	2.52	427.98
100	(10)	251	6.7195108	6.719511	0.00000	1	15	281	301	386.24	1.94	388.18
100	(11)	253	6.8329509	6.832951	0.00000	1	10	279	266	316.65	1.37	318.02
100	(12)	234	6.6706226	6.670623	0.00000	1	4	260	231	227.46	0.50	227.96
100	(13)	237	6.5052527	6.505253	0.00000	1	16	275	262	268.11	0.90	269.01
100	(14)	262	6.8825985	6.882599	0.00000	1	54	291	327	372.09	4.81	376.90
100	(15)	262	6.2051489	6.205149	0.00000	1	8	290	250	506.10	1.04	507.14

Euclidean

N		M	Z	Z	%	Nds	LPs	Constraints		CPU seconds		
				Root	Gap			IRow	RTight	FST Gen	FST Cat	Total
90	(1)	277	6.8350357	6.835036	0.00000	1	4	1017	238	4.59	0.73	5.32
90	(2)	279	7.1294845	7.129485	0.00000	1	28	923	284	5.16	3.19	8.35
90	(3)	375	7.4817473	7.481114	0.00847	1	28	1778	309	7.06	4.65	11.71
90	(4)	304	7.0910063	7.091006	0.00000	1	10	1125	303	5.07	1.56	6.63
90	(5)	290	7.1831224	7.183122	0.00000	1	4	931	288	5.12	1.03	6.15
90	(6)	324	6.8640346	6.864035	0.00000	1	5	1222	374	5.80	2.16	7.96
90	(7)	280	7.2036885	7.201542	0.02980	1	9	886	278	5.14	1.18	6.32
90	(8)	325	7.2341668	7.234167	0.00000	1	21	1402	386	5.82	2.38	8.20
90	(9)	323	6.7856007	6.782613	0.04402	3	17	1314	303	5.50	3.32	8.82
90	(10)	345	7.2310409	7.231041	0.00000	1	10	1476	349	5.85	2.15	8.00
90	(11)	387	7.2310039	7.227386	0.05003	2	93	2438	303	6.39	18.32	24.71
90	(12)	318	6.9367257	6.936726	0.00000	1	16	1131	300	5.03	2.58	7.61
90	(13)	320	7.2810663	7.278959	0.02894	2	5	1412	259	5.97	1.37	7.34
90	(14)	242	6.9188992	6.918899	0.00000	1	3	719	243	3.85	0.50	4.35
90	(15)	331	7.1778294	7.177251	0.00806	3	16	1820	229	5.77	2.88	8.65
100	(1)	384	7.2522165	7.252217	0.00000	1	25	1514	426	9.72	5.38	15.10
100	(2)	484	7.5176630	7.517663	0.00000	1	28	2901	316	10.36	4.24	14.60
100	(3)	315	7.2746006	7.274601	0.00000	1	21	1285	305	6.35	3.10	9.45
100	(4)	336	7.4342392	7.434239	0.00000	1	12	1418	354	7.50	2.13	9.63
100	(5)	319	7.5670198	7.567020	0.00000	1	6	1158	278	6.66	1.31	7.97
100	(6)	475	7.4414990	7.441499	0.00000	1	8	2864	280	9.80	3.23	13.03
100	(7)	471	7.7740576	7.774058	0.00000	1	43	2896	330	8.92	7.42	16.34
100	(8)	348	7.3033178	7.303253	0.00088	1	19	1482	333	7.40	3.77	11.17
100	(9)	385	7.7952027	7.795203	0.00000	1	7	1676	384	8.02	3.19	11.21
100	(10)	356	7.5952202	7.595220	0.00000	1	17	1518	298	7.31	3.38	10.69
100	(11)	339	7.8674859	7.858919	0.10890	3	30	1324	278	7.01	3.21	10.22
100	(12)	353	7.6131099	7.613110	0.00000	1	20	1276	326	6.60	2.21	8.81
100	(13)	383	7.4604990	7.460499	0.00000	1	11	1862	429	8.69	3.84	12.53
100	(14)	318	7.8632795	7.859664	0.04598	2	63	1098	387	6.92	8.09	15.01
100	(15)	346	7.0446493	7.044649	0.00000	1	17	1357	475	6.84	3.07	9.91

Rectilinear

Table B.7: Results for OR-library problems 90–100 points.

