Author/s:
Lee, Jae Myeong

Title:
Exact Solutions for k-Steiner Tree Problems

Date:
2025

Persistent Link:
https://hdl.handle.net/11343/361439

Exact Solutions for k-Steiner Tree Problems

by

Jae Myeong Lee

ORCID: 0000-0003-4941-2420

A thesis submitted in total fulfillment for the
degree of Doctor of Philosophy

in the
Faculty of Science
School of Mathematics and Statistics
THE UNIVERSITY OF MELBOURNE

November 2025

THE UNIVERSITY OF MELBOURNE

Abstract

Faculty of Science
School of Mathematics and Statistics

Doctor of Philosophy

by Jae Myeong Lee
ORCID: 0000-0003-4941-2420

Steiner tree problems in the plane have been extensively studied as fundamental models in the field of network design. They model a wide range of applications in areas such as VLSI design, wireless communication, and transportation networks. Steiner points enable the design of more efficient networks, either by reducing their length or minimising the distance between nodes. However, Steiner points typically represent hubs or intersections, which may incur additional costs. To address this, variations of the Steiner tree problem impose bounds on the number of Steiner points, resulting in the minimum k-Steiner tree problem, where k is the bound on the number of Steiner points. This problem is NP-hard. While several theoretical results exist, the implementation of exact algorithms has remained largely unexplored.

In this thesis, we develop exact algorithms that solve the Euclidean minimum k-Steiner tree problem for two different objectives: minimising the total Euclidean length of a network, and minimising the Euclidean length of the longest edge in a network. To mitigate the combinatorial proliferation of potentially optimal topologies, we develop what are referred to as pruning tests, that eliminate suboptimal components by leveraging geometric and structural properties of optimal solutions. To our knowledge, these are the first exact algorithms capable of solving randomly generated instances of these problems. We demonstrate their effectiveness through computational experiments, showing that they can reliably solve the given instances.

Declaration of Authorship

I, Jae Myeong Lee, declare that this thesis titled, 'Exact Solutions for k-Steiner Tree Problems' and the work presented in it are my own. I confirm that:

- The thesis comprises only my original work towards the Doctor of Philosophy except where indicated in the preface;
- due acknowledgement has been made in the text to all other material used; and
- the thesis is fewer than the maximum word limit in length, exclusive of tables, maps, bibliographies and appendices as approved by the Research Higher Degrees Committee.

Signed: _____

Date: _____

Preface

Funding was received through the Australian Government Research Training Program Scholarship from The University of Melbourne. Travel funding was received through the travel grant scheme of the University of Melbourne. This research was also supported by The University of Melbourne's Research Computing Services and Optimisation Technologies, Integrated Methodologies and Applications (OPTIMA).

List of publications

Paper 1

Marcus Brazil, Michael Hendriksen, Jae Lee, Michael S Payne, Charl Ras, and Doreen Anne Thomas. An exact algorithm for the Euclidean k-Steiner tree problem. *Computational Geometry*, 121:102099, 2024. [1]

This paper provides the content for Chapters 2 and 3.

Paper 2

Jae Lee, Marcus Brazil, Charl Ras and Doreen Anne Thomas, An Improved Exact Algorithm for the Euclidean k-Steiner Tree Problem (paper has been submitted for journal publication).

This paper provides the content for Chapters 2 and 3.

Paper 3

The content for Chapter 4 will be submitted in July 2025 as a third paper.

Acknowledgements

This thesis would not have been possible without the help of my supervisors Associate Professor Charl Ras, Associate Professor Marcus Brazil and Professor Emeritus Doreen Anne Thomas. I am grateful for your unwavering support during my candidature and I genuinely believe that none of this would have been possible without any of you. To Charl, whom I have worked with since my Master's research project, thank you for being a mentor not only during my Master's and PhD, but also beyond the research itself. I greatly appreciated your support and encouragement during challenging times. To Marcus, your willingness to help along with your expertise in the field has been immensely helpful. I have always felt I was in safe hands thanks to your keen eye for detail. To Doreen, thank you for being so warm and accommodating ever since our first meeting. I thoroughly enjoyed our time in Denmark and would like to thank you for your guidance at the EURO conference. I am honoured to be your last student. I would also like to thank my Committee Chair, Dr Kate Smith-Miles for keeping me on track during my progress reviews. Your external perspective and suggestions provided valuable insight and helped keep me grounded throughout the project.

I would like to thank Michael Payne and Michael Hendriksen for their contribution to our paper, [1]. To Michael (Hendriksen), thank you for sharing your experiences, both positive and negative, from your own PhD journey. I greatly appreciated having someone relatable to talk to about my own experiences.

Thank you Nico, for your help with implementing some pruning tests. Not only were you very open to sharing your findings, but you also made time for spontaneous meetings and patiently helped me troubleshoot the code.

I would also like to thank all my friends for their support, and for providing laughter and positivity when I needed it most.

Finally, I would like to thank my family for always being there throughout my PhD journey. Without your unwavering support, I do not believe I would have had the resilience to see this through.

Contents

Abstract i

Declaration of Authorship ii

Preface iii

Acknowledgements iv

List of Figures vii

List of Tables x

1 Background 1
- 1.1 Contribution and referenced papers . . . 7
- 1.2 Thesis Outline . . . 8

2 Geometric and structural properties of a minimum k-Steiner Tree 9
- 2.1 Preliminaries . . . 9
- 2.2 Geometric and structural properties . . . 15
  - 2.2.1 The edge-lune theorem . . . 15
  - 2.2.2 4-lune theorem . . . 22
  - 2.2.3 Rhombus Theorem . . . 29
  - 2.2.4 Trapezium Theorem . . . 32

3 Implementation of the exact k-Steiner tree algorithm 38
- 3.1 Branch trees and branches . . . 38
- 3.2 The generation algorithm . . . 39
- 3.3 Pruning tests . . . 45
  - 3.3.1 Projection test . . . 46
  - 3.3.2 Bottleneck test . . . 51
  - 3.3.3 Lune test . . . 59
  - 3.3.4 4-lune test . . . 65
  - 3.3.5 Trapezium Test and the Rhombus Test . . . 70
- 3.4 The concatenation algorithm . . . 74
- 3.5 Experimental testing . . . 81
  - 3.5.1 Core improvements to the generation phase . . . 82

3.5.2 The effect of the new pruning tests . . . 83
3.5.3 The concatenation phase . . . 86

4 The Minimum Bottleneck k-Steiner Network Problem 88
4.1 Background and Preliminaries . . . 88
4.2 Algorithm overview . . . 94
4.3 Generation Phase . . . 95
4.3.1 Branches . . . 96
4.3.2 Merging . . . 98
4.3.3 Beading . . . 100
4.3.4 Termination . . . 101
4.3.5 Optimally embedding a cluster topology . . . 102
4.3.6 Pruning tests . . . 104
4.3.6.1 Lune Test . . . 104
4.3.6.2 Cross Lune Test . . . 104
4.3.6.3 Acute Angle Test . . . 105
4.3.6.4 Convex Hull Test . . . 106
4.3.7 Calculating $b_{\max}$ . . . 107
4.3.8 Formal description of the generation phase . . . 110
4.4 The ILP phase . . . 117
4.4.1 Flow formulation . . . 117
4.4.2 Edge-cut constraints . . . 119
4.4.3 Comparison . . . 120
4.5 Experiments . . . 121
4.5.1 Algorithm performance . . . 121
4.5.2 Effectiveness of pruning tests . . . 122

5 Conclusion 126
5.1 Summary of contributions . . . 126
5.2 Future directions . . . 127

A Adjacent degree-4 Steiner points conjecture 129

Bibliography 133

List of Figures

2.1 Minimum k-Steiner trees for $k = 0, 1, 2$ on the set of terminals N. 11
2.2 Replacing t_1, t_2 and s by equilateral point x results in an FST of the same length as before replacement. Node x is a pseudoteriminal, and plays the role of a terminal in the reduced network in (b). 14
2.3 Replacing t_1, t_2, t_3 and s with a pseudoterminal $x(= t_3)$ results in an FST that has a total length that is $|t_1t_2|$ shorter than before replacement. . . . 14
2.4 The lune $L(u, v)$. 15
2.5 Edge t_1t_3 of the MST (0-Steiner tree) intersects the lune $L(t_1, t_2)$. 16
2.6 Proposition 2.2.1 states that $C(u) \subseteq C'(u)$ and $C(v) \subseteq C(v')$. 17
2.7 Case 1 : $x \in C(a) \setminus C(b)$ and $y \in C(b) \setminus C(a)$. 18
2.8 Lines L_1 and L_2 divide the plane into three partitions P_1, P_2 and P_3. . . . 20
2.9 The point x must lie in the same open half-plane defined by L_3 as $L(a, b)$. 21
2.10 Lines L_4 and L_5 divide the plane into three partitions P_a, P_b and P_c. . . . 22
2.11 A minimum Steiner tree S_d that interconnects the vertices a, b and c. . . 23
2.12 An example where t lies in the same component as c after the removal of the edges in T. Replacing the edges in T with the edges in S_d and the edge td results in a shorter k-Steiner network if $|S_d| + |dt| < |T|$, which is denoted by an open disc with centre d. 24
2.13 The vertex c lies on the line segment c_1c_2. 26
2.14 The shaded region depicts the set of points that are at least $|e_{ad}b| - |bd|$ units closer to p than to a, for some positive value of $|e_{ad}b| - |bd|$. 28
2.15 The blue circle shows the boundary of the points satisfying Inequality 1 when $c = c_1$ or $c = c_2$. The red circle shows the boundary of the points satisfying the inequality when $c = (0.5, 0)$. 28
2.16 Constructions for the proof of Lemma 2.2.6. 29
2.17 The Steiner tree $\mathcal{C}_x$. 30
2.18 An example of the hyperbolae H_1 and H_2 constructed during the proof of Theorem 2.2.7. 31
2.19 Vertex p is confined to the region A. 33
2.20 Vertex q is confined to the region $B(u)$. 34
2.21 An illustration of Theorem 2.2.7 and Theorem 2.2.8. 37

3.1 Two branches $\mathcal{B}_1$ and $\mathcal{B}_2$ are double-merged. The new pseudoterminal p is not shown here, but lies below the figure on the line containing the stem. 40
3.2 An example of a termination merge. 41
3.3 Three branches $\mathcal{B}_1$, $\mathcal{B}_2$ and $\mathcal{B}_3$ are triple-merged, with $\mathcal{B}_3$ as the source branch. 42
3.4 If $\overline{l(A_1)r(A_1)}$ intersects $\widehat{p_2p_1}$ twice, the branch is not optimal. 47
3.5 A figure depicting the subarc A'_3 of A_3. 48

3.6 The three subcases for Case 1. . . . 49
3.7 Angle $\angle l(A_3)ar(A_3)$ must be at least $2\pi/3$ since A_3 is a Steiner arc. . . . 50
3.8 Given $l(A_3) \notin T$, we can find some $l(A_3)' \in T \cap \widehat{l(A_3)a}$. . . . 50
3.9 In the branch tree shown, any path from a terminal in $N_1 = \{t_1, t_2\}$ to a terminal in $N_3 = \{t_3, t_4, t_5, t_6\}$ contains the edges s_1s and s_3s. . . . 53
3.10 Functions $h(\theta)$ and $c(\theta)$ are defined to be $|sp_3|$ and $|s_3p_3|$ respectively. . . . 54
3.11 The points s_3^r and s_3^l project onto the points $r(A)$ and $l(A)$ on the Steiner segment A from p_3. . . . 55
3.12 An example of bisection partition and standard bisection being used in tandem. . . . 56
3.13 . . . 61
3.14 Terminal t_a lies on the arc that passes through a as $\angle t_a s_1 s_2 < \pi/3$ while terminal t_b lies on the arc that passes through s_1 as $\angle t_b s_1 s_2 > \pi/3$. . . . 62
3.15 An example where the root s_3 is a terminal. Hence, if any of the three inequalities listed above hold for $s' = l(A')$, $r(A')$ and $t(A')$, then they must hold for all $s' \in A'$. . . . 67
3.16 An example where the root s_3 is a terminal. If an equality listed above holds for $s' = l(A')$ and $s' = r(A')$, then they must hold for all $s' \in A'$. . . . 67
3.17 The root s_3 is always fixed if $\mathcal{B}$ is not the source branch. . . . 68
3.18 A illustration demonstrating how 3-lune inequalities may be used to prune a branch when a double-merge is immediately followed by a triple-merge. 68
3.19 The region satisfying at least two 4-lune inequalities is shown in blue and the four lunes for the four edges are shown in green. . . . 70
3.20 The Steiner point $s_3 \in \widehat{l(A_3)r(A_3)}$ must lie inside the triangle T defined by vertices s_1 and s_2. We adjust A to be the projection of the part of the source Steiner curve that lies inside T. . . . 72
3.21 Examples of how the Trapezium and Rhombus Theorems are applied as pruning tests after a triple-merge has occurred. . . . 73
3.22 The new degree-4 Steiner point s' must lie on vw to satisfy the Trapezium and Rhombus Tests. . . . 74
3.23 . . . 75
3.24 If $S = \{t_3\}$, then $\delta(S) = \{\{t_2, t_3, t_4\}, \{t_3, t_5\}, \{t_3, t_4, t_5, t_6\}\}$. . . . 78
3.25 A minimum 2-Steiner tree. . . . 83
3.26 Improvement in time by including the Trapezium Test and the extended Rhombus Test for $k = 3$. . . . 84
3.27 Improvement in time by including the Trapezium Test and the extended Rhombus Test for $k = 4$. . . . 84
3.28 Improvement in time by including the Trapezium Test and the extended Rhombus Test for $k = 5$. . . . 85
3.29 The decrease in the number of FSTs with degree-4 Steiner points by using the Trapezium and extended Rhombus Tests for $k = 3$. . . . 85
3.30 The decrease in the number of FSTs with degree-4 Steiner points by using the Trapezium and extended Rhombus Tests for $k = 4$. . . . 86
3.31 The decrease in the number of FSTs with degree-4 Steiner points by using the Trapezium and extended Rhombus Tests for $k = 5$. . . . 86
3.32 The relationship between the number of FSTs and the corresponding time duration of the concatenation phase (for $k = 3, 4, 5$); note, log-scale is employed. . . . 87

4.1 An example of a canonical 5-Steiner tree. 91
4.2 A canonical 7-Steiner tree where the arrows indicate the determinators of each Steiner point. 93
4.3 Hierarchy of the clusters in the canonical 7-Steiner tree in Figure 4.2. . . 93
4.4 The leading region R_0 of a base branch β_0 with a terminal t_0 and root bud r_0. 97
4.5 Two branches β_1 and β_2 with intersecting leading regions R_1 and R_2. . . 98
4.6 The leading regions R_1 and R_2 intersect and hence branches β_1 and β_2 can be merged. 99
4.7 Merging β_1 and β_2 generates a new branch β' with leading region R'. . . . 99
4.8 The leading region R' intersects the leading region R_3 of β_3 and hence can be merged. 99
4.9 Merging β' and β_3. 99
4.10 The leading region $R = R_i^+$. 100
4.11 A branch β_i with a leaf bud r_i. 101
4.12 Branch β_i is beaded to generate a new branch β. In the process, r_i becomes a degree-2 Steiner point s, while a new leaf bud r and the edge rs are introduced. 101
4.13 Branch β with leading region $R = (R_i \cap R_j)^+$ can be terminated at t, since $t \in R$. 102
4.14 A cluster topology with four terminals, two degree-3 Steiner points and a single degree-2 Steiner point. 103
4.15 An example of a nonprimary cluster subtree that does not satisfy the cross lune test. 105
4.16 As $\triangle tt_it_j$ is obtuse, forcing the edges st, st_i and st_j to have equal lengths means that the Steiner point s does not coincide with the centre of the smallest disc containing t, t_i and t_j. 106
4.17 An example of a cluster subtree that does not satisfy the acute angle test but satisfies the convex hull test. The Steiner point s_2 does not lie inside the convex hull of s_1, t_3 and t_4 but the internal Steiner points s_1 and s_2 lie inside the convex hull of the terminals t_1, t_2, t_3 and t_4. 107
4.18 The set of base vertices N_1 contains the degree-3 Steiner points s_3 and s_4. 111
4.19 The set of base vertices N_1 contains the degree-2 Steiner points $s_5, s_6, ..., s_{10}$. 111
4.20 A cluster subtree T' with parent cluster subtrees T_i and T_j. 113
4.21 The average algorithm runtime over 20 randomly generated instances for $k = 3$. 122
4.22 The percentage of Steiner points pruned by different pruning tests for $k = 2$. 123
4.23 The percentage of Steiner points pruned by different pruning tests for $k = 3$. 124

A.1 The newly generated degree-4 Steiner point s is adjacent to another degree-4 Steiner point s_1. 130
A.2 131
A.3 The improvement in time by including the adjacent degree-4 pruning test for $k = 3$. 132

List of Tables

3.1 The average time duration (in seconds) of the generation phase for the previous algorithm and the current algorithm (with and without the Trapezium Test and the extended Rhombus Test) over 20 randomly generated instances for $k = 3$. Here, extra tests* refer to the Trapezium Test and the extended Rhombus Test. 83

4.1 ILP runtimes (in seconds) for the flow formulation and the edge-cut formulation with $k = 3$. 121
4.2 Percentage of Algorithm Time Spent in the ILP Phase for $k = 3$. 122
4.3 Percentage of Steiner points pruned by each pruning test for different number of terminals for $k = 2$. 124
4.4 Percentage of Steiner points pruned by each pruning test for different number of terminals for $k = 3$. 124

Chapter 1

Background

Networks are vital for our society to function on a daily basis. For example, transportation networks such as road networks and railways facilitate the movement of people and goods, communication networks such as optical fibre networks allow information to be conveyed instantly across large distances and utility networks, such as water supply networks and electrical networks, provide essential resources to a multitude of locations. Given the prevalence and necessity of networks in our society, it is imperative that we prioritise the efficacy and efficiency of our networks.

As a result, network design problems are among the most researched topics in mathematical optimisation. There are many well-known problems involving this topic, including the minimum spanning tree (MST) problem, where we are given a set of points and our aim is to find a shortest network interconnecting these points. Another classical problem in network design is the travelling salesman problem (TSP), where we are again given a set of points and our aim is to find a shortest route visiting every point then returning to the starting point.

The *minimum Steiner tree problem* emerged as an extension of the MST problem, as more efficient networks were desired. It is defined as follows: given a set of points (referred to as *terminals*) in the plane, find a shortest connected network spanning the terminals where any number of additional points (called *Steiner points*) may be introduced. The minimum Steiner tree problem is a classical network design problem that dates back to the Fermat-Torricelli problem in the 1600s [2]. In the original Fermat-Torricelli problem, we are given three points in the plane and the aim is to find a fourth

point that minimises the sum of the distances to the given points. Later, in 1810, a generalised version of this problem, where you are given any number of points in the plane, was proposed in *Annales de Gergonne* [2]. Furthermore, this text contained the first known statement of the minimum Steiner tree problem. The minimum Steiner tree problem was later shown [3] to be NP-hard.

Steiner trees are effective models for wireless communication networks [4] as well as a variety of physical networks [5] [6] [7]. One example is an underground mine network where tunnels connect a surface portal with places on the ore body underground. The junctions where the tunnels meet are costly to construct as they require extra reinforcement and must satisfy a range of other constraints [8]. Within communication networks, the Steiner points may correspond to junctions which model structures such as relays or hubs, which also represent additional costs.

The cost of the Steiner points, as demonstrated in these examples, motivates the problem we study in this thesis, namely the *k-Steiner tree problem*, which is defined as follows: given a set of n terminals in the plane and a non-negative integer k, find a shortest connected network spanning the terminals where at most k Steiner points may be introduced. This is a more complex general problem than the classical Steiner tree problem, since optimal topologies (which are the underlying graph structures of networks) may contain degree-4 Steiner points. In the classical Steiner tree problem, only degree-3 Steiner points can occur, which leads to simpler geometric properties.

In this thesis, we consider two different cost functions: the "min-sum" cost function, where the cost of a network is defined as the total Euclidean length of all edges in the network and the "min-max" cost function, where the cost is defined as the Euclidean length of the longest edge in the network. Firstly, we discuss problems related to the k-Steiner tree problem with a "min-sum" cost function. Later, we discuss related problems with a "min-max" cost function.

The k-Steiner tree problem with a "min-sum" cost function was first introduced by Georgakopoulos and Papadimitriou in [9]. In particular, they studied the 1-Steiner tree problem, where the number of Steiner points is limited to one. Georgakopoulos and Papadimitriou solved this problem in $O(n^2)$ time using a plane subdivision approach. Their algorithm first partitions the plane into $O(n^2)$ regions, called the overlayed oriented Voronoi diagram (OOVD). Each region is associated with a set of up to six terminals.

The optimal neighbours of any Steiner point in a region is a subset of these terminals, thereby significantly limiting the number of topologies that need to be considered. For each subset of these terminals, they calculate the optimal point which minimises the sum of distances to each terminal in this subset in constant time. Then, they calculate the resulting cost by measuring the length of an MST spanning this optimal point and the terminals. This process is repeated for each of the $O(n^2)$ regions. The shortest of these 1-Steiner trees is a minimum 1-Steiner tree. Although their algorithm has a time complexity of $O(n^2)$, the constant term is quite large (the process of finding an optimal point may be repeated up to 62 times for each of the $O(n^2)$ regions) and the authors were not able to generalise this algorithm for higher values of k.

Later, a generalised version of this algorithm capable of solving the k-Steiner tree problem was developed by Brazil et al. [10]. By building on the OOVD method of Georgakopoulos and Papadimitriou, they developed a very general approach that can handle any k value and a variety of cost functions and normed planes (including the "min-sum" and "min-max" objectives). Their algorithm solves the k-Steiner tree problem in $f(k)O(n^{2k})$ time, where $f(k)$ is a superexponential function of k. This left the question open of whether there exists a fixed parameter tractable (FPT) algorithm for the k-Steiner tree problem (i.e., an algorithm that can solve the problem in time $g(k)O(n^{O(1)})$, where g is a function of k only). As is the case for the 1-Steiner tree algorithm of Georgakopoulos and Papadimitriou, the Brazil et al. algorithm is predominantly a theoretical result. Therefore, based on the theoretical runtimes and the practical complexity of constructing the OOVD partition, implementations of these algorithms have not been pursued in the literature (and would most likely be very inefficient).