N		M	Z	Z	%	Nds	LPs	Constraints		CPU seconds		
				Root	Gap			IRow	RTight	FST Gen	FST Cat	Total
250	(1)	631	10.2787493	10.278749	0.00000	1	83	698	804	2688.63	15.50	2704.13
250	(2)	587	10.1096283	10.109628	0.00000	1	50	669	695	1953.25	10.25	1963.50
250	(3)	647	10.0509392	10.050939	0.00000	1	143	717	750	2655.05	21.28	2676.33
250	(4)	618	10.3914471	10.391447	0.00000	1	66	685	770	2460.38	11.28	2471.66
250	(5)	598	10.2411179	10.241118	0.00000	1	157	672	667	2195.97	47.80	2243.77
250	(6)	584	10.2291717	10.228884	0.00281	1	88	655	730	2120.24	18.74	2138.98
250	(7)	609	10.1349385	10.134938	0.00000	1	17	681	802	2252.24	5.50	2257.74
250	(8)	727	10.2988195	10.298820	0.00000	1	34	782	810	4559.43	10.11	4569.54
250	(9)	608	10.3120414	10.312041	0.00000	1	74	679	722	2663.48	12.32	2675.80
250	(10)	663	10.2468534	10.246820	0.00033	1	99	728	758	3482.97	23.79	3506.76
250	(11)	664	9.8837981	9.883798	0.00000	1	11	727	695	2801.24	5.52	2806.76
250	(12)	614	10.4839791	10.483979	0.00000	1	97	683	713	2362.36	17.62	2379.98
250	(13)	707	10.1528736	10.152874	0.00000	1	104	775	730	3452.65	21.22	3473.87
250	(14)	652	10.2689834	10.268926	0.00056	2	112	720	792	2455.02	20.86	2475.88
250	(15)	656	10.1536571	10.153657	0.00000	1	290	719	723	2838.96	42.96	2881.92
500	(1)	1415	14.3223762	14.322376	0.00000	1	237	1557	1590	12273.70	126.22	12399.92
500	(2)	1363	14.1981990	14.197574	0.00440	1	109	1490	1461	15082.95	94.16	15177.11
500	(3)	1426	14.3055601	14.305560	0.00000	1	715	1541	1600	14704.57	1909.63	16614.20
500	(4)	1312	14.4213326	14.421299	0.00023	2	191	1444	1621	12813.63	149.95	12963.58
500	(5)	1223	14.0810105	14.081010	0.00000	1	312	1364	1513	9226.70	432.37	9659.07
500	(6)	1361	14.5338846	14.533885	0.00000	1	269	1499	1576	14600.33	290.12	14890.45
500	(7)	1278	14.0592955	14.059295	0.00000	1	114	1411	1616	10677.15	43.52	10720.67
500	(8)	1258	14.1537270	14.153727	0.00000	1	159	1389	1492	10828.92	137.83	10966.75
500	(9)	1350	14.1968520	14.196852	0.00000	1	83	1465	1575	12253.89	61.63	12315.52
500	(10)	1359	13.6601144	13.660114	0.00000	1	10	709	713	14128.39	3.64	14132.03
500	(11)	1347	14.1774204	14.176406	0.00716	1	414	1481	1432	14275.90	1415.12	15691.02
500	(12)	1265	14.3975974	14.397597	0.00000	1	302	1402	1545	11687.33	847.36	12534.69
500	(13)	1211	14.1404526	14.140453	0.00000	1	176	1354	1596	8833.20	128.71	8961.91
500	(14)	1487	14.6511697	14.651170	0.00000	1	135	1610	1508	16745.81	148.53	16894.34
500	(15)	1325	14.1109532	14.110953	0.00000	1	812	1448	1504	11792.11	1107.19	12899.30

Euclidean

N		M	Z	Z	%	Nds	LPs	Constraints		CPU seconds		
				Root	Gap			IRow	RTight	FST Gen	FST Cat	Total
250	(1)	912	11.6609813	11.660981	0.00000	1	22	4021	796	123.99	11.55	135.54
250	(2)	877	11.5150079	11.514005	0.00871	3	62	3706	1003	108.04	32.26	140.30
250	(3)	838	11.4650399	11.465040	0.00000	1	127	3187	874	130.94	33.38	164.32
250	(4)	899	11.7819530	11.780478	0.01252	1	38	3925	788	128.76	23.87	152.63
250	(5)	902	11.6927089	11.692709	0.00000	1	313	3664	665	115.34	67.22	182.56
250	(6)	868	11.6256250	11.625306	0.00275	2	117	3529	853	103.40	60.98	164.38
250	(7)	880	11.5277351	11.527735	0.00000	1	38	3556	861	106.07	25.01	131.08
250	(8)	1085	11.6833323	11.677163	0.05280	5	33	6126	846	125.20	25.89	151.09
250	(9)	891	11.6821988	11.682199	0.00000	1	437	3490	800	100.07	220.25	320.32
250	(10)	1115	11.6857628	11.678762	0.05991	3	78	6876	816	127.41	69.76	197.17
250	(11)	980	11.2889613	11.287079	0.01668	5	46	4940	758	110.62	30.11	140.73
250	(12)	919	11.9035256	11.902872	0.00549	2	71	3961	963	116.81	49.24	166.05
250	(13)	979	11.6049496	11.601749	0.02758	8	238	4181	786	125.85	387.51	513.36
250	(14)	940	11.6188791	11.618879	0.00000	1	12	4438	918	109.36	10.53	119.89
250	(15)	972	11.5558198	11.555820	0.00000	1	96	4661	767	120.22	27.48	147.70
500	(1)	1877	16.2978810	16.297268	0.00376	1	37	8972	1780	1137.31	88.74	1226.05
500	(2)	2175	16.0756854	16.074292	0.00866	2	45	12005	1754	1194.68	82.94	1277.62
500	(3)	2103	16.2664661	16.266466	0.00000	1	200	11408	1635	1093.94	1648.71	2742.65
500	(4)	1839	16.4110997	16.411100	0.00000	1	165	8570	1729	1123.53	479.20	1602.73
500	(5)	1825	16.0586161	16.053088	0.03443	9	458	7821	1560	1044.50	3149.40	4193.90
500	(6)	2023	16.4685074	16.468507	0.00000	1	32	10252	1753	1148.43	96.79	1245.22
500	(7)	1900	16.0124233	16.011407	0.00635	1	31	8313	1837	1096.92	51.36	1148.28
500	(8)	1979	16.1248138	16.124644	0.00105	1	181	9278	1694	1139.57	1327.37	2466.94
500	(9)	1925	16.2100435	16.207268	0.01712	5	40	9153	1577	1123.47	94.28	1217.75
500	(10)	1880	15.5581203	15.558120	0.00000	1	76	8143	1638	1331.92	203.79	1535.71
500	(11)	2018	16.1674316	16.167418	0.00008	1	390	9309	1767	1207.85	2702.49	3910.34
500	(12)	1841	16.4009591	16.400625	0.00204	3	74	8547	1570	1079.17	267.36	1346.53
500	(13)	1813	16.1324201	16.131619	0.00497	3	54	7190	1739	1053.40	171.74	1225.14
500	(14)	2041	16.5984329	16.592090	0.03821	3	122	9497	1579	1159.58	601.91	1761.49
500	(15)	1894	16.0758467	16.074649	0.00745	3	53	9045	1632	1099.43	72.32	1171.75

Rectilinear

Table B.8: Results for OR-library problems 250–500 points.