Using a similar partitioning approach, Bose et al. developed a $\Theta(n \log n)$-time algorithm that solves a variation of the 1-Steiner tree problem where the Steiner point must lie on a given line [11]. They extended this to a more general problem where the Steiner point must lie on one of j given lines, resulting in an $O(jn \log n)$-time algorithm. Finally, based on the Brazil et al. algorithm in [10], they developed an $O(n^k)$-time algorithm for the k-Steiner tree problem when all Steiner points are restricted to lie on a given line.

Variations of the k-Steiner tree problem have also been studied for the rectilinear norm. Under this norm, the distance between two points is measured by the sum of the absolute differences of their horizontal and vertical coordinates. This norm has useful applications

in VLSI design [12]. As with the Euclidean norm, the problem under the rectilinear norm is also NP-hard [13]. Kahng and Robins developed a $O(n^3)$ heuristic using an iterative 1-Steiner approach, with a performance ratio less than 3/2 [14]. The heuristic first begins with a rectilinear MST on the set of given vertices. A Steiner point which results in the largest decrease in the length of the tree is added to the set of vertices. This process is repeated iteratively (potentially up to $n-2$ times) until adding a Steiner point does not decrease the length of a tree.

We now turn our attention to the second type of cost function, the "min-max" or *bottleneck* objective. This objective gives rise to the *bottleneck k-Steiner tree problem*, which we now define. In this problem, we are given a set of n points in the plane, a non-negative integer k and our aim is to find a network interconnecting the given set of points using at most k Steiner points such that the length of the longest edge is minimised. The problem has applications in wireless communication networks [15], where we want to minimise the length of the longest connection in a network due to the limited range of each node. It has also been applied to VLSI layout design [12]. Unlike the "min-sum" version of the Steiner tree problem, the restriction on the number of Steiner points arises naturally, because without it a minimum bottleneck Steiner network can be constructed with cost arbitrarily close to zero (by employing a large number of degree-2 Steiner points).

The bottleneck k-Steiner tree problem is also NP-hard [16]. Hence, several approximation algorithms have been developed for this problem. Firstly, Wang and Du [15] proved that a polynomial-time approximation algorithm for the bottleneck k-Steiner tree problem cannot have a performance ratio less than $\sqrt{2}$. Furthermore, they provided a polynomial-time approximation algorithm with a performance ratio of 2. The algorithm begins with an MST and a Steiner point is added to the longest edge in the MST to reduce its length. This process is repeated until all k Steiner points have been used. Additionally, Du et al. presented an improved approximation algorithm with a performance ratio of $\sqrt{3}+\epsilon$ where ϵ is an arbitrary positive number [17]. In the same paper, Du et al. also presented an $O(n^3)$-time approximation algorithm for the dual problem, where given a positive number R, the aim is to find a Steiner tree with minimum number of Steiner points such that the bottleneck length (the length of a longest edge) of the tree is at most R.

There are also several existing papers on exact algorithms for the bottleneck k-Steiner

tree problem. Bae et al. developed an $O(n \log n)$-time algorithm for $k = 1$ and an $O(n^2)$-time algorithm for $k = 2$ [16]. In the algorithm for $k = 1$, they first generate an MST on the set of given points. Then, they remove up to four edges at a time and find the centre of the smallest disc containing all the resulting components. This centre becomes a potential location for the Steiner point. By repeating this method for every possible combination of one to four edges, the optimal location of the Steiner point and consequently the minimum bottleneck 1-Steiner tree, can be found. Later, by employing a type of Voronoi diagram, they expanded on this algorithm to solve the problem for general values of k in $f(k) \cdot (n^k + n \log n)$ time (where $f(k) = k^{5k}2^{O(k)}$) [18]. In terms of implementability, the algorithm of Bae et al. runs into similar issues as the OOVD approach for the "min-sum" problem. Bae et al.'s method is a theoretical result and there are no existing (nor any conceivably efficient) implementations of it.

The potential existence of an FPT algorithm for the bottleneck Steiner tree problem has been an ongoing theme in the literature. Some of these results relate to the L^p norm, which we define here for reference: given a point $x = (x_1, x_2, ..., x_d)$ in $\mathbb{R}^d$, the L^p norm of x is defined as $\|x\|_p = \left(\sum_{i=1}^{d} |x_i|^p\right)^{\frac{1}{p}}$. In particular, we define the L^∞ norm of x as $\|x\|_\infty = \lim_{p\to\infty} \left(\sum_{i=1}^{d} |x_i|^p\right)^{\frac{1}{p}}$. Bae et al. showed that the bottleneck Steiner tree problem is fixed parameter tractable in the rectilinear (L^1) norm and the L^∞ norm [18]. Recently, Bandyapadhyay et al. developed an $k^{O(k)}n^{O(1)}$-time algorithm for the Euclidean bottleneck Steiner tree problem, which proved that the problem was FPT for the L^2 norm [19]. They then generalised this result to any L^p norm, where $1 \leq p \leq \infty$.

In [20], Brazil et al. explored a variation of the bottleneck Steiner tree problem where the solution is required to be 2-connected. A graph is defined to be 2-connected if there are at least two internally disjoint paths between every pair of vertices. They developed two exact algorithms - an $O(n^2)$ and an $O(n^2 \log n)$-time algorithm for $k = 1$ and $k = 2$ respectively. In [21], Ras studied a variation of the 2-connected problem where only degree-2 Steiner points were allowed. They developed an $O(n^4k^62^k)$-time algorithm, showing that, while the problem remained NP-hard, it is in fact FPT. Furthermore, they proved that the bottleneck length of an optimal degree-2 bounded solution is at most twice that of an optimal solution without the degree constraint.

Even though there exist exact algorithms for both the minimum k-Steiner tree problem and the minimum bottleneck k-Steiner tree problem, these algorithms are impractical.

To the best of our knowledge, there are no known implementations of these algorithms. One of our main aims in this thesis is to develop exact algorithms for both problems and to implement them and test their performance experimentally. With this objective in mind, we take inspiration from the leading algorithm for the minimum Steiner tree problem, called the GeoSteiner algorithm.

The GeoSteiner Algorithm [22] is an exact algorithm that can rapidly calculate an optimal Euclidean Steiner tree, given a randomly generated set of terminals in the plane. It can solve instances with $10,000$ terminals in less than 500 seconds [13]. Recent improvements have allowed the GeoSteiner Algorithm to solve instances with up to $1,000,000$ terminals [23]. The algorithm consists of two main phases: the *generation* phase and the *concatenation* phase. Every Steiner tree can be decomposed into individual Steiner tree components, known as *full components*, in which terminals only appear as leaves of the component. In the generation phase, the aim is to construct a set of candidate full components guaranteed to include the full components of an optimal tree. The main difficulty in this phase is the exponential number of candidate full components that are potentially optimal. To address this issue, the GeoSteiner Algorithm implements several *pruning tests* that help control the number of full components from proliferating during this phase. Then, in the concatenation phase, a selection of these components (that spans the given set of terminals and is connected) of minimum total length is found by solving an integer program.

In this thesis, our first main contribution is to present an exact algorithm motivated by the GeoSteiner Algorithm that solves the minimum k-Steiner tree problem (i.e., when cost is the "min-sum" objective). Unlike minimum Steiner trees, minimum k-Steiner trees may contain degree-4 Steiner points. Degree-4 Steiner points lead to various significant geometric challenges that must be overcome when developing a GeoSteiner-like algorithm for the k-Steiner tree problem. They also lead to a marked increase in the number of potentially optimal topologies that need to be considered. To overcome these challenges, we introduce a novel way of constructing degree-4 Steiner points during the generation phase and introduce a number of novel pruning tests that vastly reduce the number of full components containing a degree-4 Steiner point. Then, we present the first integer programming formulation and corresponding implementation for the concatenation phase of the k-Steiner tree problem. Finally, we present computational

results to demonstrate the general efficiency of the algorithm as well as the impact of the pruning tests on its performance.

Our second main contribution is to present a new exact algorithm that solves the minimum bottleneck k-Steiner network problem. This algorithm builds on some of the key results of Bae et al. [18], but is based on a framework consisting of generating potentially optimal subtrees (analogous to full components) and an ILP phase which solves a combinatorial problem. The "min-max" objective gives rise to solutions for which the geometry is more challenging than for the "min-sum" problem. For instance, there are no simple conditions for the angles between incident edges to a Steiner point (which is a crucial property for much of the GeoSteiner algorithm). We therefore developed a new approach for generation using Minkowski sums. The output of the generation algorithm is a set of potentially optimal Steiner points, which then serve as input to an ILP. Our ILP then employs an edge-cut formulation for connectivity in order to select an optimal set of Steiner points. As before, we run some experiments to test the efficiency of the algorithm.

1.1 Contribution and referenced papers

Paper 1

Marcus Brazil, Michael Hendriksen, Jae Lee, Michael S Payne, Charl Ras, and Doreen Anne Thomas. An exact algorithm for the Euclidean k-Steiner tree problem. *Computational Geometry*, 121:102099, 2024. [1]

I was intimately involved in all aspects of the research for this paper, including proving theorems, algorithm design, and coding/experimentation. As an estimate, I contributed around 30% of this paper.

Paper 2

Jae Lee, Marcus Brazil, Charl Ras and Doreen Anne Thomas, An Improved Exact Algorithm for the Euclidean k-Steiner Tree Problem (paper has been submitted for journal publication).

I was lead author and the main driver of the research of this paper.

The two papers listed above provide the content for Chapters 2 and 3.

Paper 3

The content for Chapter 4 will be submitted in July 2025 as a third paper.

1.2 Thesis Outline

Chapters 2 - 3 cover the minimum k-Steiner tree problem. In Chapter 2, we first introduce fundamental definitions and concepts required for our algorithm. Then, we present various geometric and structural properties of minimum k-Steiner tree. In Chapter 3, we present our novel exact algorithm in detail. This chapter covers both phases of the algorithm, as well as various pruning tests that are implemented and computational results of the algorithm.

Chapter 4 explores the bottleneck k-Steiner network problem. Again, we first introduce pertinent definitions and concepts. Then, we present our exact algorithm in detail. Finally, we present various computational results of our algorithm.

Chapter 5 concludes the thesis.

Chapter 2

Geometric and structural properties of a minimum k-Steiner Tree

As stated in Section 1.1, the content of this chapter is based on the papers "An exact algorithm for the Euclidean k-Steiner tree problem" [1] and "An Improved Exact Algorithm for the Euclidean k-Steiner Tree Problem". The content from these papers has been adapted to fit the context of a thesis.

In this chapter, we present various geometric and structural properties of a minimum k-Steiner tree. These properties will later be developed into pruning tests that eliminate suboptimal topologies to potentially accelerate our algorithm, as described in Chapter 3. First, we revisit basic definitions and concepts related to the problem in Section 2.1. Then, we present the properties in Section 2.2.

2.1 Preliminaries

In this section, we introduce basic definitions, concepts and theorems that are relevant to the minimum k-Steiner tree problem (covered in Chapters 2 and 3), including our minimum k-Steiner tree algorithm which is covered in Chapter 3.

First, we define the minimum k-Steiner tree problem.

Definition 2.1.1. Let $N = \{t_1, t_2, ..., t_n\}$ be a set of distinct points in $\mathbb{R}^2$ and k be a non-negative integer. A *k-Steiner network* on N is defined to be a connected geometric network $T = (V(T), E(T))$ in $\mathbb{R}^2$, where:

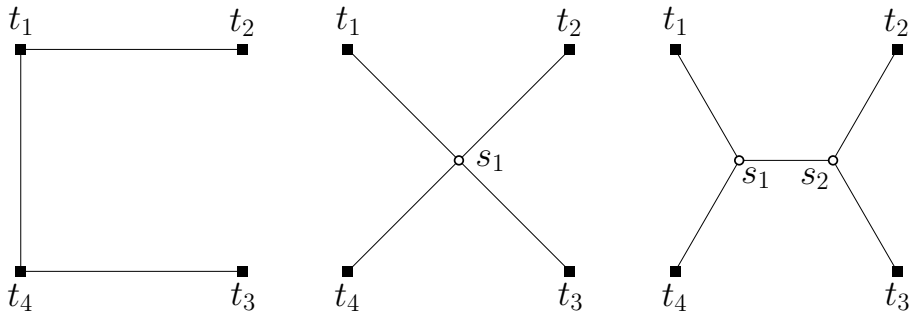
1. $N \subseteq V(T)$, and
2. $|V(T)| \leq n + k$.

The *minimum Euclidean k-Steiner tree problem* may be stated as follows: find a k-Steiner network T on N such that

$$|T| := \sum_{e \in E(T)} |e|$$

is minimum, where $|e|$ denotes the Euclidean length of e. Note, by minimality, that all edges of T are straight line segments and T is necessarily a tree. We refer to $|T|$ as the *length* of T.

The vertices $t_1, t_2, ..., t_n$ are referred to as *terminals*. Any additional vertices are referred to as *Steiner points*. For the classical Steiner tree problem, where the number of Steiner points is not limited to some non-negative integer k, only degree-3 Steiner points may be present in an optimal solution [13]. However, degree-4 Steiner points may occur when the number of Steiner points is limited to k. Steiner points of degree 5 or higher cannot occur, as shown in [24]. Here, we provide a simple example where four terminals are placed at the vertices of a unit square. Minimum k-Steiner trees for $k = 0, 1$ and 2 are shown below in Figures 2.1(a), 2.1(b) and 2.1(c) respectively. Note that a minimum 0-Steiner tree is equivalent to an MST. In this example, a minimum 1-Steiner tree contains a degree-4 Steiner point, while a minimum 2-Steiner tree contains two degree-3 Steiner points.



(a) An MST on the set of terminals $N = \{t_1, t_2, t_3, t_4\}$.

(b) A minimum 1-Steiner tree on N.

(c) A minimum 2-Steiner tree on N.

FIGURE 2.1: Minimum k-Steiner trees for $k = 0, 1, 2$ on the set of terminals N.

For a given geometric network, the underlying graph, which consists of the set of vertices and edges without any associated embedding in the plane, is referred to as its *topology*. Here, the terminals are labelled (and hence distinguishable from one another) but all other vertices are unlabelled.

Definition 2.1.2. A *full Steiner topology* for the minimum k-Steiner tree problem is a tree topology containing at most k Steiner points where every terminal has degree 1 and every Steiner point has degree 3 or 4.

Definition 2.1.3. A *full Steiner tree* (*FST*) is a non-degenerate geometric network (a network with no zero length edges) with a full Steiner topology that is of minimum length with respect to its topology.

Every minimum k-Steiner tree with $n \geq 2$ terminals can be expressed as a union of FSTs that share common terminals. We now state a theorem that characterises degree-3 and degree-4 Steiner points in an FST.

Theorem 2.1.4. Let F be a non-degenerate k-Steiner network with a full Steiner topology. Then F is an FST if and only if each pair of edges incident to a common Steiner point of degree 3 has an interior angle of $2\pi/3$ between them, and every Steiner point of degree 4 is incident to two pairs of collinear edges.

Proof. Clearly F can be viewed as the geometric overlay of a set of FSTs containing either no Steiner points or degree-3 Steiner points only, where each degree-4 Steiner point of F is the intersection of a pair of edges of distinct FSTs from this set.

The problem of finding the optimal Steiner point locations for a given topology is a convex optimisation problem, since it minimises a sum of norms. Therefore, if every perturbation of the Steiner points of F produces an increase in length then F must be an FST.

Any perturbation of the Steiner points will lead to at least one topology in the set of FSTs in the overlay increasing in length, and no other FST in the set will decrease in length (as shown in [25]). Therefore the result follows. □

Proposition 2.1.5. Any angle between two adjacent edges in a minimum k-Steiner tree must be at least $\pi/3$.

The proposition follows from the standard properties of an MST [26], as a minimum k-Steiner tree is an MST on the set of its vertices.

Next, we provide a brief overview of the algorithmic framework for solving the minimum k-Steiner tree problem. A detailed version of the algorithm is covered in Chapter 3. The algorithm consists of two main phases: the *generation phase* and the *concatenation phase*.

In the generation phase, the aim is to construct a set of FSTs that contains the components of a minimum k-Steiner tree. The generation of FSTs begins by merging terminals to form partial FSTs (known as *branches*), which are then iteratively combined (*merged*) to create larger branches. During this process, the branches are tested for optimality and are discarded if they do not satisfy certain geometric properties. These tests are referred to as *pruning tests*. The branches are then *terminated* (by merging directly with a terminal) whenever feasible, to generate FSTs. These concepts will be covered in more detail below, as well as in Chapter 3.

At the end of the generation phase, we have a set of FSTs that contains the full components of at least one minimum k-Steiner tree for N. In the concatenation phase, our aim is to find a subset of these FSTs that constitutes a minimum k-Steiner tree. This can be done by modelling and solving the problem as an ILP. The concatenation phase is explained in detail in Section 3.4.

Next, we present two results which are crucial to the iterative merging process for generating FSTs in the generation phase. Let F be an FST with at least one Steiner

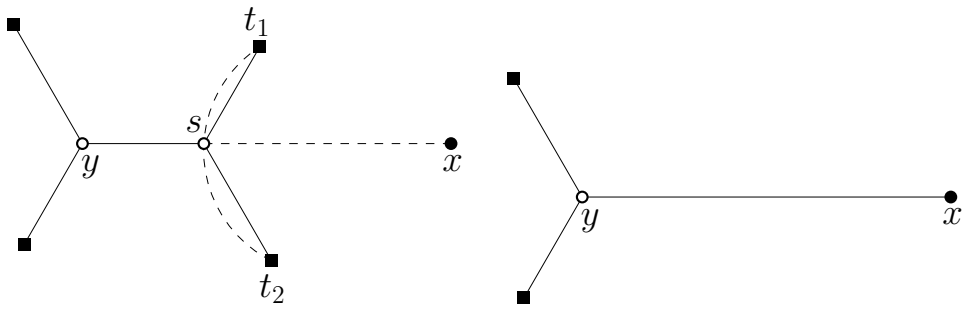
point. Then, there exists either a degree-3 Steiner point that is adjacent to at least two terminals or a degree-4 Steiner point adjacent to at least three terminals. In the case of a degree-3 Steiner point, we can replace the Steiner point and its two terminal neighbours with a *pseudoterminal* and obtain an FST on a smaller set of vertices with the same length. Note that this replacement can be done without knowing the precise location of the Steiner point, if we know the topology of F. Similarly, for a degree-4 Steiner point, we replace the Steiner point and its three terminal neighbours with a pseudoterminal and obtain an FST on a smaller set of vertices with the same length minus a constant. By iteratively doing this replacement, we eventually obtain an FST with no Steiner points, i.e., a single edge, with the same length as the original FST minus a constant. When the original FST only has degree-3 Steiner points then this iterative process is the well-known *Melzak-Hwang algorithm* [13] and the final single-edge FST is called a *Simpson line*.

We now provide more detail on the replacement process, beginning with the case of a degree-3 Steiner point. Given distinct vertices $a, b \in \mathbb{R}^2$, we define the *equilateral point* e_{ab} to be the third vertex of the equilateral triangle whose vertices are a, b and e_{ab}, where the vertices are listed in a counter-clockwise order. Note that e_{ba} is different from e_{ab}.

Let F be a full Steiner tree with terminal-set N_F, where $|N_F| > 2$, and where all Steiner points are of degree 3. Let s be a Steiner point of F adjacent to at least two terminals, say t_1, t_2 (such a Steiner point must exist). Let x be the equilateral point of t_1, t_2 that lies in the half-plane bounded by the line through t_1 and t_2 and that does not contain s (see Figure 2.2(a)). We define the *pseudoterminal* of t_1, t_2 and s as the point x. The pseudoterminal replaces the three aforementioned vertices, and plays the role of a terminal in the reduced network. This will be clarified below. Let y be the vertex of F adjacent to s that is distinct from both t_1 and t_2. Then x, s and y are collinear. Finally, let F^* be the tree that results by removing all edges incident to s, removing nodes t_1, t_2, s, and adding edge xy; see Figure 2.2(b).

The following theorem follows directly from known results (see eg. [25] or [13]), and is illustrated in Figure 2.2.

Theorem 2.1.6. *F^* is a full Steiner tree on $N_F \cup \{x\} \setminus \{t_1, t_2\}$ and $|F^*| = |F|$.*



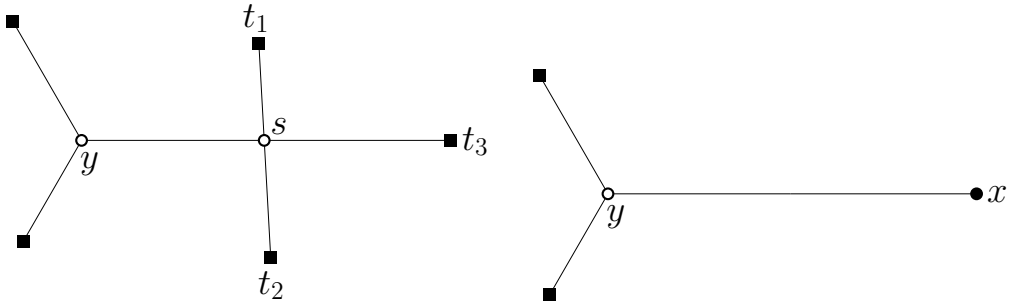
(a) F is a full Steiner tree on N_F.

(b) F^* is a full Steiner tree on $N_F \cup \{x\} \setminus \{t_1, t_2\}$.

FIGURE 2.2: Replacing t_1, t_2 and s by equilateral point x results in an FST of the same length as before replacement. Node x is a pseudoteriminal, and plays the role of a terminal in the reduced network in (b).

Now, we give the corresponding result for a degree-4 Steiner point, illustrated in Figure 2.3.

Corollary 2.1.7 ([1])**.** Let F be an FST, with terminal set N_F, containing a degree-4 Steiner point s adjacent to terminals t_1, t_2, t_3 of N_F, where t_1, t_2 and s are collinear. Let y be the fourth vertex of F adjacent to s. Let F^* be the tree that results by removing edges t_1s, t_2s, t_3s and ys and adding edge yx where $x = t_3$. Then F^* is an FST on terminal set $N_F \cup \{x\} \setminus \{t_1, t_2, t_3\}$ and the length of F^* is $|F| - |t_1t_2|$.



(a) F is a full Steiner tree on N_F.

(b) F^* is a full Steiner tree on $N_F \cup \{x\} \setminus \{t_1, t_2, t_3\}$.

FIGURE 2.3: Replacing t_1, t_2, t_3 and s with a pseudoterminal $x(= t_3)$ results in an FST that has a total length that is $|t_1t_2|$ shorter than before replacement.

The point x in Corollary 2.1.7 is defined to be the pseudoterminal of the points t_1, t_2, t_3 and s.