N		M	Z	Z Root	% Gap	Nds	LPs	Constraints		CPU seconds		
								IRow	RTight	FST Gen	FST Cat	Total
1000	(1)	2519	20.2375147	20.237515	0.00000	1	207	2817	3070	54378.15	290.64	54668.79
1000	(2)	2628	20.0770115	20.077011	0.00000	1	403	2906	3480	55014.51	829.35	55843.86
1000	(3)	2545	19.9644390	19.964439	0.00000	1	128	2788	3318	55826.65	248.59	56075.24
1000	(4)	2787	20.2341007	20.234101	0.00000	1	285	3027	3316	62346.85	410.28	62757.13
1000	(5)	2548	20.0592614	20.059261	0.00000	1	221	2809	3230	57033.21	376.83	57410.04
1000	(6)	2639	20.2982354	20.298235	0.00000	1	736	2895	2875	58181.42	26271.00	84452.42
1000	(7)	2538	20.2735687	20.273429	0.00069	2	259	2812	2973	50935.55	2013.17	52948.72
1000	(8)	2618	20.2179823	20.217400	0.00288	3	2995	2863	2997	59912.45	212605.95	272518.40
1000	(9)	2735	20.0901054	20.090105	0.00000	1	169	2981	3137	66569.00	679.66	67248.66
1000	(10)	2582	20.1299493	20.129949	0.00000	1	84	2839	3169	50642.13	194.93	50837.06
1000	(11)	2626	20.3131596	20.313160	0.00000	1	417	2886	3363	56819.50	850.54	57670.04
1000	(12)	2751	20.3558789	20.355879	0.00000	1	434	3003	3207	70516.06	1643.22	72159.28
1000	(13)	2575	19.9929902	19.992990	0.00000	1	473	2823	3370	47915.31	2586.08	50501.39
1000	(14)	2633	20.5686689	20.568669	0.00000	1	1188	2939	3091	64688.76	8444.38	73133.14
1000	(15)	2650	20.1739736	20.173974	0.00000	1	641	2909	3212	59522.88	4237.71	63760.59

Euclidean

N		M	Z	Z Root	% Gap	Nds	LPs	Constraints		CPU seconds		
								IRow	RTight	FST Gen	FST Cat	Total
1000	(1)	4047	23.0535806	23.042695	0.04722	15	190	20966	3418	13091.26	2442.62	15533.88
1000	(2)	3883	22.7886471	22.788544	0.00045	1	123	17112	3455	11378.59	1089.12	12467.71
1000	(3)	3978	22.7807756	22.780639	0.00060	2	1077	19670	3332	12026.53	40275.44	52301.97
1000	(4)	3983	23.0200846	23.017442	0.01148	6	552	18309	3556	12076.06	2410.22	14486.28
1000	(5)	3916	22.8330602	22.832172	0.00389	2	904	18869	3156	12023.65	35497.08	47520.73
1000	(6)	4138	23.1028456	23.095362	0.03239	19	3287	21908	3160	12413.24	374356.59	386769.83
1000	(7)	3916	23.0945623	23.093270	0.00560	2	1050	19344	3416	11289.35	42697.75	53987.10
1000	(8)	4173	23.0639115	23.062650	0.00547	5	782	23043	3198	11390.31	46974.55	58364.86
1000	(9)	4355	22.7745838	22.773655	0.00408	6	112	25468	3209	16288.00	1074.69	17362.69
1000	(10)	3886	22.9267101	22.923663	0.01329	5	852	17704	3311	11648.66	45708.86	57357.52
1000	(11)	3884	23.1605619	23.157667	0.01250	6	593	18739	3631	11685.70	2143.10	13828.80
1000	(12)	4564	23.0904712	23.088224	0.00973	5	1597	27747	3389	14804.49	258108.23	272912.72
1000	(13)	3782	22.8031092	22.803109	0.00000	1	1837	18229	3280	11603.79	165687.02	177290.81
1000	(14)	4173	23.4318491	23.426697	0.02199	15	2298	21471	3322	12112.12	364853.03	376965.15
1000	(15)	4011	22.9965775	22.994263	0.01006	2	280	19768	3263	12641.83	2446.81	15088.64

Rectilinear

Table B.9: Results for OR-library problems 1000 points.