We now end this section with a known property of a minimum k-Steiner tree which is required for the next section (Section 2.2).

Definition 2.1.8. Given a line segment uv, we define the *lune of* uv, $L(u, v)$, to be the region consisting of all points x such that $|ux| < |uv|$ and $|vx| < |uv|$.

Figure 2.4 illustrates the lune $L(u, v)$ for the line segment uv.

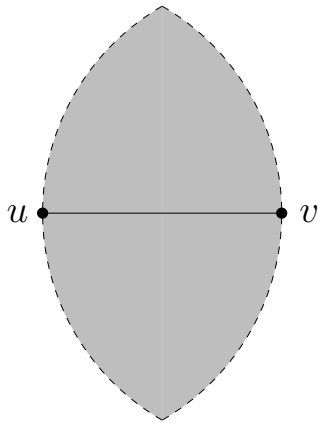


FIGURE 2.4: The lune $L(u, v)$.

Theorem 2.1.9 (The lune theorem, [13], 1.3.2)**.** If uv is an edge of a minimum k-Steiner tree T, then the lune $L(u, v)$ does not contain any Steiner point or terminal of the tree T.

The proof in [13] shows that a Euclidean Steiner tree is not optimal if the lune condition is not satisfied, since there exists a shorter tree. As this shorter tree does not introduce additional Steiner points, the result also applies to k-Steiner trees.

2.2 Geometric and structural properties

In this section, we provide various geometric and structural properties of a minimum k-Steiner tree. These properties can be used create pruning tests, which can then be implemented in the algorithm either during or after the generation phase to reduce the number of FSTs.

2.2.1 The edge-lune theorem

In the context of the minimum Steiner tree problem, the lune theorem states that the lune of an edge in a minimum Steiner tree does not contain any vertices of the tree.

This result can be extended to the interiors of edges: the interior of an edge in a minimum Steiner tree cannot intersect the lune of another edge (referred to as an edge-lune crossing) in the same tree. We now prove this claim.

Let T be a minimum Steiner tree with at least two edges. Let ab be an edge of T. Suppose that there exists another edge xy that intersects the lune $L(a, b)$. Note that these edges may have up to one common end-vertex. Let z be one such point in the intersection, i.e., $z \in xy \cap L(a, b)$. Then, adding the vertex z to T and replacing the edge xy with two edges xz, yz results in a new Steiner tree with one additional Steiner point z and the same length. Since $z \in L(a, b)$, this modified Steiner tree, and consequently T, cannot be optimal. Hence, we have a contradiction.

However, this replacement approach may not be feasible for a minimum k-Steiner tree. Specifically, if a minimum k-Steiner tree contains k Steiner points, an additional Steiner point cannot be introduced. In fact, a simple example shows that an edge-lune crossing can occur in a minimum k-Steiner tree (Figure 2.5). Let $t_1 = (0, 0)$, $t_2 = (4, 0)$ and $t_3 = (1, 3)$ be three terminals given in the problem. We demonstrate that an edge-lune crossing is possible in an MST (which is equivalent to a 0-Steiner tree). Here, $|t_1t_2| = 4$, $|t_1t_3| \approx 3.16$ and $|t_2t_3| \approx 4.24$. Hence, the MST consists of the edges t_1t_2 and t_1t_3. The edge t_1t_3 intersects the lune $L(t_1, t_2)$ as the point $(0.5, 1.5)$ lies on t_1t_3 and is contained in $L(t_1, t_2)$.

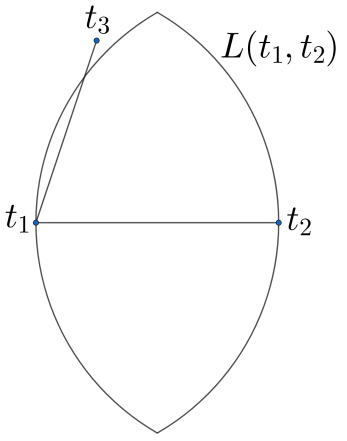


FIGURE 2.5: Edge t_1t_3 of the MST (0-Steiner tree) intersects the lune $L(t_1, t_2)$.

Therefore, we cannot derive an equivalent result for the edges of a minimum k-Steiner tree. However, the following theorem states that an edge-lune crossing cannot occur in

certain cases (specified below). This result may yield additional pruning opportunities beyond those provided by the lune theorem. It can be used either during the generation phase to help avoid the construction of non-minimal FSTs or during the concatenation phase to rule out certain combinations of FSTs.

First, we introduce the following proposition, which will be used in the proof of the edge-lune theorem.

Proposition 2.2.1. Let u, v and v' be three distinct collinear points, where v lies on uv' (as shown in Figure 2.6). Let $C(u) = \{w : |wu| < |uv|\}$, $C'(u) = \{w : |wu| < |uv'|\}$ and similarly, $C(v) = \{w : |wv| < |uv|\}$, $C(v') = \{w : |wv'| < |uv'|\}$. If $p \in C(u)$, then $p \in C'(u)$ and similarly, if $p \in C(v)$, $p \in C(v')$.

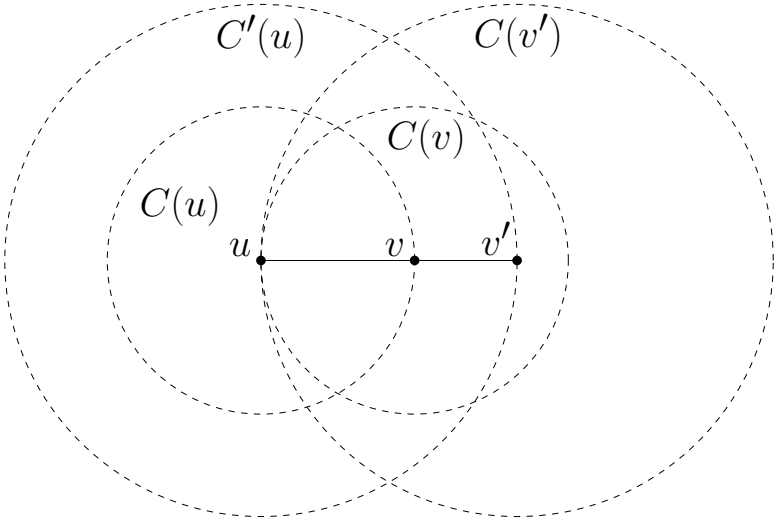


FIGURE 2.6: Proposition 2.2.1 states that $C(u) \subseteq C'(u)$ and $C(v) \subseteq C(v')$.

Proof. Firstly, let p be a vertex such that $p \in C(u)$. Then $|pu| < |uv|$. Hence, $p \in C'(u)$ as $|pu| < |uv| < |uv'|$.

Now, suppose that $p \in C(v)$. Then, $|pv| < |uv|$. Using the triangle inequality, $|pv'| \leq |pv| + |vv'| < |uv| + |vv'| = |uv'|$. Hence, $p \in C(v')$. □

Note that $C(u) \cap C(v) = L(u, v)$. Hence, the proposition implies that if $p \in L(u, v)$, then $p \in L(u, v')$.

We now demonstrate that in the following cases an edge-lune crossing cannot occur in a minimum k-Steiner tree.

Theorem 2.2.2. Let ab and xy be two distinct edges of a k-Steiner network T spanning N such that $xy \cap L(a,b) \neq \emptyset$. Let $C(a) = \{u : |ua| < |ab|\}$ and $C(b) = \{u : |ub| < |ab|\}$. If one end-vertex of xy lies in $C(a) \setminus C(b)$ and the other in $C(b) \setminus C(a)$ or if $x, y \notin C(a) \cup C(b)$, then T is not a minimum k-Steiner tree.

Proof. We will consider the two cases separately. We begin with the following case:

Case 1: One end-vertex of xy lies in $C(a) \setminus C(b)$ and the other in $C(b) \setminus C(a)$

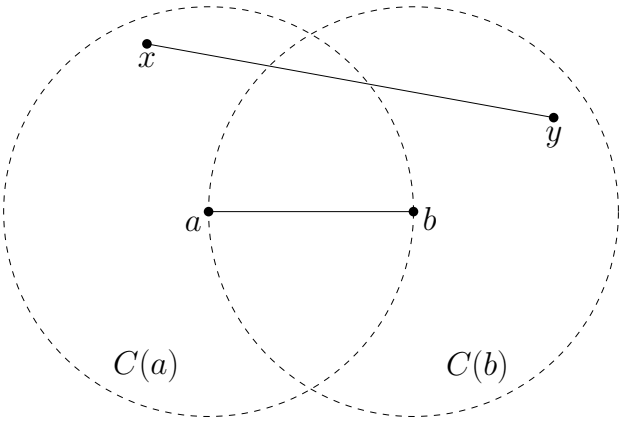


FIGURE 2.7: Case 1 : $x \in C(a) \setminus C(b)$ and $y \in C(b) \setminus C(a)$.

Without loss of generality, we can assume $x \in C(a) \setminus C(b)$ and $y \in C(b) \setminus C(a)$, as shown in Figure 2.7. Note that removing the edge xy from T results in two connected components. Again without loss of generality, we can assume one contains x and the other contains y, a and b.

Case 1a: The unique (y,a)-path in T does not contain the vertex b.

We denote this path by P_{ya}. Adding the edge yb to T creates a unique cycle consisting of P_{ya}, ab and yb. Note that $|yb| < |ab|$ as $y \in C(b) \setminus C(a)$. As $T' = (V(T), E(T) \cup \{yb\} \setminus \{ab\})$ is a k-Steiner tree with $|T'| < |T|$, it follows that T is not a minimum k-Steiner tree.

Case 1b: The unique (y,b)-path in T does not contain the vertex a.

We denote this path by P_{yb}. Adding the edge xa to T creates a unique cycle consisting of xy, P_{yb}, ab and xa. As $x \in C(a) \setminus C(b)$, $|xa| < |ab|$. Since $T' = (V(T), E(T) \cup \{xa\} \setminus \{ab\})$ is a k-Steiner tree with $|T'| < |T|$, T cannot be a minimum k-Steiner tree.

Now, we consider the second case:

Case 2: $x, y \notin C(a) \cup C(b)$.

Let x' be the point on xy lying on the boundary of $C(a) \cup C(b)$ that is closest to x. Similarly, we define y' to be such point that lies closest to y. Note that $x = x'$ and $y = y'$ are possible.

Since $L(a, b) \subset C(a) \cup C(b)$ and xy intersects $L(a, b)$, $x'y'$ must also intersect $L(a, b)$. According to Proposition 2.2.1, if $a \in L(x', y')$, then $a \in L(x, y)$, and similarly, if $b \in L(x', y')$, then $b \in L(x, y)$, which implies that T is not a minimum k-Steiner tree as it does not satisfy the lune theorem for the lune $L(x, y)$.

Hence, we can now assume that x and y lie on the boundary of the region $C(a) \cup C(b)$ and that xy intersects $L(a, b)$ and aim to show that either a or b lies inside $L(x, y)$.

Firstly, we show that if $|xy| > |ab|$, then T cannot be a minimum k-Steiner tree.

Assume $|xy| > |ab|$. Points x and y either lie on the boundary of the same circle (either $C(a)$ or $C(b)$) or on the boundary of different circles. In the first case, without loss of generality, we assume that both x and y lie on the boundary of $C(a)$. Then, $|xa| = |ab| < |xy|$. Similarly, $|ya| = |ab| < |xy|$. Hence, $a \in L(x, y)$ and therefore T cannot be a minimum k-Steiner tree.

In the second case, without loss of generality, let x be the vertex that lies on the boundary of $C(a)$ and y the vertex on the boundary of $C(b)$. Then, $|xa| = |ab| < |xy|$ and $|yb| = |ab| < |xy|$. We now define $C(x) := \{u : |xu| < |xy|\}$ and $C(y) := \{u : |yu| < |xy|\}$. Then, $a \in C(x)$ and $b \in C(y)$. If $a \in C(y)$, this implies that $a \in L(x, y)$ and hence T is not optimal. Hence, the only case remaining to be considered is where $a \in C(x) \setminus C(y)$ and $b \in C(y) \setminus C(x)$. This is equivalent to Case 1, with vertices a, b and x, y switched around, which implies that T is not a minimum k-Steiner tree. Therefore, $|xy| > |ab|$ implies that T is not a minimum k-Steiner tree.

Now, we assume that $|xy| \leq |ab|$ and show that T cannot be optimal.

We first divide the plane into three partitions. Let L_1 be the line orthogonal to ab that passes through a and L_2 the line orthogonal to ab that passes through b. Let P_1 be the open half-plane defined by L_1 that does not include b. Similarly, let P_3 be the open half-plane defined by L_2 that does not include a. Let P_2 be the remaining partition. This is shown in Figure 2.8 below.

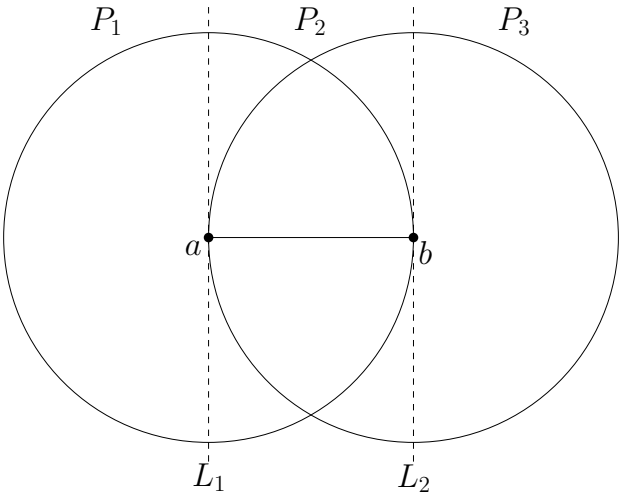


FIGURE 2.8: Lines L_1 and L_2 divide the plane into three partitions P_1, P_2 and P_3.

Without loss of generality, we can assume x lies in P_1 or P_2 and to the left of y. This leads to the following cases based on the locations of x and y.

(a) $x, y \in P_1$

(b) $x \in P_1$, $y \in P_2$

(c) $x \in P_1$, $y \in P_3$

(d) $x, y \in P_2$

The lune $L(a, b)$ exists entirely in P_2. Hence, if $x, y \in P_1$, xy cannot intersect $L(a, b)$ and therefore case (a) is infeasible.

Case (c) cannot occur as it contradicts the assumption that $|xy| \leq |ab|$.

In case (d), xy only intersects $L(a, b)$ if it intersects ab. However, if xy intersects ab, then these edges can be replaced by either the edges ax and by or ay and bx. Both replacements decrease the length of T due to the strict triangle inequality, and at least one of these replacements maintains the connectivity of T. Therefore, T is not a minimum k-Steiner tree.

Hence, we can assume that case (b) holds, that is, $x \in P_1$ and $y \in P_2$. The point y can lie on the boundary of either $C(a)$ or $C(b)$. First, suppose that y lies on the boundary of $C(a)$.

There are two points where the boundaries of $C(a)$ and $C(b)$ intersect. Let the intersection point that lies in the same half-plane defined by $\overline{ab}$ as y be denoted by c. Note that we use $\overline{ab}$ to denote the infinite straight line passing through points a and b.

Let L_3 be the line tangential to the boundary of $C(b)$ at c (as shown in Figure 2.9). Then, L_3 divides the plane into two open half-planes, one of which contains $L(a, b)$. The point y either lies in the other open half-plane or on L_3 (when $y = c$). Hence, x must lie in the same open half-plane as $L(a, b)$ for xy to intersect $L(a, b)$.

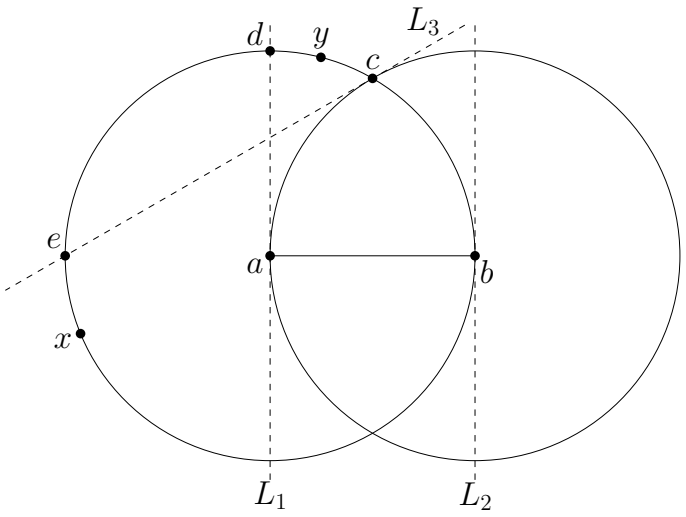


FIGURE 2.9: The point x must lie in the same open half-plane defined by L_3 as $L(a, b)$.

Let d denote the intersection point between L_1 and the boundary of $C(a)$ which lies in the same half-plane defined by $\overline{ab}$ as c. Finally, let e be the intersection point between the boundary of $C(a)$ and L_3 other than c. Then, as x and $L(a, b)$ must lie in the same open half-plane defined by L_3, the chord xy must be subtended by an angle of at least $\angle ead$. Since $\angle ecb = \pi/2$, it follows that be is the diameter of $C(a)$. Therefore, the points e, a and b are collinear. Hence, $\angle ead = \pi/2$. As the chord xy is subtended by an angle of at least $\pi/2$ (as illustrated in Figure 2.9), $|xy| > |ab|$, contradicting our initial assumption.

Now, suppose that y lies on the boundary of $C(b)$. Let c, d, e and L_3 be defined as before. Let L_4 be the line tangent to the boundary of $C(a)$ at c and L_5 the line parallel to L_4 that passes through a. Again, the lines L_4 and L_5 divide the plane into three partitions. Let the open half-plane defined by L_5 that does not contain c be P_a and P_c the open half-plane defined by L_4 that does not contain vertex a. Let P_b be the remaining partition (as shown in Figure 2.10). Since L_4 and L_5 are $|ac| = |ab|$ distance

apart, if $x \in P_a$, then $|xy| > |ab|$ as y either lies in P_c or at c by our assumption. This gives a contradiction, therefore, $x \in P_b$.

Let f denote the point in P_c where L_2 intersects the boundary of $C(b)$. Furthermore, let g be the point in P_1 where L_5 intersects the boundary of $C(a)$.

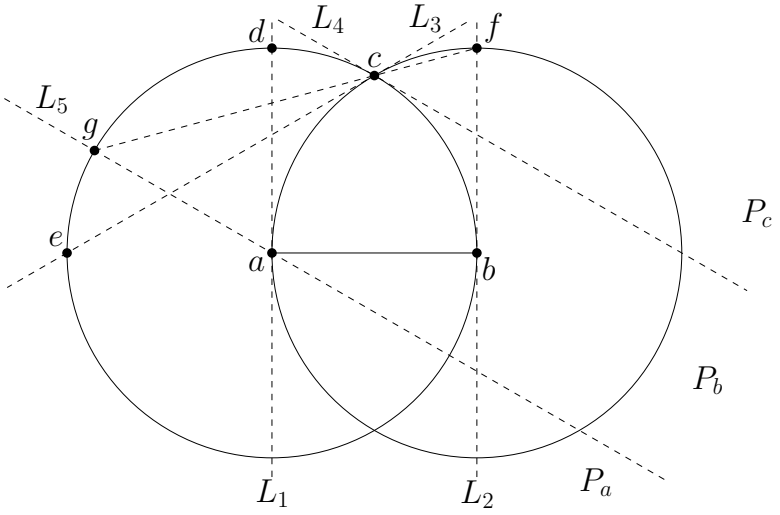


FIGURE 2.10: Lines L_4 and L_5 divide the plane into three partitions P_a, P_b and P_c.

Then, $\triangle gac$ is an isosceles triangle with $\angle gac = \pi/2$ since $|ga| = |ca|$. Similarly, $\triangle bcf$ is an isosceles triangle with $\angle cbf = \pi/2 - \pi/3 = \pi/6$. Hence $\angle gcf = \angle gca + \angle acb + \angle bcf = \pi/4 + \pi/3 + 5\pi/12 = \pi$. That is, g, c and f are collinear. Since $x \in P_b \cap P_1$ and $y \in (P_c \cap P_2) \cup \{c\}$, both x and y lie in the same closed half-plane defined by $\overline{gf}$ as d. The lune $L(a, b)$ lies in the other open half-plane. Hence, xy cannot intersect the lune $L(a, b)$.

This is a contradiction and hence, T is not a minimum k-Steiner tree. □

2.2.2 4-lune theorem

The lune test is an example of an empty region test; it determines the optimality of branches by verifying whether the regions known as lunes are devoid of terminals. The concept of a lune can be generalised to a k-lune (where a lune corresponds to a 2-lune), as defined in [27]. Each value of k defines a distinct region; by varying k, we can potentially obtain regions that are not entirely eclipsed by the lune.

In the minimum Steiner tree problem, the 4-lune regions are defined for every pair of adjacent degree-3 Steiner points (the number 4 is derived from the four inequalities, one at each of the neighbouring vertices of these Steiner points, excluding the two Steiner points themselves). These regions can be generalised to the minimum k-Steiner tree problem as the proof given in [27] does not require any additional Steiner points, ensuring that the number of Steiner points does not exceed k. Here we extend the notion of 4-lunes to a degree-4 Steiner point and its four neighbours that may appear in a minimum k-Steiner tree. We then develop a novel pruning test based on this new 4-lune property in Subsection 3.3.4.

Firstly, we establish the four inequalities associated with the four neighbours of a degree-4 Steiner point. Then, we prove that for three of these inequalities, that if the inequality holds at two distinct points, then it must hold at any convex combination of the two points. This property allows them to be implemented as a dynamic pruning test even when certain vertices do not have fixed locations.

Let T' be a k-Steiner tree that contains a degree-4 Steiner point, s, and T its subtree induced by s and its four neighbouring vertices a, b, c and d (ordered in an anticlockwise manner, as shown in 2.11).

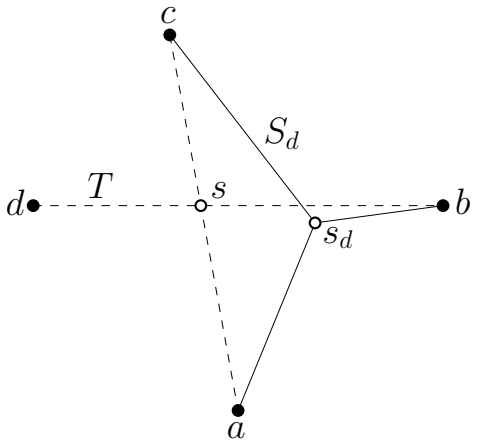


FIGURE 2.11: A minimum Steiner tree S_d that interconnects the vertices a, b and c.

Let S_d be a minimum Steiner tree that connects the set of vertices $\{a, b, c\}$. If this tree contains a degree-3 Steiner point, then the Steiner point is denoted by s_d. Assume that there exists a vertex t in T' that is not in T. Now, suppose that we remove the edges in T, resulting in four components. If t lies in the same component as either a, b or c, then adding the edges in S_d and the edge dt leads to a connected tree. An example where t

lies in the same component as c is shown in Figure 2.12. However, if t lies in the same component as d, the same procedure generates a disconnected network. Replacing the edges in T with those in S_d and the edge dt results in a shorter network if $|S_d|+|dt| < |T|$ (as shown in the figure below). If the connectivity holds under this replacement, then the inequality shows that T' is not minimum. Finally, this replacement procedure does not increase the number of Steiner points and is therefore valid regardless of the number of Steiner points in T'.

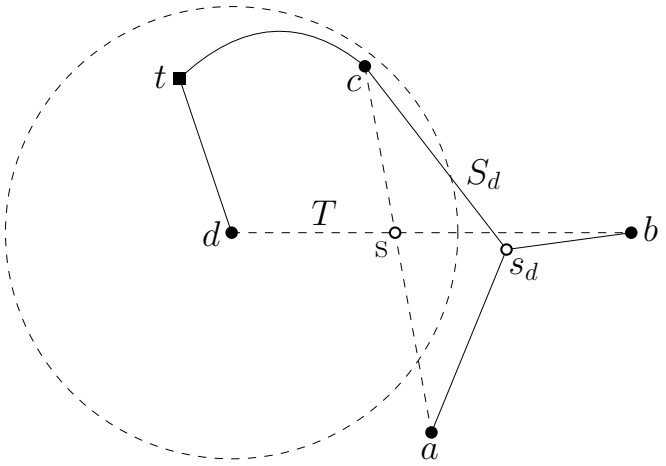


FIGURE 2.12: An example where t lies in the same component as c after the removal of the edges in T. Replacing the edges in T with the edges in S_d and the edge td results in a shorter k-Steiner network if $|S_d| + |dt| < |T|$, which is denoted by an open disc with centre d.

We now define the 4-lune inequalities for the neighbours of a degree-4 Steiner point. These inequalities are derived from comparing the lengths of different networks as shown in the example above.

Definition 2.2.3. Let T' be a k-Steiner network containing a degree-4 Steiner point s and T its subtree induced by s and its four neighbouring vertices a, b, c and d (labelled in an anticlockwise manner). Then, we define the four *4-lune inequalities* for a vertex p as follows:

1. $|ap| < |T| - |S_a|$
2. $|bp| < |T| - |S_b|$
3. $|cp| < |T| - |S_c|$
4. $|dp| < |T| - |S_d|$

where S_i is a minimum Steiner tree that connects the set of vertices $\{a, b, c, d\} \setminus \{i\}$ for $i \in \{a, b, c, d\}$.

Suppose for example that there exists a vertex $p^* \in V(T') \setminus V(T)$ that satisfies the fourth 4-lune inequality in 2.2.3. As previously shown, we may replace the edges in T to generate a shorter k-Steiner tree if either the unique (p^*, a)-path, the (p^*, b)-path or the (p^*, c)- path in T' does not contain the Steiner point s. However, if the unique (p^*, d)-path in T' does not contain s, the same replacement procedure results in a disconnected network.

We will show that if any two 4-lune inequalities hold, then at least one of the corresponding replacements that generate a shorter k-Steiner tree maintain connectivity, leading to the following theorem.

Theorem 2.2.4. Let T' be a k-Steiner tree that contains a subtree T induced by a degree-4 Steiner point s and its four neighbours. Suppose that there exists some vertex p that satisfies at least two 4-lune inequalities (2.2.3) in T. Then, T' cannot be a minimum k-Steiner tree.

Proof. Assume that the two 4-lune inequalities satisfied by p are $|ap| < |T| - |S_a|$ and $|bp| < |T| - |S_b|$. A similar argument holds regardless of which pair of inequalities are satisfied. Let P be the unique path in T' between p and a vertex in T that does not contain any edges of T.

If path P is between p and either vertices b, c or d, we can replace the edges in T with those in S_a and ap. As $|S_a| + |ap| < |T|$, this results in a shorter k-Steiner tree. Similarly, if path P is between p and a, we can replace the edges in T with those in S_b and bp, resulting in a shorter k-Steiner tree. Hence, T' cannot be a minimum k-Steiner tree. □

In the generation phase of GeoSteiner, branches (partial FSTs) are not fully embedded in the plane until they are completed as FSTs. Therefore, some Steiner points in a branch do not have fixed locations, and instead either lie on an arc or a segment (this is explained in Section 3.1). This makes it difficult to calculate the 4-lune inequalities in a branch during the generation phase. However, the following theorem shows that for three out of the four 4-lune inequalities (shown in 2.2.3), that if these inequalities are satisfied at two distinct points, then they must also be satisfied at every convex

combination of the two points. This implies that if the inequalities are satisfied at the extreme points of a polygonal region that contains the arc or segment that the Steiner point lies on, then the inequalities are satisfied regardless of where on the arc or the segment the Steiner point is located. The proof follows a similar line of reasoning to the one shown in [27] which proves the same property for all three 3-lune inequalities.

Theorem 2.2.5. Let T' be a minimum k-Steiner tree with a degree-4 Steiner point where the 4-lune inequalities are defined as in 2.2.3. Now, suppose that c is a variable point. If there exists some vertex p that satisfies Inequality 2, 3 or 4 for some $c = c_1$ and $c = c_2$, then the inequality holds for all $c \in c_1c_2$.

Proof. Without loss of generality, we assume that $c = (x, 0)$ lies on the line segment c_1c_2 where $c_1 = (0, 0)$ and $c_2 = (1, 0)$ (as shown in Figure 2.13).

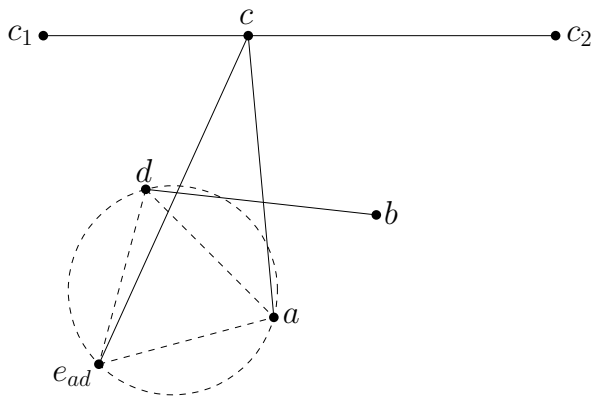


FIGURE 2.13: The vertex c lies on the line segment c_1c_2.

Inequality 2

Assume that $|bp| < |T| - |S_b|$. Then, $|bp| - |bd| < |ca| - |e_{ad}c|$ where e_{ad} is the equilateral point of a and d. Let $a = (a_1, a_2)$ and $e_{ad} = (e_1, e_2)$. Let h be a function of the RHS of the inequality in terms of x. Then,

$$h(x) := \sqrt{(x - a_1)^2 + a_2^2} - \sqrt{(x - e_1)^2 + e_2^2}$$

$$h'(x) = \frac{x - a_1}{\sqrt{(x - a_1)^2 + a_2^2}} - \frac{x - e_1}{\sqrt{(x - e_1)^2 + e_2^2}}$$

Let $\alpha := \angle c_1ce_{ad}$ and $\delta := \angle c_1ca$. Note that $0 \leq \alpha, \delta \leq \pi$.

Then,

$$h'(x) = \cos\delta - \cos\alpha$$

Therefore, $h'(x) = 0$ if and only if $\alpha = \delta$, which implies that c, a and e_{ad} are collinear. Since $\angle dae_{ad} = \pi/3$, $\angle dac = 2\pi/3$. Then, $\angle dsa < \pi/3$ which contradicts Proposition 2.1.5, as T' is a minimum k-Steiner tree. Hence, $h'(x)$ is never 0 for $c \in c_1c_2$ and therefore, if Inequality 2 in 2.2.3 is satisfied at $c = c_1$ and $c = c_2$, then it is also satisfied for all $c \in c_1c_2$.

Inequality 4

By symmetry, this is also true for Inequality 4.

Inequality 3

Assume that $|cp| < |T| - |S_c|$. Then, $|ca| - |cp| > |e_{ad}b| - |bd|$. Let the s_c be the Fermat point of vertices a, b and d. Then $|e_{ad}b| = |s_cb| + |s_cd| + |s_ca| > |bd| + |s_ca|$ by the triangle inequality. The inequality must be strict because $\angle bs_cd = 2\pi/3$. Hence,

$$|ca| - |cp| > |e_{ad}b| - |bd| > 0$$

Note that moving c does not affect $|e_{ad}b| - |bd|$. So, this inequality defines the set of points that are at least $|e_{ad}b| - |bd|$ units closer to p than to a, which is a convex region (shown below in Figure 2.14). Hence, if c_1 and c_2 lie inside this region, then any point on the line segment c_1c_2 must also lie within this region, which implies that the inequality is satisfied for any $c \in c_1c_2$.

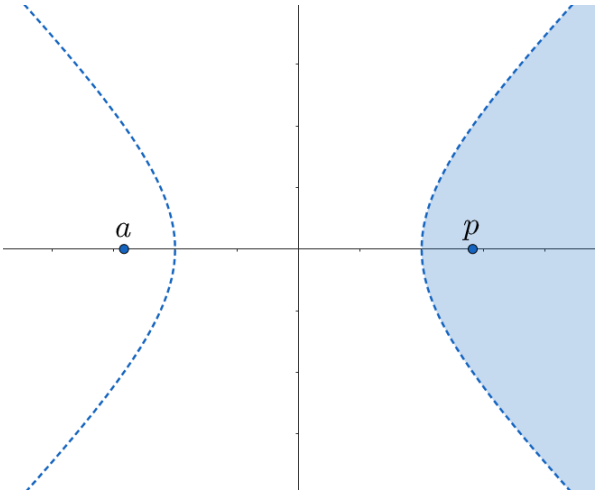


FIGURE 2.14: The shaded region depicts the set of points that are at least $|e_{ad}b| - |bd|$ units closer to p than to a, for some positive value of $|e_{ad}b| - |bd|$.

□

However, this property does not hold for Inequality 1 in Definition 2.2.3, as shown by a counter-example below.

Inequality 1

Let $a = (0.5, -0.5)$, $b = (0.7, -0.3)$ and $d = (0.3, -0.3)$. Vertex c lies on the line segment c_1c_2 where $c_1 = (0.2, 0)$ and $c_2 = (0.8, 0)$ (shown below in Figure 2.15).

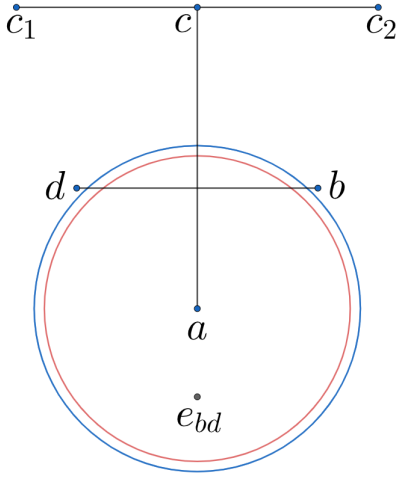


FIGURE 2.15: The blue circle shows the boundary of the points satisfying Inequality 1 when $c = c_1$ or $c = c_2$. The red circle shows the boundary of the points satisfying the inequality when $c = (0.5, 0)$.

A minimum Steiner tree spanning vertices b, d and c is denoted by S_a.

Due to symmetry, $|T| - |S_a| = \frac{\sqrt{34}}{10} + \frac{2}{5} - \frac{\sqrt{30+12\sqrt{3}}}{10} \approx 0.2705$ when $c = c_1$ and c_2. If $c = (0.5, 0)$, $|T| - |S_a| = 0.9 - (\frac{3}{10} + \frac{\sqrt{3}}{5}) \approx 0.2536$.

In Figure 2.15, the boundary of the region satisfying Inequality 1 is shown in blue for $x = 0.2$ and $x = 0.8$, and in red for $x = 0.5$. There are points within the blue boundary that lie outside the red boundary. Hence, if a vertex p satisfies $|ap| < |T| - |S_a|$ for some $c = c_1$ and $c = c_2$, it does not necessarily satisfy the inequality for all $c \in c_1c_2$.

2.2.3 Rhombus Theorem

In this subsection, we present the Rhombus Theorem, which states that a neighbour of a degree-4 Steiner point must lie within a region specified by the other neighbours of the Steiner point. This theorem, alongside the Trapezium Theorem (Theorem 2.2.8, covered in the following subsection), will be developed into a new pruning test in Subsection 3.3.5.

Firstly, we establish a lemma that is required for the main theorem.

Lemma 2.2.6. Let $P = \{a, b, c, d\}$ (arranged clockwise) be the vertices of a convex polygon with two consecutive internal angles (say $\angle abc$ and $\angle bcd$) of at least $2\pi/3$. Let N_0 denote the geometric network spanning P consisting of a degree 4 Steiner point s, located at the intersection of ac and bd, together with the edges as, bs, cs and ds. Let N_1 denote the geometric network on P with edges ab, bc, cd. Then $|N_1| < |N_0|$.

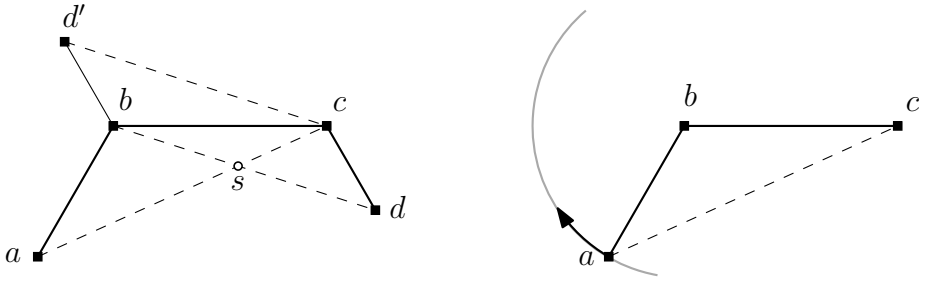


FIGURE 2.16: Constructions for the proof of Lemma 2.2.6.

Proof. Assume, in the first instance, that $\angle abc = \angle bcd = 2\pi/3$. Let d' be the point such that the vectors $\overrightarrow{d'b}$ and $\overrightarrow{cd}$ are equal, as in Figure 2.16 (left). By similar triangles it follows that $|d'c| = |bd|$ and $\angle d'bc = 2\pi/3$. This angle property implies that the tree with edges $d'b \cup bc \cup ab$ is the unique minimum Steiner tree for $\{a, c, d'\}$ (with Steiner

point b). Hence:

$$|N_1| = |d'b| + |bc| + |ab| < |d'c| + |ca| = |N_0|.$$

To see that the inequality still holds in the case where $\angle abc > 2\pi/3$, consider the effect on N_0 and N_1 of moving a around the circle centre b and radius $|ab|$, so that $\angle abc$ increases (as illustrated in Figure 2.16 (right)). This movement does not change the length of N_1 but continuously increases the length of ac (until $\angle abc = \pi$). A similar argument applies for angle $\angle bcd$. □

Now, we present the Rhombus Theorem.

Theorem 2.2.7 (The Rhombus Theorem)**.** Let $P = \{u, p, v, q\}$ be the vertices of a convex quadrilateral (arranged clockwise) such that a minimum 1-Steiner tree on P contains a degree-4 Steiner point s. Then p and q lie in the rhombus with vertices u, e_{uv}, v and e_{vu}.

Proof. By Theorem 2.1.4, the points u, s, v are collinear and the points p, s, q are collinear (that is, s is the intersection point of the diagonals of the quadrilateral with vertices u, p, v, q). Let x be any point for which there exists a 1-Steiner tree on u, p, v, x with degree-4 Steiner point s, and edges us, ps, vs, xs. We denote such a Steiner tree $\mathcal{C}_x$; see Figure 2.17.

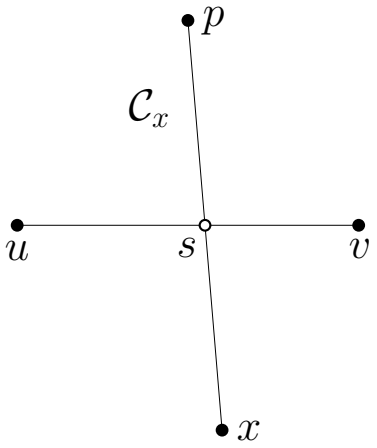


FIGURE 2.17: The Steiner tree $\mathcal{C}_x$.

It suffices to show that q lies in the triangle $\triangle uve_{vu}$, which implies the statement of the theorem is true by symmetry. We proceed by considering three cases.

Case 1: $\angle upv \leq 2\pi/3$

We consider the curves that form the locus of points x (representing possible locations of q) for which $\mathcal{C}_x$ and a certain 1-Steiner tree on u, p, v, x have the same total length. In particular, we will consider the 1-Steiner tree formed by union of the Steiner tree on u, p, v, together with either edge ux or vx. Call the former tree T_u and the latter T_v. We note that as a consequence of Theorems 2.1.4 and 2.1.6, the total lengths of T_u and T_v are simply $|pe_{vu}| + |ux|$ and $|pe_{vu}| + |vx|$ respectively, as $\angle upv \leq 2\pi/3$. Indeed, the locus of points x for which the lengths of $\mathcal{C}_x$ and T_u are equal, must obey the relation

$$|px| + |uv| = |pe_{vu}| + |ux|$$

so

$$|px| - |ux| = |pe_{vu}| - |uv|.$$

This is one arc of a hyperbola H_1, with foci u and p. Similarly, for T_v,

$$|px| - |vx| = |pe_{vu}| - |uv|,$$

which is one arc of a hyperbola H_2, with foci p and v. See Figure 2.18 for an example of this construction.

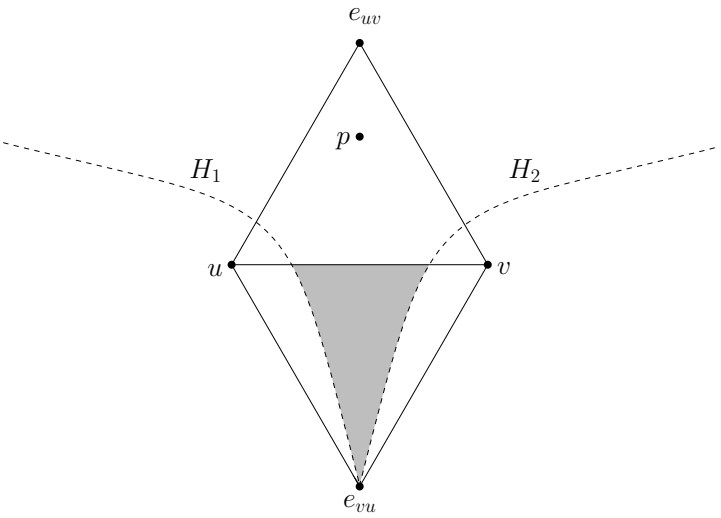


FIGURE 2.18: An example of the hyperbolae H_1 and H_2 constructed during the proof of Theorem 2.2.7.

We note that in order for $\mathcal{C}_x$ to have a shorter total length than T_u, it must be that H_1 separates x from u. Similarly, for T_v, it must be that H_2 separates x from v.

We now consider the equilateral point of u and v that lies on the opposite side of the interval uv to p, that is, e_{vu}. This point is located a distance of $|uv|$ from u, and a distance of $|pe_{vu}|$ from p - i.e. the difference of the distances $|ue_{vu}|$ and $|pe_{vu}|$ is $|pe_{vu}| - |uv|$, and so e_{vu} must lie on the hyperbola H_1. By considering the distance to v, we see that e_{vu} must also lie on the hyperbola H_2, and so e_{vu} is an intersection point of H_1 and H_2.

Note that q must lie on the opposite side of the interval uv to p, that is, q must lie in the half-plane $\mathcal{H}$ supported by $\overline{uv}$ that does not contain p. As H_1 has focus u and passes through the equilateral point e_{vu}, and H_2 has focus v and passes through e_{vu}, the intersection of $\mathcal{H}$ and the exteriors of H_1 and H_2 (shown in grey in Figure 2.18) lies entirely in the triangle $\triangle uve_{vu}$, and so q must lie in $\triangle uve_{vu}$. By symmetry, p must also lie in the triangle $\triangle uve_{uv}$.

Case 2: $\angle upv > 2\pi/3$, and both $\angle qup$ and $\angle qvp \leq 2\pi/3$

Case 1 showed that if $\angle upv \leq 2\pi/3$, then q must lie in the triangle $\triangle uve_{vu}$. Similarly, as $\angle qup \leq 2\pi/3$, v must lie in the triangle $\triangle qpe_{pq}$, and as $\angle qvp \leq 2\pi/3$, u must lie in the triangle $\triangle qpe_{qp}$ - that is, u and v must lie inside the rhombus with vertices q, p, e_{pq} and e_{qp}. However, this is impossible as $\angle upv > 2\pi/3$. Therefore in this case a 1-Steiner tree on P containing a degree-4 Steiner point cannot be minimum.

Case 3: $\angle upv$ and either $\angle qup$ or $\angle qvp$ are greater than $2\pi/3$

By Lemma 2.2.6, there exists a spanning tree on P with length less than that of $\mathcal{C}_x$, so again $\mathcal{C}_x$ cannot be minimum.

As all cases have been considered the theorem is proven. □

2.2.4 Trapezium Theorem

In this subsection, we present and prove the Trapezium Theorem. This theorem relies on the Rhombus Theorem (Theorem 2.2.7), which states that a neighbour of a degree-4 Steiner point cannot lie too far from the Steiner point in an optimal k-Steiner tree.

Conversely, the Trapezium Theorem states that a neighbour of a degree-4 Steiner point cannot lie too close to the Steiner point. The Rhombus Theorem and the Trapezium Theorem will be developed into a new pruning test in Subsection 3.3.5.

Theorem 2.2.8 (The Trapezium Theorem)**.** Let s be a degree-4 Steiner point in a minimum k-Steiner tree and let u, p, v, q (labelled clockwise) be the four vertices adjacent to s. Denote the perpendicular distance between p and $\overline{uv}$ by d_p. Then, $d_p \geq \frac{1}{2\sqrt{3}}|uv|$.