Bibliography

[1] M. L. Balinski. On a selection problem. *Management Sci.*, 17:230–231, 1970.

[2] J. E. Beasley. Or-library — a collection of data sets for a variety of or problems.

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

[4] C. Berge. *Graphs and Hypergraphs.* North-Holland, Amsterdam, Netherlands, 1973.

[5] P. Berman and V. Ramaiyer. Improved approximations for the Steiner tree problem. In *Proceedings of the Third Symposium on Discrete Algorithms*, pages 325–334, 1992.

[6] C. W. Borchardt. *Journal f. d. reine und angewandte Math.*, 57:111–121, 1860.

[7] W. M. Boyce and J. E. Seery. STEINER 72: An improved version of Cockayne and Schiller's program STEINER for the minimal network problem. Technical Report 35, Bell Laboratories, Murray Hill, New Jersey, 1973.

[8] A. Cayley. *Collected Mathematical Papers of A. Cayley*, volume 13. 1889.

[9] E. J. Cockayne. On the Steiner problem. *Canadian Mathematical Bulletin*, 10:431–450, 1967.

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

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

[12] E. J. Cockayne and D. G. Schiller. Computation of a Steiner minimal tree. In D. J. A. Welsh and D. R. Woodall, editors, *Combinatorics*, pages 53–71. Inst. Math. Appl., 1972.

[13] R. Courant and H. Robbins. *What is Mathematics?* Oxford University Press, New York, 1941.

[14] G. B. Dantzig and B. C. Eaves. Fourier-Motzkin elimination and its dual. *Journal of Combinatorial Theory (A)*, 14:288–297, 1973.

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

[16] U. Fößmeier and M. Kaufmann. On exact solutions for the rectilinear Steiner problem, part i: Theoretical results. Technical Report WSI-96-09, Universität Tübingen, Germany, 1996.

[17] J. L. Ganley. *Geometric Interconnection and Placement Algorithms.* PhD thesis, Department of Computer Science, University of Virginia, Charlottesville, Virginia, 1995.

[18] 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.

[19] 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.

[20] 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.

[21] M. R. Garey and D. S. Johnson. The rectilinear Steiner tree problem is NP-complete. *SIAM Journal on Applied Mathematics*, 32:826–834, 1977.

[22] R. L. Graham, D. E. Knuth, and O. Patashnik. *Concrete Mathematics*. Addison-Wesley, Reading, Massachusetts, 1990.

[23] M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.

[24] M. Grötschel, L. Lovász, and A. Schrijver. Corregendum to our paper "the ellipsoid method and its consequences in combinatorial optimization". *Combinatorica*, 4(4):291–295, 1984.

[25] S. L. Hakimi. Steiner's problem in graphs and its implications. *Networks*, 1:113–133, 1971.

[26] M. Hanan. On Steiner's problem with rectilinear distance. *SIAM Journal on Applied Mathematics*, 14:255–265, 1966.

[27] F. K. Hwang. On Steiner minimal trees with rectilinear distance. *SIAM Journal on Applied Mathematics*, 30:104–114, 1976.

[28] F. K. Hwang. A linear time algorithm for full Steiner trees. *Operations Research Letters*, 4:235–237, 1986.

[29] 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.

[30] F. K. Hwang and J. F. Weng. The shortest network under a given topology. *Journal of Algorithms*, 13(3):468–488, 1992.

[31] V. Jarník and O. Kössler. O minimálních grafech obsahujících n daných bodu. *Ĉas. Pêstování Mat.*, 63:223–235, 1934.

[32] 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.

[33] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.