Proof. By the Rhombus Theorem, p lies inside the triangle $\triangle uve_{uv}$. Arguing by contradiction, assume the perpendicular distance between p and uv is strictly less than $\frac{1}{2\sqrt{3}}|uv|$. Furthermore, without loss of generality, assume that p lies closer to u than to v, i.e., $|pu| \leq |pv|$. If p lies strictly inside the circumcircle of triangle $\triangle uve_{vu}$, then $\angle upv > 2\pi/3$ and thus u and v cannot both lie inside the rhombus whose vertices are p, q and their two equilateral points e_{pq} and e_{qp}. Hence, p must lie in the region A as shown in Figure 2.19.

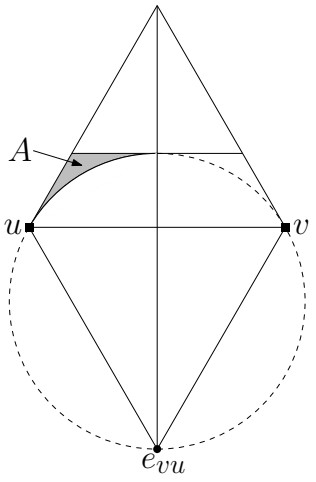


FIGURE 2.19: Vertex p is confined to the region A.

The location of p has a number of consequences that are illustrated in Figure 2.20. Firstly, q does not lie strictly inside the circle $C(p, v)$ centred at p with radius pv. Otherwise, $|pq| < |pv|$ which implies that v cannot lie inside the triangle $\triangle pqe_{pq}$. Next, angle $\angle qpv \leq \pi/3$ since otherwise v does not lie inside the triangle $\triangle pqe_{pq}$. Similarly, $\angle upq \leq \pi/3$ otherwise u does not lie inside the triangle $\triangle qpe_{qp}$. Let u_l be the point on $C(p, v)$ in $\triangle uve_{vu}$ such that $\angle u_l pv = \pi/3$, and let u_r be the point on $C(p, v)$ in $\triangle uve_{vu}$ such that $\angle upu_r = \pi/3$. Then q lies in the part of the convex cone between the two rays

$\overrightarrow{pu_l}$ and $\overrightarrow{pu_r}$ that lies on or outside $C(p, v)$. We denote this feasible region as $B(u)$, as illustrated in Figure 2.20.

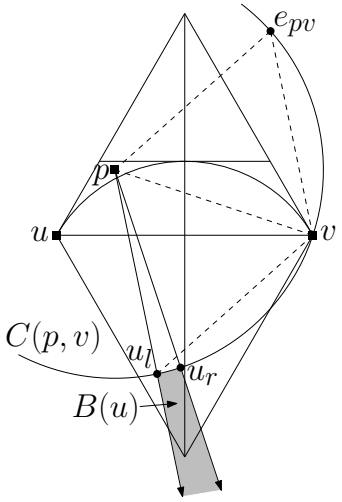


FIGURE 2.20: Vertex q is confined to the region $B(u)$.

Let T_3 be the tree with a degree-3 Steiner point adjacent to vertices p, v, q and the edge up and let T_4 be the tree interconnecting u, p, v and q with a degree-4 Steiner point. Then $|T_4| = |uv| + |pq|$ and $|T_3| = |up| + |e_{pv}q|$. Now, we aim to find the location of q that minimises $|T_4| - |T_3|$ for all points $q \in B(u)$.

For a given $p \in A$, we initially let q coincide with u_l. For any point $q' \in B(u)$, we can always move q further along the ray $\overrightarrow{pu_l}$ then rotate it anticlockwise about p to get $q = q'$. Note that moving q in this manner only affects $|pq|$ and $|e_{pv}q|$ in $|T_4|$ and $|T_3|$ respectively.

Moving q from its initial position at u_l along the ray $\overrightarrow{pu_l}$ (away from p) increases $|pq|$ at a strictly faster rate than $|e_{pv}q|$, since e_{pv}, p and q are not collinear. Hence, $|T_4| - |T_3|$ increases, as q moves along $\overrightarrow{pu_l}$.

Next, for any location of q on the ray $\overrightarrow{pu_l}$, if we move q anticlockwise about the circle with centre p and radius pq, then $|pq|$ remains fixed, whereas $|e_{pv}q|$ decreases until q, e_{pv} and p are collinear. Hence, $|pq| - |e_{pv}q|$ increases, resulting in $|T_4| - |T_3|$ also increasing.

Thus, moving q from u_l to any other point in $B(u)$ always increases $|T_4| - |T_3|$. Hence, $|T_4| - |T_3|$ is minimised when q is at u_l. To prove the theorem, it now suffices to show that $|T_3| \leq |T_4|$ for this location of q.

Without loss of generality, let $u = (0, 0)$ and $v = (1, 0)$. Also, for a given $p = (x, y)$, let $q = u_l$. Observe that

$$\begin{aligned} |T_4| &= |uv| + |pq| \\ &= |uv| + |pv| \\ &= 1 + \sqrt{(x-1)^2 + y^2}. \end{aligned}$$

Similarly,

$$\begin{aligned} |T_3| &= |up| + |e_{pv}q| \\ &= |up| + \sqrt{3}|pv| \\ &= \sqrt{x^2 + y^2} + \sqrt{3}\sqrt{(x-1)^2 + y^2}. \end{aligned}$$

Hence,

$$|T_4| - |T_3| = 1 - \sqrt{x^2 + y^2} - (\sqrt{3} - 1)\sqrt{(x-1)^2 + y^2}.$$

Let $f(x, y) := \sqrt{x^2 + y^2} + (\sqrt{3} - 1)\sqrt{(x-1)^2 + y^2}$. If $\max_{(x,y)\in A} f(x, y) \leq 1$ then $|T_4| \geq |T_3|$.

Since $y \geq 0$, the function $f(x, y)$ is increasing with respect to y. Hence, we consider the value of f when $y = \sqrt{3}x$ for $0 \leq x \leq 1/6$ and $y = \sqrt{3}/6$ for $1/6 \leq x \leq 1/2$.

For $0 \leq x \leq 1/6$,

$$\max_{(x,y)\in A, x\in[0,1/6]} f(x, y) = \max_{x\in[0,1/6]} f(x, \sqrt{3}x)$$

where

$$\begin{aligned} f(x, \sqrt{3}x) &= \sqrt{x^2 + 3x^2} + (\sqrt{3} - 1)\sqrt{(x-1)^2 + 3x^2} \\ &= 2x + 2(\sqrt{3} - 1)\sqrt{(x - 1/4)^2 + 3/16}. \end{aligned}$$

However,

$$f'(x, \sqrt{3}x) = 2 + \frac{2(\sqrt{3} - 1)(x - 1/4)}{\sqrt{((x - 1/4)^2 + 3/16)}} > 0$$

since $-(x-1/4) < \sqrt{(x-1/4)^2+3/16}$ and $0 < \sqrt{3}-1 < 1$.

Therefore,

$$\max_{(x,y)\in A, x\in[0,1/6]} f(x,y) = f(1/6, \sqrt{3}/6) = 1/3 + (\sqrt{3}-1)\sqrt{7}/3 \approx 0.97894.$$

Similarly, for $x \in [1/6, 1/2]$

$$\max_{(x,y)\in A, x\in[1/6,1/2]} f(x,y) = \max_{x\in[1/6,1/2]} f(x, \sqrt{3}/6)$$

where

$$\begin{aligned} f(x,\sqrt{3}/6) &= \sqrt{x^2+(\sqrt{3}/6)^2} + (\sqrt{3}-1)\sqrt{(x-1)^2+(\sqrt{3}/6)^2} \\ &= \sqrt{x^2+1/12} + (\sqrt{3}-1)\sqrt{(x-1)^2+1/12}. \end{aligned}$$

Both $g(x) = \sqrt{x^2+1/12}$ and $h(x) = \sqrt{(x-1)^2+1/12}$ are convex functions since they are $||(x, 1/\sqrt{12})||_2$ and $||(x-1, 1/\sqrt{12})||_2$ respectively. Furthermore, $\sqrt{3}-1 > 0$, so $f(x,\sqrt{3}/6) = g(x) + (\sqrt{3}-1)h(x)$ is a convex function. Hence, the maximum value of $f(x,\sqrt{3}/6)$ is reached at either $x = 1/6$ or $1/2$.

Now,

$$\begin{aligned} f(1/6, \sqrt{3}/6) &= 1/3 + (\sqrt{3}-1)\sqrt{7}/3 \approx 0.97894, \\ f(1/2, \sqrt{3}/6) &= 1, \end{aligned}$$

therefore

$$\max_{(x,y)\in A, x\in[0,1/2]} f(x,y) = \max_{(x,y)\in A, x\in[1/6,1/2]} f(x,y) = 1$$

It follows that $\max_{(x,y)\in A} f(x,y) \leq 1$ and hence $|T_4| \geq |T_3|$, as required. □

The Rhombus Theorem (Theorem 2.2.7) and the Trapezium Theorem (Theorem 2.2.8) restrict vertex p to the region T_{uv}, an equilateral triangular region consisting of all points in the region $\triangle ue_{uv}v$ whose distance from uv is at least $\frac{1}{2\sqrt{3}}|uv|$ (as shown in Figure 2.21).

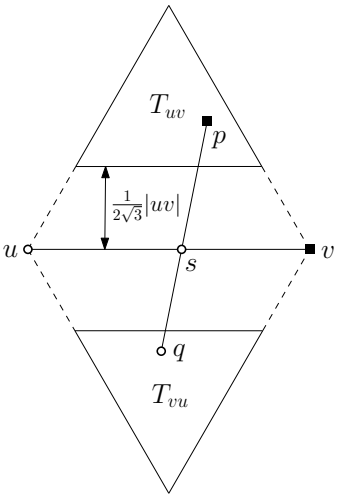


FIGURE 2.21: An illustration of Theorem 2.2.7 and Theorem 2.2.8.

Chapter 3

Implementation of the exact k-Steiner tree algorithm

In this chapter, we present our exact algorithm that solves the minimum k-Steiner tree problem.

First, we define the notion of branch trees and branches in Section 3.1. Then, we present the different types of merges used during the generation phase, including a new type of merge called the *triple-merge* in Section 3.2. We conclude this section with a detailed description of the generation phase. Section 3.3 presents various pruning tests, the majority of which have been incorporated into our algorithm. The following section, Section 3.4, details the concatenation phase of the algorithm. Section 3.5 concludes the chapter with experimental results and discussion of our k-Steiner tree algorithm.

3.1 Branch trees and branches

The pseudoterminal results (Theorem 2.1.6 and Corollary 2.1.7) form the basis of an iterative framework for constructing FSTs. During the generation phase, the FSTs are generated by merging geometric structures known as *branches*. In this section, we describe the concept of a branch in detail.

Definition 3.1.1. A *branch tree* B is a tree spanning a set of terminals N_B, and containing a ray (known as the *stem*) incident to one of the vertices v of the tree (known

as the *root*), such that for any point $a \neq v$ on the stem, the union of av with every other edge of B excluding the stem is an FST for $\{a\} \cup N_B$. A *branch* $\mathcal{B}$ is a (possibly infinite) set of branch trees, each of which spans a common set of terminals $N_{\mathcal{B}}$, such that every branch tree $B \in \mathcal{B}$ has the same topology.

Note that the root of a branch tree B is always a Steiner point unless $|N_B| = 1$. Every branch $\mathcal{B}$ can be represented by a pseudoterminal p and an associated *Steiner curve* A. The Steiner curve of a branch $\mathcal{B}$ is the locus of roots of the branch trees $B \in \mathcal{B}$. The Steiner curve is an *arc* if the root is a degree-3 Steiner point and a *segment* (that is, a straight line segment) if the root is a degree-4 Steiner point. The endpoints of the Steiner curve are given by $l(A)$ and $r(A)$ where $l(A)$ is the left endpoint and $r(A)$ is the right endpoint of the curve with respect to the pseudoterminal. Otherwise, if the root of the branch is its only terminal, then its root is fixed, and we treat the Steiner curve as the complete arc of a circle with zero radius; and we think of the terminal as also being a pseudoterminal.

In Section 3.3, we introduce what are known as *pruning tests* that restict these Steiner curves to help mitigate the number of possible subsequent merges and thereby the number of resulting branches and FSTs. These tests are based on the geometric and structural properties of minimum k-Steiner trees (some of which are covered in Chapter 2).

3.2 The generation algorithm

In this section, we describe the generation phase. Firstly, we introduce the three types of merges that occur during this phase: the double-merge, the termination merge and the triple-merge. The double-merge and the termination merge are standard features of GeoSteiner for the Euclidean Steiner tree problem, and are covered in detail in [13]. Then, we present the pseudocode for the generation phase and describe each of the processes in depth.

Double-merge

Suppose that branches $\mathcal{B}_1$ and $\mathcal{B}_2$ are double-merged to generate a new branch $\mathcal{B}$ with Steiner arc A and pseudoterminal p. Let the pseudoterminal, root and Steiner curve of branch $\mathcal{B}_i$ be p_i, s_i and A_i respectively, for $i = 1, 2$. The new pseudoterminal, p, is an

equilateral point of vertices p_1 and p_2. The new Steiner arc, A, is an arc of the circle containing p_1, p_2 and p. An example where A_1 is a Steiner segment and A_2 is a Steiner arc is shown below in Figure 3.1.

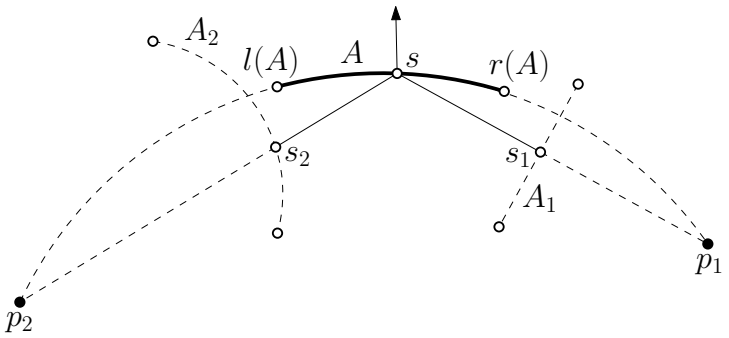
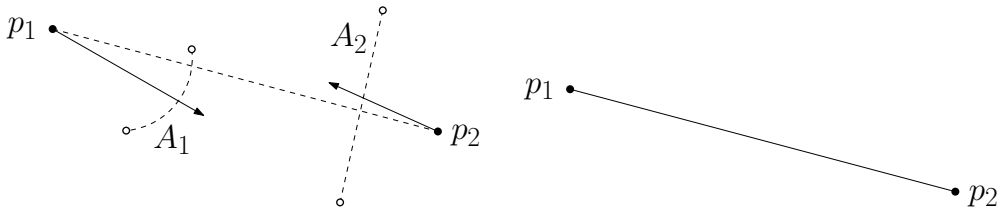


FIGURE 3.1: Two branches $\mathcal{B}_1$ and $\mathcal{B}_2$ are double-merged. The new pseudoterminal p is not shown here, but lies below the figure on the line containing the stem.

Termination merging

Suppose that $\mathcal{B}_i$ is a branch with pseudoterminal p_i and Steiner curve A_i for $i = \{1, 2\}$. Then branch $\mathcal{B}_1$ can be termination merged with branch $\mathcal{B}_2$ only if p_1p_2 intersects A_1 at s_1 strictly between p_1 and s_2 and A_2 at s_2 strictly between s_1 and p_2. In the case where p_i is a terminal (that is, $\mathcal{B}_i$ contains no Steiner points), then we assume $s_i = p_i$. The merge results in the generation of a new FST from the two branches. The length of the new FST is given by calculating the distance between the two pseudoterminals, i.e., $|p_1p_2|$, and adding any lengths that were stored when triple-merges (described below) occurred in the construction of $\mathcal{B}_1$ and $\mathcal{B}_2$. The locations of the Steiner points can be determined by using the reconstruction stage of the Melzak-Hwang algorithm [13]. In practice, termination merging to create an FST can always be done by merging a branch with a single terminal, an approach which is useful for minimising the duplication of FSTs during the generation phase. But on the other hand, it is necessary to implement a more general termination merging process as part of the triple-merge, as we will see shortly.



(a) Branches $\mathcal{B}_1$ and $\mathcal{B}_2$ with pseudoterminals p_1 and p_2 are termination merged.

(b) The resulting FST is obtained by reversing the merges.

FIGURE 3.2: An example of a termination merge.

Triple-merge

Let $\mathcal{B}_i$ be a branch with pseudoterminal p_i, root s_i and Steiner curve A_i for $i = \{1, 2, 3\}$. Suppose that points p_1, p_2 and p_3 form the vertices of a triangle where the vertices are ordered in a counterclockwise direction, as shown in Figure 3.3. The triple-merge is feasible if $\mathcal{B}_1$ can be termination merged with $\mathcal{B}_2$. Briefly, this means that p_1p_2 intersects A_1 and A_2 at s_1 and s_2 respectively, where s_1 lies strictly between p_1 and s_2 and s_2 strictly between s_1 and p_2. The length of the FST generated from termination merging $\mathcal{B}_1$ and $\mathcal{B}_2$ is calculated and stored for later use. The branch that is not involved in the termination merge is referred to as the *source branch* in the triple-merge.

Let the new branch generated from the triple-merge be $\mathcal{B}$ with pseudoterminal p and Steiner segment A. The new branch $\mathcal{B}$ can be viewed as the geometric overlay of the FST resulting from termination merging of $\mathcal{B}_1$ and $\mathcal{B}_2$, and the branch $\mathcal{B}_3$. We define the initial Steiner segment A as the portion of the line segment between s_1 and s_2 that arises from the termination merging of $\mathcal{B}_1$ and $\mathcal{B}_2$. The Steiner segment A then contains all feasible intersections of the branch trees for $\mathcal{B}_3$ with the initial Steiner segment, and is updated during the projection pruning test (covered in Subsection 3.3.1).

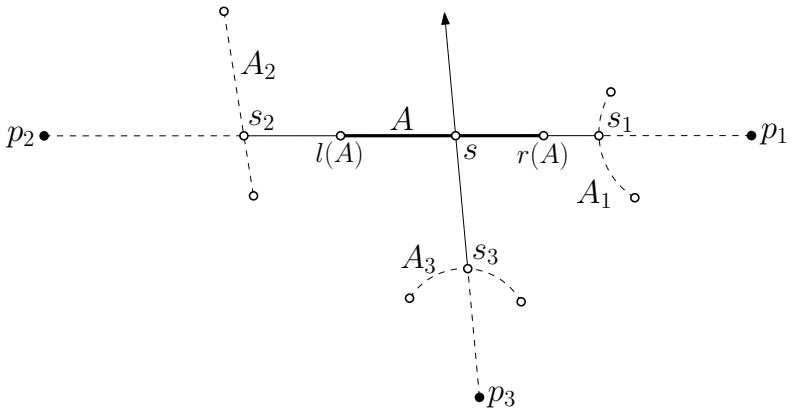


FIGURE 3.3: Three branches $\mathcal{B}_1$, $\mathcal{B}_2$ and $\mathcal{B}_3$ are triple-merged, with $\mathcal{B}_3$ as the source branch.

Now, we present the pseudocode for the generation phase.

Algorithm 1: Generation Phase

Input: A set of terminals N and a non-negative integer k.

Output: A set $\mathcal{F} = \{\mathcal{F}_0, \mathcal{F}_1, ..., \mathcal{F}_k\}$ where $\mathcal{F}_i$ contains candidate FSTs with i Steiner points.

```
1  Let F_0 := E(T) where T is an MST on N.
2  Let F_i := ∅ for i ∈ {1, 2, ..., k}.
3  Let B_i := ∅ for i ∈ {0, 1, ..., k}.
4  for t_p ∈ N and p ≠ n do
5      Generate a branch on t_p and add it to B_0.
6  for i ∈ {1, 2, ..., k} do
7      for x ∈ {0, 1, ..., ⌊(i − 1)/2⌋} do
8          for B_p ∈ B_x do
9              for B_q ∈ B_{i−1−x} where p < q if x = i − 1 − x do
10                 Double-merge (with pruning) B_p and B_q.
11                 Add the resulting branch to B_i.
12     for x ∈ {0, ..., ⌊(i−1)/3⌋} do
13         for y ∈ {x, ..., ⌊(i−1−x)/2⌋} do
14             for B_p ∈ B_x do
15                 for B_q ∈ B_y where p < q if x = y do
16                     for B_r ∈ B_{i−1−x−y} where q < r if y = i − 1 − x − y do
17                         Triple-merge (with pruning) B_p, B_q and B_r.
18                         Add the resulting branch to B_i.
19     for B_p ∈ B_i do
20         for t ∈ N do
21             Termination merge (with pruning) B_p and t to generate an FST.
22             Add the resulting FST to F_i.
```

We now briefly discuss Algorithm 1.

In the generation phase, we are given a set of terminals $N = \{t_1, t_2, ..., t_n\}$ and a non-negative integer k. Our aim is to find a set of candidate FSTs that include the FSTs of a minimum k-Steiner tree. We define $\mathcal{F} = \{\mathcal{F}_0, \mathcal{F}_1, ..., \mathcal{F}_k\}$ where $\mathcal{F}_i$ contains candidate

FSTs with i Steiner points. Furthermore, we define B_i to be the set of branches that contain i Steiner points for $i = 0, 1, ..., k$.

We begin by adding the edges of an MST to $\mathcal{F}_0$, as any edge between a pair of terminals in a minimum k-Steiner tree is also an edge of a minimum spanning tree on N (Line 1). This follows from the argument given for minimum Steiner trees in [13]. The replacement procedure given in the proof does not require any additional Steiner points and therefore the result also applies to minimum k-Steiner trees. In Line 2, the other sets of FSTs are initialised as empty sets. Line 3 initiates B_i for $i \in \{0, 1, ..., k\}$ as empty sets. In Lines 4 - 5, we generate branches at each of the terminals except at the terminal with the largest index. A branch at the terminal with the largest index is unnecessary due to the mechanism of the termination merge which we discuss below.

Now, we generate branches with i Steiner points where $i \in \{1, 2, ..., k\}$ by double-merging (Lines 7 - 11) and triple-merging (Lines 12 - 18). Then, we generate FSTs with i Steiner points by termination merging (Lines 19 - 22).

Note that during each merging step (Lines 10, 17 and 21), we run basic feasibility checks as well as pruning tests. Here, we outline the feasibility checks that are employed during the merging process. Firstly, branches may be double-merged or triple-merged only if the terminals they span are mutually exclusive. Otherwise, this leads to a creation of a cycle. Furthermore, the union of the subset of terminals spanned by the branches cannot be N, as this generates a branch spanning N. This branch cannot be termination merged as there are no remaining terminals and is therefore redundant. Suppose $\mathcal{B}_1$ and $\mathcal{B}_2$ are two branches spanning the subset of terminals $N_{\mathcal{B}_1}$ and $N_{\mathcal{B}_2}$ respectively. Then, a double-merge between $\mathcal{B}_1$ and $\mathcal{B}_2$ is infeasible if either $N_{\mathcal{B}_1} \cap N_{\mathcal{B}_2} \neq \emptyset$ or $N_{\mathcal{B}_1} \cup N_{\mathcal{B}_2} = N$.

The restrictions in the pseudocode on branches in which B_is can be merged with each other is effectively a feasibility check based on the bound k, in other words, it ensures that no branch (and consequently FST) ever contains more than k Steiner points.

In our algorithm, we allow a branch and a terminal to be termination merged only if the index of the terminal is higher than that of any terminal spanned by the branch. For example, suppose that there exists a branch $\mathcal{B}$ and a terminal t_i. Then, a termination merge is feasible only if $i > \max\{j | t_j \in N_{\mathcal{B}}\}$. This index-based restriction helps mitigate the issue of generating duplicate FSTs. Any branch spanning t_n (the terminal with the

largest index) cannot be termination merged and hence we do not generate a branch on that terminal (Lines 4 - 5).

Finally, we note that during a double-merge or a triple-merge, we consider the distinct ways in which the branches can be merged. For example, two branches $\mathcal{B}_1$ and $\mathcal{B}_2$ (with pseudoterminals p_1 and p_2) can be double-merged in two different ways, resulting in two distinct branches: one whose pseudoterminal is $e_{p_1p_2}$, and another whose pseudoterminal is $e_{p_2p_1}$. Similarly, when branches $\mathcal{B}_1$, $\mathcal{B}_2$, and $\mathcal{B}_3$ (with pseudoterminals p_1, p_2, and p_3) are triple-merged, there are three distinct possibilities, as the pseudoterminal of any one of the three branches can serve as the pseudoterminal of the resulting branch. Consequently, a double-merge may generate up to two branches, while a triple-merge may generate up to three. In practice, the feasibility tests and the pruning tests help control this potential exponential explosion of branches.

3.3 Pruning tests

The difficulty in solving the minimum k-Steiner tree problem stems from the vast number of candidate FSTs that potentially belong to an optimal solution. In order to address this, we implement *pruning tests* during the generation phase. Pruning tests utilise geometric properties of minimum k-Steiner trees (such as the Trapezium Theorem 2.2.8) to verify the potential optimality of branches and FSTs after every merge, and discard any that do not satisfy the test. When a branch is discarded, we no longer consider any branches or FSTs that may arise from this particular branch. This results in a much more efficient generation of the candidate FSTs. Furthermore, a smaller number of FSTs translates to a smaller input size for the concatenation phase. Overall, pruning tests are crucial for improving the efficiency of both phases of the algorithm.

In this section, we first discuss the projection test, as well as its various improvements specifically for a triple-merge in Subsection 3.3.1. Subsections 3.3.2 and 3.3.3 cover the bottleneck test and the lune test for a triple-merge respectively. We then explore how the 4-lune theorems (Theorems 2.2.4 and 2.2.5) may be implemented as a dynamic pruning test. Finally, we illustrate how the Rhombus Theorem (2.2.7) and Trapezium Theorem (2.2.8) are implemented as a pruning test in Subsection 3.3.5.

3.3.1 Projection test

The projection test is a fundamental pruning test that ensures that a new branch generated from either a double-merge or a triple-merge does not violate the angle requirements of the merged branches. Consequently, this pruning test prevents degenerate topologies from being generated.

Double-merge projection testing

We consider the case where two branches $\mathcal{B}_1$ and $\mathcal{B}_2$ with pseudoterminals p_1, p_2 and roots s_1, s_2 are double-merged, generating a new branch $\mathcal{B}$ with Steiner arc A. Let the Steiner curve associated with branch $\mathcal{B}_i$ be denoted A_i. Finally, let $p = e_{p_1p_2}$ be the pseudoterminal of $\mathcal{B}$. Note that A is a subarc of the minor arc $\widehat{p_2p_1}$ of the circle through p, p_1 and p_2. The notation $\widehat{ab}$ denotes an arc from point a to point b on a specified circle. The underlying circle will be clear from context or explicitly stated when needed.

The aim of this projection test is to find the largest subarc $\widehat{l(A)r(A)}$ of $\widehat{p_2p_1}$ such that for all $s \in \widehat{l(A)r(A)}$, and for each $i \in \{1, 2\}$ the ray $\overrightarrow{p_is}$ intersects A_i strictly between p_i and s. An example where A_1 is a Steiner segment and A_2 is a Steiner arc is shown in Figure 3.1.

When A_1 and A_2 are Steiner arcs, the double-merge projection test is discussed in detail in [13]. Generally, the projection test can be applied in a similar manner even if one or both of the A_i are Steiner segments. However, a special case may arise if A_i intersects A twice, which can only occur when A_i is a Steiner segment. In this case, we may prune the entire branch, as stated in the following proposition. Note that the proposition establishes a stronger result, showing that if the line through A_i (not just A_i itself) intersects A twice, then $\mathcal{B}$ is not optimal (i.e., cannot lead to an FST that occurs in a minimum k-Steiner tree).

Proposition 3.3.1. Suppose A_1 is a Steiner segment. If branch $\mathcal{B}$ is optimal then $\overline{l(A_1)r(A_1)}$ does not intersect $\widehat{p_2p_1}$ twice.

Proof. Assume by contradiction that $\overline{l(A_1)r(A_1)}$ intersects $\widehat{p_2p_1}$ twice. Then the segment p_2s must intersect $\overline{l(A_1)r(A_1)}$ at say, y (shown in Figure 3.4). Since the angle $\angle p_2sp_1 = 2\pi/3$, we must have $\angle ss_1y < \pi - 2\pi/3 = \pi/3$. Hence, there exists an angle that is strictly less than $\pi/3$, giving a contradiction by Proposition 2.1.5. □

In practice, we do not check this explicitly, as the angle condition (Proposition 2.1.5) is already verified during a triple-merge.

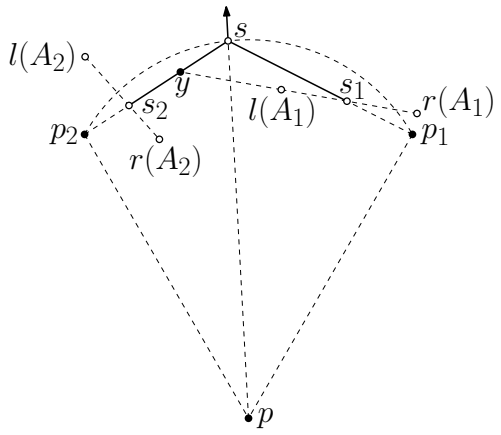


FIGURE 3.4: If $\overline{l(A_1)r(A_1)}$ intersects $\widehat{p_2p_1}$ twice, the branch is not optimal.

Triple-merge projection testing

Suppose that three branches $\mathcal{B}_1$, $\mathcal{B}_2$ and $\mathcal{B}_3$ are triple-merged (with $\mathcal{B}_3$ as the source branch), creating a new branch $\mathcal{B}$ with Steiner segment A. We denote the root, the pseudoterminal and the Steiner curve of branch $\mathcal{B}_i$ by s_i, p_i and A_i respectively. Firstly, we must ensure that $\mathcal{B}_1$ and $\mathcal{B}_2$ can undergo termination merge (since otherwise the new branch is pruned entirely due to infeasibility), i.e., the line segment p_1p_2 must intersect A_1 and A_2 at s_1 and s_2 respectively, where s_1 lies strictly between p_1 and s_2 and s_2 strictly between s_1 and p_2. We also test that the condition in Proposition 2.1.5 is satisfied.

Let $A = s_1s_2$ be the initial Steiner segment for $\mathcal{B}$. We must find the subsegment $A' \subseteq A$ where A' consists of all points $s \in A$ such that $\overrightarrow{p_3s}$ intersects A_3 strictly between p_3 and A. To achieve this, we first find the curve $A'_3 \subseteq A_3$ that lies in the closed half-plane with boundary $\overline{s_1s_2}$ and containing vertex p_3; see Figure 3.5. Then, we project this subarc from p_3 onto the Steiner arc A.

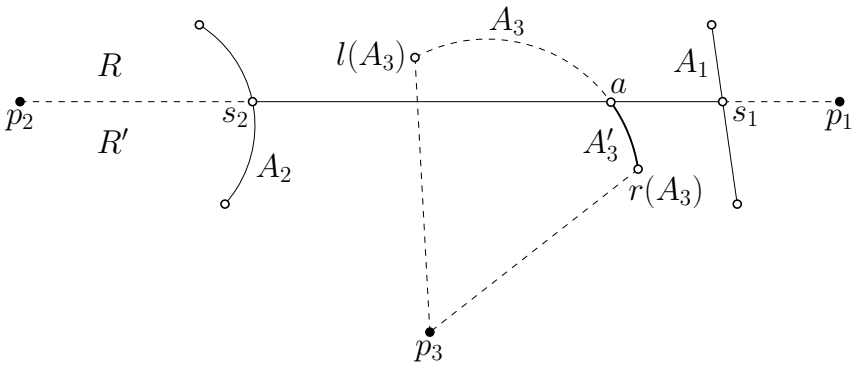


FIGURE 3.5: A figure depicting the subarc A_3' of A_3.

There are two main cases for finding A_3' depending on whether the source curve A_3 is an arc or a segment.

Case 1: The source curve A_3 is an arc.

Firstly, we define the regions R and R'. The line $\overline{s_1 s_2}$ divides the plane into two half-planes. Let R' denote the open half-plane containing p_3 and R the complement of R'. This is shown above in Figure 3.5.

This case has a number of subcases. We begin by proving a proposition that eliminates a specific subcase where both $l(A_3), r(A_3) \in R$.

Proposition 3.3.2. If $l(A_3), r(A_3) \in R$, then the arc A_3 does not project onto A.

Proof. Since $l(A_3), r(A_3) \in R$ and $p_3 \in R'$, the circle that contains the source arc A_3 must intersect $\overline{s_1 s_2}$ on the major arc $l(\widehat{A_3)p_3 r(A_3)}$. Hence, the entire minor arc $A_3 = l(\widehat{A_3)r(A_3)}$ must lie inside R. For A_3 to project onto A, there must be some point $x \in A_3 \cap R'$ and hence it follows immediately. □

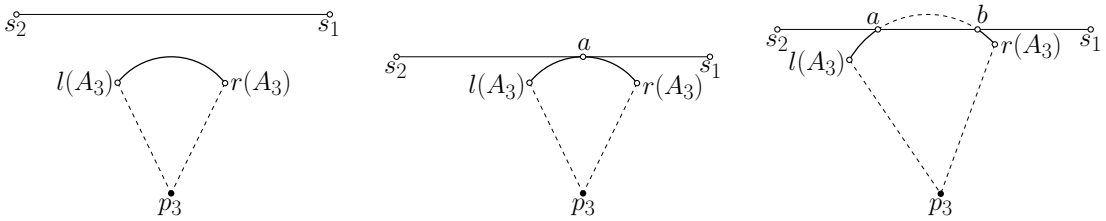
If exactly one of $l(A_3)$ and $r(A_3)$ lies inside R', say $r(A_3)$ (without loss of generality), then $l(\widehat{A_3)r(A_3)}$ intersects $\overline{s_1 s_2}$ once at say, a. Then $A_3' = \widehat{ar(A_3)}$, as shown in Figure 3.5.

Finally, if both $l(A_3)$ and $r(A_3)$ lie inside R', then there are three possible subcases, shown below in Figure 3.6.

(a) There are no intersection points. In this case, $A_3' = A_3$.

(b) There is a single intersection point.

(c) There are two intersection points, say a and b (where $\angle l(A_3)p_3a < \angle l(A_3)p_3b$).



(a) The arc A_3 does not intersect $\overline{s_1s_2}$.

(b) The arc A_3 intersects $\overline{s_1s_2}$ once at a.

(c) The arc A_3 intersects $\overline{s_1s_2}$ twice at a and b.

FIGURE 3.6: The three subcases for Case 1.

In the third Subcase (c), where there are two intersection points, there are two disjoint subarcs $\widehat{l(A_3)a}$ and $\widehat{br(A_3)}$ that satisfy the projection test. Storing both valid subarcs may lead to a proliferation in the number of branches, whereas storing the entire arc may result in the generation of degenerate topologies that do not satisfy the projection test. Hence, we show that while both subarcs satisfy the projection test in this case, only one of them may form an optimal tree and hence the other can be discarded during the projection test. Furthermore, we include the second subcase in the following theorem, by letting $a = b$ be the single intersection point.

Theorem 3.3.3. Suppose the source arc $l(\widehat{A_3)r(A_3})$ in a triple-merge either intersects the line $\overline{s_1s_2}$ twice at a and b (where a lies on the minor arc $\widehat{l(A_3)b}$) or once at $a = b$. Then both subarcs $\widehat{l(A_3)a}$ and $\widehat{br(A_3)}$ are in R', but only one of them can contain an optimal Steiner point.

Proof. Let T denote the triangular region defined by the Trapezium Theorem (Theorem 2.2.8) and the Rhombus Theorem (Theorem 2.2.7) where s_3 must lie, as shown in Figure 3.7. Our aim is to show that $\widehat{l(A_3)a}$ and $\widehat{br(A_3)}$ cannot both have nonempty intersection with the region T.

Assume otherwise. We assume that both $\widehat{l(A_3)a}$ and $\widehat{br(A_3)}$ have a nonempty intersection with T.

Firstly, we note that angle $\angle l(A_3)ar(A_3)$ is the same regardless of where a lies on $l(\widehat{A_3)r(A_3})$. Since A_3 is a Steiner arc, $\angle l(A_3)ar(A_3) \geq 2\pi/3$.

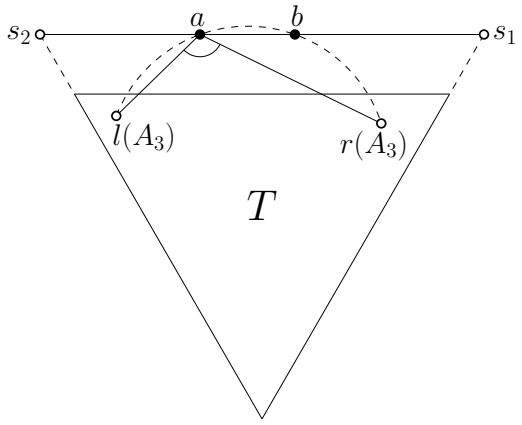


FIGURE 3.7: Angle $\angle l(A_3)ar(A_3)$ must be at least $2\pi/3$ since A_3 is a Steiner arc.

Secondly, we assume that $l(A_3), r(A_3) \in T$. Otherwise, if $l(A_3) \notin T$ and $\widehat{l(A_3)a}$ has a nonempty intersection with T, there exists some $l(A_3)' \in \widehat{l(A_3)a} \cap T$. Since $\angle l(A_3)ar(A_3) < \angle l(A_3)'ar(A_3)$ (as shown in Figure 3.8), it suffices to show that

$$\angle l(A_3)'ar(A_3) < 2\pi/3$$

to disprove our initial assumption. Hence, we let $l(A_3) := l(A_3)'$ and now $l(A_3) \in T$. Similarly, we may assume that $r(A_3) \in T$. Now, we want to show that $\angle l(A_3)ar(A_3) < 2\pi/3$, assuming that $l(A_3), r(A_3) \in T$.

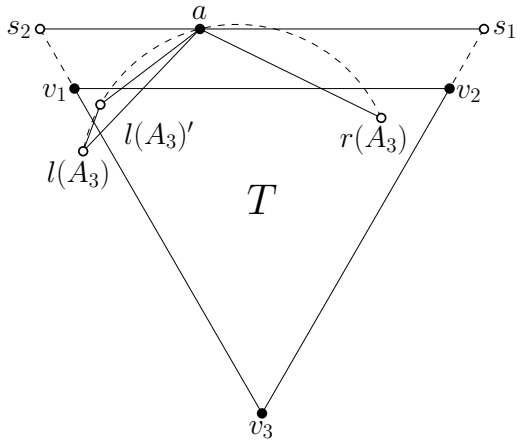


FIGURE 3.8: Given $l(A_3) \notin T$, we can find some $l(A_3)' \in T \cap \widehat{l(A_3)a}$.

Let $s_1 = (1, 0)$ and $s_2 = (0, 0)$. Then $a = (x, 0)$. Let the vertices of T be denoted by $v_1 = (\frac{1}{6}, \frac{-1}{2\sqrt{3}})$, $v_2 = (\frac{5}{6}, \frac{-1}{2\sqrt{3}})$ and $v_3 = (\frac{1}{2}, \frac{-\sqrt{3}}{2})$ (shown above in Figure 3.8). When $x \leq 0$, the angle $\angle l(A_3)ar(A_3)$ is at its largest when $l(A_3) = v_3$ and $r(A_3) = v_2$, as T is entirely contained within the cone defined by the rays $\overrightarrow{av_3}$ and $\overrightarrow{av_2}$. So we let

$l(A_3) = v_3$ and $r(A_3) = v_2$ and since $x \leq 0$, the angle $\angle l(A_3)ar(A_3)$ is at its largest when $a = (0, 0)$, which leads to $\angle l(A_3)ar(A_3) = \angle v_3 a v_2 = \arccos(\frac{2}{\sqrt{7}}) < 2\pi/3$. By symmetry, this is also the case when $x \geq 1$. Similarly, if $0 \leq x \leq 1$, the angle is at its largest when $l(A_3) = v_1$ and $r(A_3) = v_2$ (since T is inside the cone defined by arcs $\overrightarrow{av_1}$ and $\overrightarrow{av_2}$). Once $l(A_3)$ and $r(A_3)$ are fixed, it is evident that $x = 1/2$ creates the largest angle of $\angle l(A_3)ar(A_3) = \arccos(\frac{-1}{7}) < 2\pi/3$.

Hence, if $l(A_3), r(A_3) \in T$ and $a \in \overline{s_1 s_2}$, we have shown that $\angle l(A_3)ar(A_3) < 2\pi/3$, so the source arc is infeasible. Hence, we have shown that subarcs $\overset{\frown}{l(A_3)a}$ and $\overset{\frown}{br(A_3)}$ cannot both have a nonempty intersection with T, proving the result. □

The above proof also allows us to immediately identify which (if either) of the two subarcs can contain an optimal Steiner point, namely the one that intersects T.

Case 2: The source curve A_3 is a segment.

In this case, obtaining the subsegment A_3' is straightforward. If $l(A_3), r(A_3) \in R$, then $A_3 \in R$ hence the triple-merge is infeasible. If both endpoints of A_3 lie in R', then $A_3' = A_3$. Otherwise, one endpoint is in R' and the other in R. Without loss of generality, let $l(A_3) \in R'$. Then A_3 must intersect $\overline{A}$ once, at say, a. Then $A_3' = l(A_3)a$.

3.3.2 Bottleneck test

In the GeoSteiner Algorithm, the bottleneck Steiner distance bound (BSDB) test [13] is an effective pruning technique based on the *matroid property* of minimum spanning trees; namely, the fact that the distance between any pair of vertices u, v in an MST, say T, is no shorter than the length of any edge on the path in T connecting u and v [28]. The BSDB places a bound on the length of every edge in a minimum k-Steiner tree and hence, any branch tree that contains an edge that does not satisfy the bound can be discarded during the generation phase.

In this subsection, we extend the BSDB test to triple-merges. Firstly, we define the bottleneck Steiner distance between two terminals.

Definition 3.3.4. Given two terminals $t_1, t_2 \in N$, the *bottleneck Steiner distance*, $\text{BSD}(t_1, t_2)$, is equal to the length of the longest edge on the unique path between t_1 and t_2 in some minimum spanning tree on N.

Note that this definition does not depend on the choice of MST, even if there are multiple different ones for the given terminals.

Theorem 3.3.5. Any edge on the unique t_1-t_2 path in some minimum k-Steiner tree T for N has its length bounded by the bottleneck Steiner distance, $\mathrm{BSD}(t_1, t_2)$.

The proof of the above theorem is identical to the one found in Optimal Interconnection Trees in the Plane [13] for minimum Steiner trees and hence will be omitted.

Suppose that three branches $\mathcal{B}_1$, $\mathcal{B}_2$ and $\mathcal{B}_3$ are triple-merged (with $\mathcal{B}_3$ as the source branch) to form a new branch $\mathcal{B}$ with pseudoterminal p and Steiner segment A. Let the root and Steiner curve of branch $\mathcal{B}_i$ be denoted by s_i and A_i respectively for $i \in \{1, 2, 3\}$. A new degree-4 Steiner point $s \in A$ and three new edges s_1s, s_2s and s_3s are generated in this process. For each $i \in \{1, 2, 3\}$, let N_i be the set of terminals in $\mathcal{B}_i$. There may be multiple paths between pairs of terminals in $\mathcal{B}$ that use the edges s_1s, s_2s or s_3s. For instance, any path in $\mathcal{B}$ from a terminal in N_1 to a terminal in N_3 contains the edges s_1s and s_3s (see Figure 3.9 for an example). The bottleneck Steiner distance of these terminal pairs then give an upper bound on the length of these edges.

Our aim is to find the largest connected subsegment $A' = r(A')l(A')$ of A so that, for any $s \in A'$, edges s_1s, s_2s and s_3s satisfy their respective BSDBs. We first define the BSDB between two subsets of terminals as follows:

Definition 3.3.6. The bottleneck Steiner distance between two subsets of terminals N_1 and N_2 is defined to be $\mathrm{BSD}(N_1, N_2) = \min_{t_1 \in N_1, t_2 \in N_2} \mathrm{BSD}(t_1, t_2)$.

Definition 3.3.7. For any edge uv of a branch tree or Steiner tree T, let N_u and N_v be the terminals for the two connected components of $T \setminus uv$ containing u and v respectively. Then we define the *bottleneck Steiner distance bound* (BSDB) of uv to be $\mathrm{BSD}(N_u, N_v)$.

In $\mathcal{B}$, the edge s_1s lies on each $u-v$ path for $u \in N_1$ and $v \in N_2 \cup N_3$. Hence, the length of edge s_1s is bounded by $\mathrm{BSD}(N_1, N_2 \cup N_3)$. Similarly, $|s_2s| \leq \mathrm{BSD}(N_2, N_1 \cup N_3)$ and $|s_3s| \leq \mathrm{BSD}(N_3, N_1 \cup N_2)$.