[34] T. Koch and A. Martin. Solving Steiner tree problems in graphs to optimality. Technical Report SC 96–42, Konrad-Zuse-Zentrum für Informationstechnik, Berlin, Germany, 1996.

[35] J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proc. Amer. Math. Soc.*, 7:48–56, 1956.

[36] 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.

[37] L. Lovász. *Combinatorial Problems and Exercises.* North-Holland, Amsterdam, Netherlands, 1979.

[38] A. Lucena and J. E. Beasley. Branch and cut algorithms. In J. E. Beasley, editor, *Advances in Linear Integer Programming.* Oxford University Press, 1996.

[39] T. McCormick. Personal communication, September 1997.

[40] Z. A. Melzak. On the problem of Steiner. *Canadian Mathematics Bulletin*, 4:143–149, 1961.

[41] M. Padberg and L. Wolsey. Trees and cuts. *Annals of Discrete Mathematics*, 17, 1983.

[42] J.-C. Picard. Maximal closure of a graph and application to combinatorial problems. *Management Sci.*, 22:1268–1272, 1976.

[43] J.-C. Picard and M. Queyranne. A network flow solution to some nonlinear $0-1$ programming problems, with applications to graph theory. *Networks*, pages 141–159, 1982.

[44] H. Prüfer. *Arch. Math. u. Phys.*, 27:142–144, 1918.

[45] M. Queyranne. Personal communication, May 1997.

[46] G. Reinelt. TSPLIB - a traveling salesman problem library. *ORSA Journal on Computing*, 3(4):376–384, 1991.

[47] J. M. W. Rhys. A selection problem of shared fixed costs and network flows. *Management Sci.*, 17:200–207, 1970.

[48] J. S. Salowe and D. M. Warme. An exact rectilinear Steiner tree algorithm. In *Proceedings of the International Conference on Computer Design*, pages 472–475, 1993.

[49] J. S. Salowe and D. M. Warme. 35-point rectilinear Steiner minimal trees in a day. *Networks*, 25:69–87, 1995.

[50] A. F. Sidorenko. On minimal rectilinear Steiner trees. *Diskretnaya Matematika*, 1:28–37, 1989. (In Russian).

[51] W. D. Smith. Personal communication, February 1998.

[52] J. Soukup and W. F. Chow. Set of test problems for the minimum length connection networks. *ACM/SIGMAP Newsletter*, 15:45–81, 1973.

[53] R. P. Stanley. *Enumerative Combinatorics*, volume 1. Wadsworth Brooks Cole, 1986.

[54] 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.

[55] I. Tomescu and M. Zimand. Minimum spanning hypertrees. *Discrete Applied Mathematics*, 54:67–76, 1994.

[56] D. Trietsch and F. K. Hwang. An improved algorithm for Steiner trees. *SIAM Journal on Applied Mathematics*, 50:244–263, 1990.

[57] R. A. Wagner. Evaluating uniform expressions within two steps of minimum parallel time. *Journal of the ACM*, 44(2):345–361, 1997.

[58] D. M. Warme, P. Winter, and M. Zachariasen. Exact algorithms for plane Steiner tree problems: A computational study. In D.-Z. Du, J. M. Smith, and J. H. Rubinstein, editors, *Advances in Steiner Trees*. Kluwer Academic Publishers, Norwell, Massachusetts, 1998.

[59] H. S. Wilf. *Generatingfunctionology*. Academic Press, San Diego, California, second edition, 1994.

[60] P. Winter. An algorithm for the Steiner problem in the Euclidean plane. *Networks*, 15:323–345, 1985.

[61] P. Winter. Reductions for the rectilinear Steiner tree problem. manuscript, 1994.

[62] P. Winter and M. Zachariasen. Euclidean Steiner minimum trees: an improved exact algorithm. *Networks*, 30:149–166, 1997.

[63] 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.

[64] M. Zachariasen. Rectilinear full Steiner tree generation. Technical Report DIKU–TR–97/29, University of Copenhagen, Universitetsparken 1, DK–2100 KBH Ø, Denmark, 1997.