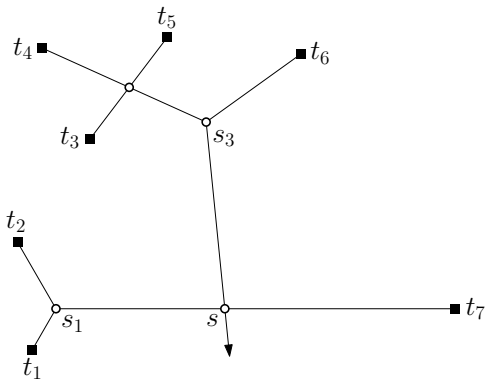


FIGURE 3.9: In the branch tree shown, any path from a terminal in $N_1 = \{t_1, t_2\}$ to a terminal in $N_3 = \{t_3, t_4, t_5, t_6\}$ contains the edges $s_1 s$ and $s_3 s$.

Our algorithm is comprised of two rounds of pruning. In the first round of pruning, we restrict s to $A' = r(A')l(A')$, so that $s_1 s$ and $s_2 s$ satisfy their respective BSDBs. Then we redefine $A := A'$ and further restrict A to a new $A' = r(A')l(A')$ so that $s_3 s$ satisfies the BSDB in the second round of pruning. In this step, we only consider the case where A_3 is an arc (i.e., p_3 was constructed during a double-merge operation), as the procedure described below relies on a key property stated in Theorem 3.3.8, which requires A_3 to be an arc.

The first round of pruning is straightforward. If $|s_1 l(A)| > \mathrm{BSD}(N_1, N_2 \cup N_3)$, we select $l(A')$ on A so that $|s_1 l(A')| = \mathrm{BSD}(N_1, N_2 \cup N_3)$. Then $|s_1 s| \leq \mathrm{BSD}(N_1, N_2 \cup N_3)$ for all $s \in r(A)l(A')$. Similarly, we select $r(A')$ on A so that $|s_2 r(A')| = \mathrm{BSD}(N_2, N_1 \cup N_3)$. Then, if $s \in s_2 r(A')$, it follows that $|s_2 s| \leq \mathrm{BSD}(N_2, N_1 \cup N_3)$. Note that if $|s_1 l(A')| < |s_1 r(A')|$, the intersection of these two intervals are empty. In this case, there are no branch trees for which $s_1 s$ and $s_2 s$ satisfy their respective BSDBs and hence, we discard this branch. Otherwise, we obtain a new interval $A' = r(A')l(A')$.

As described above, we now let $A := A'$, $r(A) := r(A')$ and $l(A) := l(A')$. In the second round of pruning, the location of s (and s_3) for which $|s_3 s| = \mathrm{BSD}(N_3, N_1 \cup N_2)$ does not seem to have a simple analytical expression. However, a numerical method for solving the equation can be found using the following theorem.

Theorem 3.3.8. The distance between a degree-4 Steiner point s on the Steiner segment $s_1 s_2$ and the adjacent Steiner point s_3 on the arc A_3 is a convex function with respect to the angle at the pseudoterminal p_3.

Proof. Let the Steiner arc A_3 be part of a circle C with centre a and radius r and let the Steiner segment A be part of a line L. Without loss of generality, we translate and rotate the plane so that p_3 is at the origin and line L is vertical and to the right of p_3. Let $(l, 0)$ be the intersection point of L and the x-axis, where $l > 0$. Denote the ray $\overrightarrow{p_3 s}$ by S. Note that S intersects C at s_3 and L at s since s is a projection of s_3.

We use an angle range of $(-\pi, \pi]$ where positive angles are measured anticlockwise from the positive x-axis. Let the angle of segment p_3a be δ and the angle of the ray S be θ; see Figure 3.10. Note that $\theta \in (-\pi/2, \pi/2)$, because otherwise, S cannot intersect L and hence also not A. Let $h(\theta) = |p_3 s|$ and let $c(\theta) = |p_3 s_3|$. Then, our aim is to show that $h(\theta) - c(\theta)$ is a convex function with respect to θ.

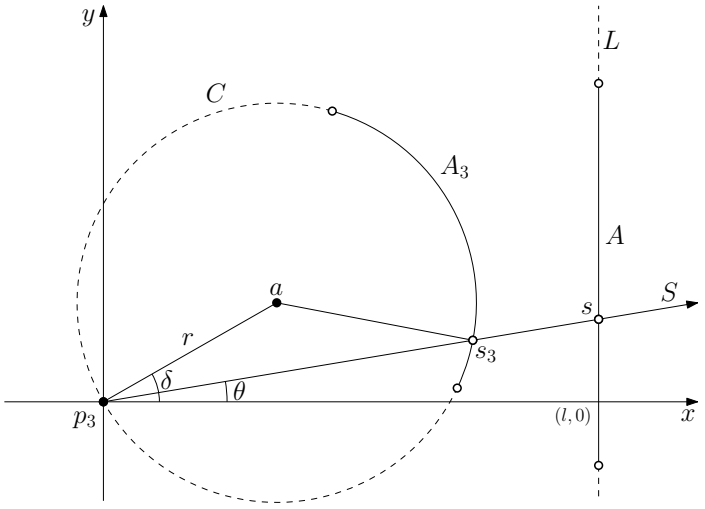


FIGURE 3.10: Functions $h(\theta)$ and $c(\theta)$ are defined to be $|sp_3|$ and $|s_3p_3|$ respectively.

Observe that $h(\theta) = l \sec\theta$, which is a convex function for $\theta \in (-\pi/2, \pi/2)$. The function $c(\theta)$ is the length of the chord subtended by an angle of $\pi - 2|\delta - \theta|$. Hence, $c(\theta) = 2r \sin((\pi - 2|\delta - \theta|)/2) = 2r \cos(\theta - \delta)$.

The first and second derivatives of $c(\theta)$ are given by: $c'(\theta) = -2r \sin(\theta - \delta)$ and $c''(\theta) = -2r \cos(\theta - \delta)$. For S to intersect A_3, $\theta \in [\delta - \pi/6, \delta + \pi/6]$ and hence $c(\theta)$ is continuous and $c''(\theta) < 0$. Therefore, $c(\theta)$ is a concave function. Since $h(\theta)$ is convex and $c(\theta)$ is concave, $h(\theta) - c(\theta)$ must be a convex function. Hence the theorem follows. □

The solutions to the equation $|s_3 s| = \mathrm{BSD}(N_3, N_1 \cup N_2)$ can now be found numerically within a desired accuracy ϵ by using root-finding algorithms. Due to its simplicity and efficacy, we use the bisection method in our approach. By Theorem 3.3.8, there are at

most two points $s \in A$ which satisfy the equation. These solutions, together with the endpoints of A, yield a connected interval A' that satisfies the BSDB for ss_3.

The bisection method is used in two different ways. If there is a single solution to the equation $|s_3 s| = \text{BSD}(N_3, N_1 \cup N_2)$ in some interval, then we use bisection to reduce the interval length down to 2ϵ. We call this *standard bisection*. In the second method, we use bisection to partition an interval into two sub-intervals sharing a common endpoint, where each sub-interval contains one solution. We refer to this process as *bisection partition*.

We now describe in more detail the cases that arise and how they determine the use of the two bisection methods. We assume (by reducing A_3 or A, if necessary) that A is the projection from p_3 of A_3 onto the interval $s_1 s_2$. Let s_3^r and s_3^l be the Steiner points on A_3 which project onto the points $r(A)$ and $l(A)$ (respectively) on the Steiner segment A from p_3 (as shown below in Figure 3.11).

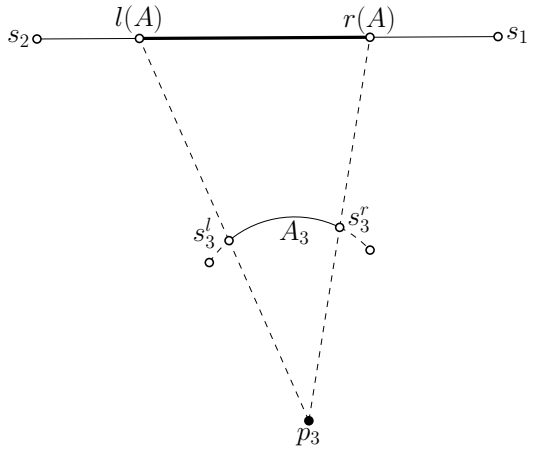


FIGURE 3.11: The points s_3^r and s_3^l project onto the points $r(A)$ and $l(A)$ on the Steiner segment A from p_3.

Let $\alpha = \text{BSD}(N_3, N_1 \cup N_2)$. There are three possible cases depending on the lengths $|s_3^r r(A)|$ and $|s_3^l l(A)|$ relative to α:

Case 1: $|s_3^r r(A)| \leq \alpha$ **and** $|s_3^l l(A)| \leq \alpha$
In this case, $|s_3 s| \leq \alpha \ \forall \ s \in A$ so A cannot be reduced.

Case 2: $(|s_3^r r(A)| \leq \alpha$ **and** $|s_3^l l(A)| > \alpha)$ **or** $(|s_3^r r(A)| > \alpha$ **and** $|s_3^l l(A)| \leq \alpha)$

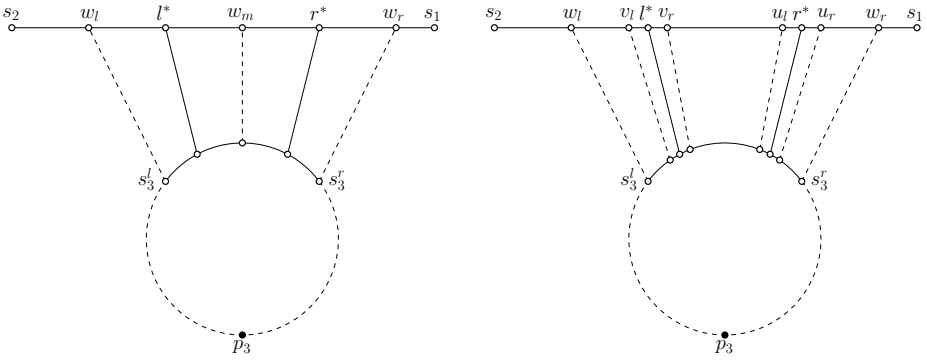
In this case, there is exactly one solution to the equation $|s_3 s| = \alpha$. Without loss of generality, let $|s_3^r r(A)| \leq \alpha$ and $|s_3^l l(A)| > \alpha$. Standard bisection then provides an

interval $u_r u_l$ of length at most 2ϵ containing the solution (where the subscript r denotes the endpoint that lies closer to $r(A)$, and similarly for $l(A)$). Then, the subsegment $r(A)u_l$ of A contains all the points which satisfy the BSDB.

Case 3: $|s_3^r r(A)| > \alpha$ **and** $|s_3^l l(A)| > \alpha$

If the BSDB is not satisfied at both endpoints $r(A)$ and $l(A)$, then we initially run the bisection partition method, potentially followed by standard bisection, as follows.

Let the interval in the current iteration be $w_r w_l$, with w_m as the midpoint of the interval. In the first iteration, we set $w_r := r(A)$ and $w_l := l(A)$. If the BSDB is satisfied at w_m, then $w_r w_m$ and $w_m w_l$ contain one solution each, as shown in Figure 3.12(a). Therefore, we run a standard bisection on these intervals to obtain $u_r u_l$ and $v_r v_l$ respectively (refer to Figure 3.12(b)). The interval $u_r v_l$ contains all the points satisfying the BSDB.



(a) The first iteration of bisection partition where $w_r = r(A)$ and $w_l = l(A)$. The BSDB is satisfied at the midpoint, w_m. Hence, there is one solution, r^* in the interval $w_r w_m$ and one solution l^* in the interval $w_m w_l$.

(b) Standard bisection is run on intervals $w_r w_m$ and $w_m w_l$ to obtain $u_r u_l$ and $v_r v_l$ respectively. The interval $u_r v_l$ contains all the points satisfying the BSDB.

FIGURE 3.12: An example of bisection partition and standard bisection being used in tandem.

If the BSDB is not satisfied at w_m, we check the gradient of the length at w_m (which, in the notation of Theorem 3.3.8 and its proof, is given by $h'(\theta) - c'(\theta)$). If this gradient is positive, we let $w_l := w_m$ and if it is negative, we let $w_r := w_m$ and repeat for the new interval. The interval is continuously halved until its midpoint satisfies the BSDB or its length is less than or equal to 2ϵ. If the midpoint of this final interval (which has a length of at most 2ϵ), say w_m of $w_r w_l$, does not satisfy the BSDB, then we let $A' := w_r w_l$.

These two bisection methods form part of the pruning test described in Algorithm 2.

Algorithm 2: The Bottleneck Pruning Test for a Triple-Merge

Input: A branch $\mathcal{B}$ resulting from a triple-merge, with Steiner segment

$$A = r(A)l(A).$$

Output: An updated feasible subsegment $A' = r(A')l(A')$.

if $|s_1 r(A)| > BSD(N_1, N_2 \cup N_3)$ **then**
  return $A' := \emptyset$.
if $|s_1 l(A)| > BSD(N_1, N_2 \cup N_3)$ **then**
  Let $l(A')$ be the point on $r(A)l(A)$ such that $|s_1 l(A')| = \mathrm{BSD}(N_1, N_2 \cup N_3)$.
else
  Let $l(A') := l(A)$.
if $|s_2 l(A')| > BSD(N_2, N_1 \cup N_3)$ **then**
  return $A' := \emptyset$.
if $|s_2 r(A)| > BSD(N_2, N_1 \cup N_3)$ **then**
  Let $r(A')$ be the point on $r(A)l(A')$ such that $|s_2 r(A')| = \mathrm{BSD}(N_2, N_1 \cup N_3)$.
else
  Let $r(A') := r(A)$.
if $|s_1 l(A')| < |s_1 r(A')|$ **then**
  return $A' := \emptyset$.
Let $A := A'$, $r(A) := r(A')$ and $l(A) := l(A')$.
if $|s_3^r r(A)| > BSD(N_3, N_1 \cup N_2)$ *and* $|s_3^l l(A)| > BSD(N_3, N_1 \cup N_2)$ **then**
  Perform bisection partition method on $r(A)l(A)$.
  if *output of bisection partition is two intervals* **then**
    Run standard bisection on $w_r w_m$, $w_m w_l$ to obtain $u_r u_l$, $v_r v_l$.
    Let $r(A') := u_r$ and $l(A') := v_l$
  else
    Let $r(A') := w_r$ and $l(A') := w_l$
else if $|s_3^r r(A)| > BSD(N_3, N_1 \cup N_2)$ *and* $|s_3^l l(A)| \leq BSD(N_3, N_1 \cup N_2)$ **then**
  Run standard bisection on $r(A)l(A)$ to obtain $u_r u_l$.
  Let $r(A') := u_r$ and $l(A') := l(A)$
else if $|s_3^r r(A)| \leq BSD(N_3, N_1 \cup N_2)$ *and* $|s_3^l l(A)| > BSD(N_3, N_1 \cup N_2)$ **then**
  Run standard bisection on $r(A)l(A)$ to obtain $u_r u_l$.
  Let $r(A') := r(A)$ and $l(A') := u_l$

In the Bottleneck Steiner Distance Bound Algorithm, the first phase of pruning involves using the BSDB of s_1s and s_2s. The second phase of pruning involves using the BSDB of s_3s.

We now discuss the complexity of the algorithm. Note that the BSDB between every pair of terminals is calculated at the start of the algorithm as preprocessing. The complexity of finding the shortest paths from one terminal to every other terminal is at most $n \log n$ by Dijkstra's Algorithm. Since this process is repeated for all n terminals, the total complexity is at most $n^2 \log n$. Calculating $\text{BSD}(N', N'')$ for some $N', N'' \subseteq N$ for which $N' \cap N'' = \emptyset$ then takes $O(n^2)$ time, as it is assumed that it requires constant time to perform a table look-up of preprocessed values.

In the first phase of pruning, calculating the length of line segments and finding points on the line segment takes a constant amount of time. In the second phase of pruning, the maximum number of iterations run in both the bisection partition algorithm and the standard bisection algorithm is finite and depends on the error term ϵ. In the worst case, bisection partition runs until the interval length is at most 2ϵ. If the length of the starting interval is l, then this requires at most $\lceil \log_2(l/2\epsilon) \rceil$ iterations.

In turn, each iteration of standard bisection and bisection partition has a constant number of evaluations of $f(\theta)$ and $f'(\theta)$, where $f(\theta) = \frac{l}{\cos\theta} - 2r\cos(\theta - \delta)$ and $f'(\theta) = \frac{l\sin\theta}{\cos^2\theta} + 2r\sin(\theta - \delta)$. Hence, the BSDB Algorithm runs in $n^2 \log n$ time.

The correctness of the algorithm follows directly from the discussion in the preceding paragraphs. Note that in the second phase of pruning, in general, the interval A' is a strict superset of the smallest connected interval satisfying BSDB. This is due to the error term ϵ employed in the bisection test.

3.3.3 Lune test

Recall that given a line segment uv, the lune of uv (denoted by $L(u, v)$), is defined to be the region consisting of all points x such that $|ux| < |uv|$ and $|vx| < |uv|$ (Definition 2.1.8). Furthermore, a minimum Steiner tree satisfies the *lune property*, which says that the lune of every edge in the tree does not contain any terminals (see eg., [29] and [13]); otherwise, one can obtain a shorter Steiner tree by replacing some edges with new ones.

This process does not create additional Steiner points, so this proof can also be applied to minimum k-Steiner trees (Theorem 2.1.9).

Since a minimum k-Steiner tree must satisfy the lune property, branch trees that do not can be discarded during the generation phase. The GeoSteiner Algorithm prunes branch trees generated from double-merging that do not satisfy this property [13]. In this subsection, we develop an algorithm that extends this to triple-merges.

Suppose that three branches $\mathcal{B}_1$, $\mathcal{B}_2$ and $\mathcal{B}_3$ are triple-merged (where, as usual, $\mathcal{B}_3$ is the source branch), creating a new degree-4 Steiner point s (as in Figure 3.3). Since edges s_1s, s_2s and s_3s are part of the new branch $\mathcal{B}$, the lunes of these edges cannot contain any terminals. In this subsection, we will restrict our attention to finding the largest connected subsegment $A' = r(A')l(A')$ of A so that, for any $s \in A'$, $L(s_1, s)$ and $L(s_2, s)$ do not contain a terminal. Recall that s_1 and s_2 are fixed points for the new branch under the triple merge.

Firstly, we focus on ensuring that there are no terminals in $L(s_1, s)$ by restricting the location of s. If s lies on $r(A)l(A)$ and s is moved towards $r(A)$, $L(s_1, s)$ shrinks while $L(s_2, s)$ expands. Hence, the size of the lune $L(s_1, s)$ is maximal when $s = l(A)$ and minimal when $s = r(A)$. In order to ensure that $L(s_1, s)$ does not contain a terminal, we first let $s = l(A)$ and then shrink $L(s_1, s)$ by moving s towards $r(A)$ until there are no terminals inside $L(s_1, s)$, or until $s = r(A)$. The following theorem ensures that no terminals can enter the lune during this process.

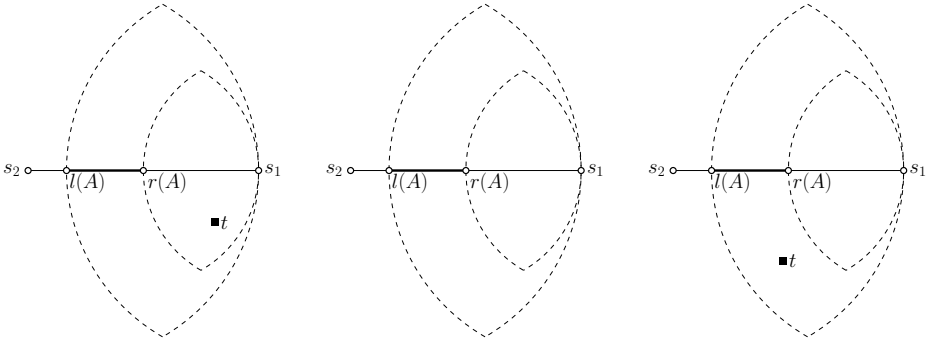
Theorem 3.3.9. Let u, v and w be three distinct and collinear points, where v lies on the line segment uw. Then $L(u, v) \subset L(u, w)$.

Proof. Let t be a point in $L(u, v)$. Then $|ut| < |uv|$ and $|vt| < |uv|$. Hence, $|ut| < |uv| < |uw|$. Also, if $|vw|$ is added to both sides of the second inequality, we obtain $|vt|+|vw| < |uv|+|vw| = |uw|$. By the triangle inequality, we obtain $|wt| \leq |vt|+|vw| < |uw|$. Hence, $|ut| < |uw|$ and $|wt| < |uw|$, which implies that t is also in $L(u, w)$. □

In our algorithm, we first check the lune $L(s_1, s)$ at the endpoints of A to determine whether the shrinking process is necessary. If the smallest possible lune, namely $L(s_1, r(A))$, contains a terminal (as shown in Figure 3.13(a)), then $L(s_1, s)$ contains a terminal for all $s \in r(A)l(A)$ by Theorem 3.3.9. Hence, s_1s never satisfies the lune property

and the branch $\mathcal{B}$ can be discarded. Otherwise, the largest possible lune $L(s_1, l(A))$ is checked to see whether it contains any terminals. If there are no terminals in $L(s_1, l(A))$, then $L(s_1, s)$ does not contain any terminals for $s \in r(A)l(A)$ by Theorem 3.3.9 (as an example, see Figure 3.13(b)). In this case, there are no branch trees that can be pruned for this lune, and we let $l(A') := l(A)$.

In the case where $L(s_1, l(A))$ contains a terminal but $L(s_1, r(A))$ does not (as shown in Figure 3.13(c)), we are required to find the endpoint $l(A') \in r(A)l(A)$ of A' such that $L(s_1, s)$ does not contain any terminals for $s \in r(A)l(A')$ and contains a terminal for every $s \in A \setminus r(A)l(A')$. In order to calculate $l(A')$, we first note that $L(s_1, l(A'))$ must have a terminal on its boundary, as any perturbation of s towards $l(A)$ from $l(A')$ will lead to a terminal inside the lune. Therefore, the shrinking process previously described, starting with the maximal lune $L(s_1, l(A))$, can be done in finite, discrete steps, rather than continuously perturbing s towards $r(A)$.



(a) Case 1: Since $t \in L(s_1, r(A))$, $t \in L(s_1, s)$ $\forall$ $s \in r(A)l(A)$ so the entire branch can be pruned.

(b) Case 2: If $L(s_1, l(A))$ contains no terminals, then $L(s_1, s)$ contains no terminals for any $s \in r(A)l(A)$ so no branch trees can be pruned for this lune.

(c) Case 3: If $L(s_1, l(A))$ contains a terminal but $L(s_1, r(A))$ does not, then pruning can be performed for this lune.

FIGURE 3.13

Specifically, for each terminal t_i inside $L(s_1, l(A))$, we find the point $s \in r(A)l(A)$ for which t_i lies on the boundary of $L(s_1, s)$. Since $t_i \in L(s_1, l(A)) \setminus L(s_1, r(A))$, such a point always exists. Of all such points, the position of the point nearest to $r(A)$ is defined to be $l(A')$. Note then, that there are no terminals in $L(s_1, l(A'))$ and also, due to Theorem 3.3.9, that any perturbation of s towards $l(A)$ from $l(A')$ results in a terminal being inside the lune $L(s_1, s)$.

To calculate the location of $s \in r(A)l(A)$ for which a given t_i lies on the boundary of $L(s_1, s)$, we first determine which side of the boundary of $L(s_1, s)$ the terminal lies on.

Lemma 3.3.10. Let t be a terminal such that $t \in L(s_1, l(A))$. Let $a \in s_1 l(A)$ be a point such that t lies on the boundary of the lune $L(s_1, a)$. Then, t lies on the arc that passes through s_1 if $\angle t s_1 s_2 > \pi/3$ and on the arc that passes through a if $\angle t s_1 s_2 < \pi/3$. If $\angle t s_1 s_2 = \pi/3$, the terminal lies on a non-smooth point of the boundary of $L(s_1, a)$.

Proof. The lemma follows directly from the fact that the two non-smooth boundary points of the lune $L(s_1, s)$ will lie on a ray from s_1 forming an angle of $\pi/3$ with $s_1 s_2$ (see Figure 3.14). □

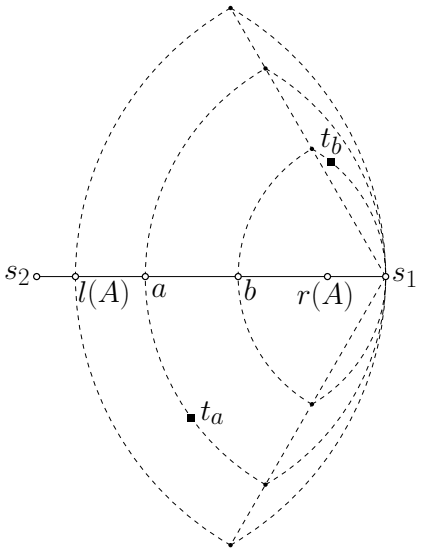


FIGURE 3.14: Terminal t_a lies on the arc that passes through a as $\angle t_a s_1 s_2 < \pi/3$ while terminal t_b lies on the arc that passes through s_1 as $\angle t_b s_1 s_2 > \pi/3$.

Once we know which side of the lune a terminal lies on, we can calculate the position of s for which the terminal lies on the boundary of $L(s_1, s)$. For each terminal t_i inside $L(s_1, l(A))$, we define $\theta_i := \angle t_i s_1 s_2$ and $d_i := |s_1 s_i'|$ where $s_i' \in r(A)l(A)$ such that t_i lies on the boundary of the lune $L(s_1, s_i')$. From Lemma 3.3.10, if $\theta_i > \pi/3$, then $|t_i s_i'| = |s_1 s_i'|$. Since triangle $\triangle t_i s_1 s_i'$ is an isosceles triangle,

$$\frac{|s_1 s_i'|}{\sin(\theta_i)} = \frac{|t_i s_1|}{\sin(\pi - 2\theta_i)} = \frac{|t_i s_1|}{\sin(2\theta_i)}.$$

Note that $\sin(\theta_i), \cos(\theta_i) \neq 0$ since $\pi/3 < \theta < \pi/2$. Hence,

$$d_i = |s_1 s_i'| = |t_i s_1| \frac{\sin(\theta_i)}{\sin(2\theta_i)} = \frac{|t_i s_1|}{2\cos(\theta_i)}.$$

If $\theta_i \leq \pi/3$ then

$$d_i = |s_1 s_i'| = |t_i s_1|$$

Let the terminal with the smallest value of d_i be t_j. Then, $l(A') := s_j'$. Now, the subsegment $r(A)l(A)$ has been reduced to $r(A)l(A')$ so that the edge $s_1 s$ satisfies the lune test for any $s \in r(A)l(A')$.

Our algorithm then uses a similar method to find $r(A') \in r(A)l(A')$ such that the lune property is satisfied for $s_2 s$ if and only if $s \in r(A')l(A')$ (or we conclude that $s_2 s$ cannot satisfy the lune property for any $s \in r(A)l(A')$).

Algorithm 3: The Lune Pruning Test for a Triple-Merge

Input: A branch $\mathcal{B}$ resulting from a triple-merge, with Steiner segment $A = r(A)l(A)$.

Output: An updated feasible subsegment $A' = r(A')l(A')$.

if $L(s_1, r(A))$ *contains a terminal* **then**
  return $A' := \emptyset$.
if $\exists$ *at least one terminal in* $L(s_1, l(A))$ **then**
  for *each terminal* $t_i \in L(s_1, l(A))$ **do**
    Let $\theta_i := \angle t_i s_1 s_2$.
    if $\theta_i > \pi/3$ **then**
      $d_i := |t_i s_1|/(2\cos\theta_i)$
    else
      $d_i := |t_i s_1|$
  $d := \min\{d_i : t_i \in L(s_1, l(A))\}$
  Let $l(A')$ be the point on $r(A)l(A)$ such that $|s_1 l(A')| = d$.
else
  $l(A') := l(A)$.
if $L(s_2, l(A'))$ *contains a terminal* **then**
  return $A' := \emptyset$.
if $\exists$ *at least one terminal in* $L(s_2, r(A))$ **then**
  for *each terminal* $t_i \in L(s_2, r(A))$ **do**
    Let $\theta_i := \angle t_i s_2 s_1$.
    if $\theta_i > \pi/3$ **then**
      $d_i := |t_i s_2|/(2\cos\theta_i)$
    else
      $d_i := |t_i s_2|$
  $d := \min\{d_i : t_i \in L(s_2, r(A))\}$
  Let $r(A')$ be the point on $r(A)l(A')$ such that $|s_2 r(A')| = d$.
else
  Let $r(A') := r(A)$.

For any given lune, it takes $O(n)$ time to scan through every terminal to check whether they lie inside the lune. Calculating d_i for each terminal t_i that lies inside either

$L(s_1, l(A))$ or $L(s_2, r(A))$ requires a constant amount of time. Therefore, the complexity of the algorithm is $O(n)$.

In order to establish the correctness of the algorithm, we show that all branch trees pruned by the first part of the algorithm (where we find the location of $l(A')$, if it exists) violate the lune property and hence, cannot be part of a minimum k-Steiner tree. Then by symmetry, this must also hold for the second part of the algorithm when we find $r(A')$. In the case where there is a terminal in $L(s_1, r(A))$, the terminal lies inside $L(s_1, s)$ for all $s \in r(A)l(A)$ by Theorem 3.3.9. Hence, the entire branch is discarded as $s_1 s$ does not satisfy the lune property for all $s \in r(A)l(A)$. If there are no terminals in $L(s_1, r(A))$ but there is at least one terminal in $L(s_1, l(A))$, we find $l(A')$, which is the point on $r(A)l(A)$ such that $L(s_1, l(A'))$ is the smallest lune which has a terminal on its boundary. Then by Theorem 3.3.9, every lune $L(s_1, s)$ with $s \in r(A)l(A) \setminus r(A)l(A')$ fails the lune property. Therefore, all branch trees that are eliminated during this process do not satisfy the lune property.

The two lunes $L(s, s_1)$ and $L(s, s_2)$ possess properties from Theorem 3.3.9 and Lemma 3.3.10 that make them straightforward to use for pruning. In contrast, the lune $L(s, s_3)$ lacks these properties and is therefore much harder to utilise effectively for pruning. Consequently, we omit $L(s, s_3)$ from the lune test.

3.3.4 4-lune test

In this subsection, we explore how 4-lunes for a degree-4 Steiner point (covered in Subsection 2.2.2) may be utilised to dynamically prune branches during the generation phase. Due to its reliance on numerous conditions and its limited pruning potential, the 4-lune pruning test is not yet implemented in our algorithm. However, with further development, this pruning test may help improve the algorithm's efficiency.

We aim to develop a dynamic pruning test using Theorems 2.2.4 and 2.2.5. The pruning test is similar to the dynamic 3-lune pruning described in [27]. Suppose that three neighbours of a degree-4 Steiner point have fixed locations while the fourth neighbour lies on a Steiner curve. If at least two of the three 4-lune inequalities (listed in 2.2.5) hold at the extreme points of a polygonal region containing the entire Steiner curve,

then these inequalities must hold for any position of the Steiner point along the curve and hence the branch can be discarded.

Suppose that branches $\mathcal{B}_1$, $\mathcal{B}_2$ and $\mathcal{B}_3$ are triple-merged (with $\mathcal{B}_3$ as the source branch), generating a branch $\mathcal{B}$. The Steiner curve, pseudoterminal and root of branch $\mathcal{B}_i$ are denoted by A_i, p_i and s_i respectively, for $i \in \{1, 2, 3\}$. There are two cases, depending on whether the next merge of branch $\mathcal{B}$ is a double-merge or a triple-merge.

Case 1: Branch $\mathcal{B}$ is double-merged.

Suppose that $\mathcal{B}$ is double-merged with branch $\mathcal{B}_a$, generating a new branch $\mathcal{B}'$ with root s' which lies on the Steiner arc $A' = l(\widehat{A')r(A'})$, as shown in Figure 3.15. The pseudoterminal of $\mathcal{B}_a$ is denoted p_a.

Let $t(A')$ be the intersection of the tangent line to A' at $l(A')$ and the line tangent to A' at $r(A')$. Finally, let T be the subtree induced by the degree-4 Steiner point s and its four neighbours. Two of its neighbours, namely the roots s_1 and s_2, have fixed locations. The third neighbour, s', has a variable position and lies on $l(\widehat{A')r(A'})$. The final neighbour, s_3, is fixed only if it is a terminal. If the root s_3 is not a terminal, two neighbours of s are not fixed and the conditions for Theorem 2.2.5 are not met. However, if s_3 is a terminal (as shown in Figure 3.15), then we can apply Theorems 2.2.4 and 2.2.5. If there exists a terminal p satisfying at least two of the following inequalities:

- $|s_1 p| < |T| - |S_{s_1}|$
- $|s_2 p| < |T| - |S_{s_2}|$
- $|s' p| < |T| - |S_{s'}|$

for $s' = l(A'), r(A')$ and $t(A')$, then we can prune the branch $\mathcal{B}'$.

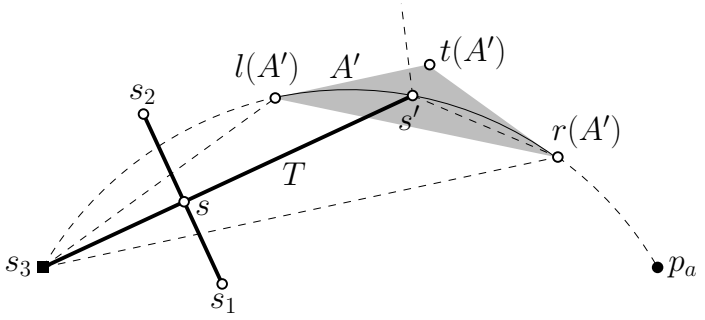


FIGURE 3.15: An example where the root s_3 is a terminal. Hence, if any of the three inequalities listed above hold for $s' = l(A')$, $r(A')$ and $t(A')$, then they must hold for all $s' \in A'$.

Case 2: Branch $\mathcal{B}$ is triple-merged.

Now, suppose that $\mathcal{B}$ is triple-merged with two other branches $\mathcal{B}_a$ and $\mathcal{B}_b$ (with roots s_a and s_b), generating a branch $\mathcal{B}'$ with Steiner segment $A' = l(A')r(A')$. If $\mathcal{B}$ is the source branch in this triple-merge, then there are again two possibilities. By a similar reasoning as the case above, if s_3 is a terminal (as shown in Figure 3.16) and there exists another terminal that satisfies at least two of the inequalities listed above for $s' = l(A')$ and $s' = r(A')$, then the branch $\mathcal{B}'$ can be pruned. Note that there is a slight difference as the Steiner curve A' is now a segment, and as a consequence, we only need to verify the inequalities at two extreme points, $l(A')$ and $r(A')$.

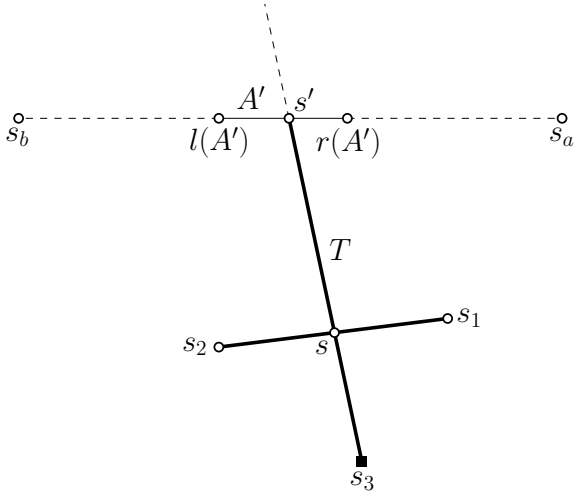


FIGURE 3.16: An example where the root s_3 is a terminal. If an equality listed above holds for $s' = l(A')$ and $s' = r(A')$, then they must hold for all $s' \in A'$.

If $\mathcal{B}$ is not the source branch in the second triple-merge, then s_3 is always fixed (an example where s_3 is not a terminal is shown in Figure 3.17). As in the previous case,

if there exists a terminal that satisfies at least two of the inequalities listed above at $s' = l(A')$ and $s' = r(A')$, then the branch $\mathcal{B}'$ can be pruned.

In Appendix A, we hypothesise that in a minimum k-Steiner tree, two degree-4 Steiner points can never be adjacent. If this hypothesis holds, then Case 2 need not be considered.

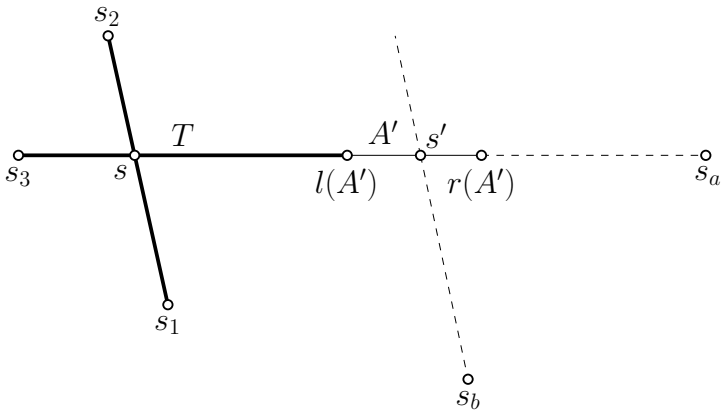


FIGURE 3.17: The root s_3 is always fixed if $\mathcal{B}$ is not the source branch.

Note that the 3-lune pruning method described in [27] can be implemented in a similar manner when a double-merge is immediately followed by a triple-merge (as shown in Figure 3.18). The method remains the same, except that we are only required to check the inequalities at the endpoints of the Steiner segment $l(A')$ and $r(A')$.

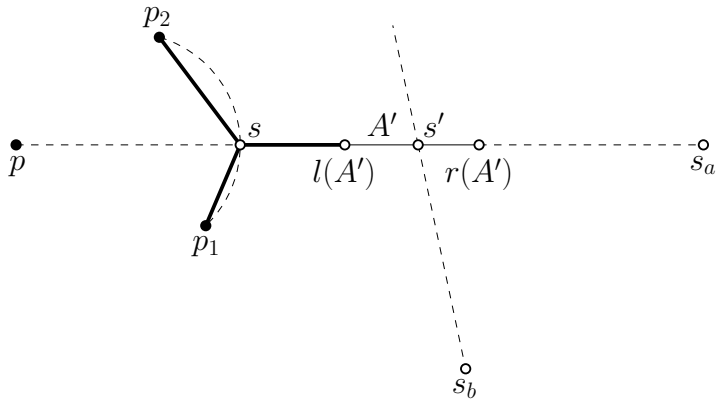


FIGURE 3.18: A illustration demonstrating how 3-lune inequalities may be used to prune a branch when a double-merge is immediately followed by a triple-merge.

We have described a method of implementing the 4-lune theorems as a dynamic pruning test. Additionally, they may be used to prune FSTs after the generation phase. The advantage of pruning during the generation phase is that it can prune branches early

on, removing the branch and any potential branches or FSTs that may arise from that branch. This can accelerate both phases of the algorithm. However, as shown above, this method requires a lot of conditions to be met in order to prune branches. For example, the inequalities must be satisfied at every extreme point of the polygon containing the Steiner curve. Furthermore, as one Steiner point does not have a fixed location, only three of the four 4-lune inequalities may be used. Alternatively, the post-generation pruning method may yield more fruitful results. This method only affects the speed of the concatenation phase. However, as FSTs are completely embedded unlike branches, all four neighbours of a degree-4 Steiner point are fixed, and we can therefore utilise all four 4-lune inequalities for a more extensive pruning.

The performance of 3 and 4-lune tests for the classical Steiner tree problem is shown in [27]. The 3 and 4-lune tests were conducted post-generation on FSTs that satisfied the lune test. The tests were described as having a "modest" impact on the number of FSTs, due to a significant overlap between the lune regions and the 3 and 4-lune regions (the 3 and 4-lune tests eliminated 8.2% of the FSTs on average for $n = 100$).

However, we provide the following example to demonstrate the potential usefulness of implementing 4-lune tests (for a degree-4 Steiner point) post-generation. In Figure 3.19, four lunes for edges as, bs, cs and ds are shown in green and the set of points satisfying at least two 4-lune inequalities for a degree-4 Steiner point and its neighbours are shown in blue. A significant portion of the blue region is not covered by the green region, and therefore post-generation pruning could be beneficial. However, we implemented various other powerful pruning tests for a degree-4 Steiner point and therefore have not included the 4-lune pruning test.

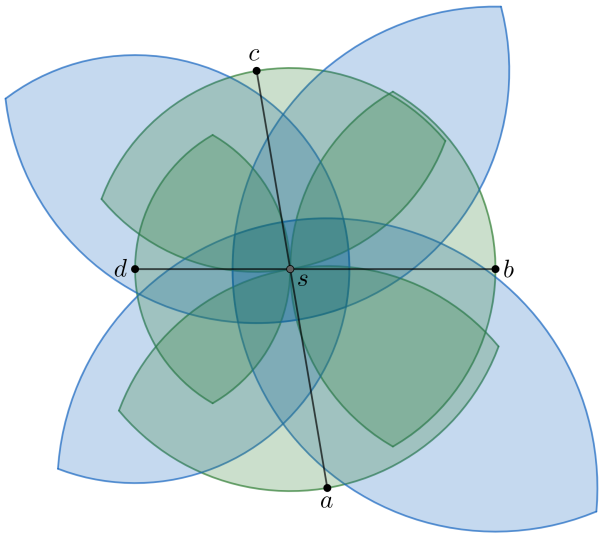


FIGURE 3.19: The region satisfying at least two 4-lune inequalities is shown in blue and the four lunes for the four edges are shown in green.

3.3.5 Trapezium Test and the Rhombus Test

In this subsection, we discuss how the Rhombus Theorem (Theorem 2.2.7) and the Trapezium Theorem (Theorem 2.2.8) are used to prune suboptimal branch trees during the generation phase. The respective pruning tests employing these theorems are referred to as the *Rhombus Test* and the *Trapezium Test*. Note that these tests take place after the fundamental projection tests.

In a preliminary version of our algorithm (as described in [1]), the Rhombus Test was used to restrict the location of the root of the source branch during a triple-merge. In this subsection, we extend the Rhombus Test in order to restrict additional neighbours of degree-4 Steiner points. This is referred to as the *extended Rhombus test*.

Note that the Trapezium Test and the extended Rhombus Test both restrict neighbours of degree-4 Steiner points to specific regions. Therefore, we combine these properties into one pruning test for simplicity and efficiency.

Suppose branches $\mathcal{B}_1$, $\mathcal{B}_2$ and $\mathcal{B}_3$ are triple-merged to create a new branch $\mathcal{B}$, where $\mathcal{B}_3$ is the source branch. The new branch $\mathcal{B}$ has a Steiner segment A, contained in s_1s_2, and its root is a degree-4 Steiner point, s. Let the Steiner curve of $\mathcal{B}_i$ be A_i, its root

s_i and pseudoterminal p_i. During the triple-merge, the location of the roots s_1 and s_2 become fixed. Since s_3 is adjacent to s, its location can be restricted according to the Rhombus and Trapezium Theorems. Furthermore, suppose that a subsequent merge occurs involving $\mathcal{B}$, resulting in a new branch $\mathcal{B}'$. Then, the root of the new branch $\mathcal{B}'$, say s', is adjacent to the degree-4 Steiner point s and hence, its location can also be restricted according to the Rhombus and Trapezium Theorems.

We refer to the pruning that occurs when $\mathcal{B}$ is generated as Phase 1 of the pruning test. Phase 2 of the pruning test occurs when we restrict the other neighbour of the degree-4 Steiner point s during a subsequent merge involving $\mathcal{B}$. Here, we describe both phases in detail, exploring the different types of merges that may arise in Phase 2.

Phase 1: Restricting the location of s_3 when $\mathcal{B}$ is generated

The root of the source branch, s_3, must be confined to the region that satisfies both the Trapezium and Rhombus Tests. This region, which is defined by the points s_1 and s_2, is denoted by T; see Figure 3.20.

If the source branch consists of a single terminal, say t_3, the triple-merge is not optimal if the source branch root, $s_3 = t_3$, does not lie inside T. Hence, if s_3 lies strictly outside T, then we disallow the current triple-merge.

Otherwise, if the source branch contains at least two terminals, we restrict the source Steiner curve to T and then project this restricted curve onto the new Steiner segment A. For example, in Figure 3.20, we first obtain the source Steiner subarc that lies inside T (denoted $\widehat{ar(A_3)}$ in the figure) and then project it onto A. We then replace A by this potentially shorter projection, further restricting the possible locations of s.

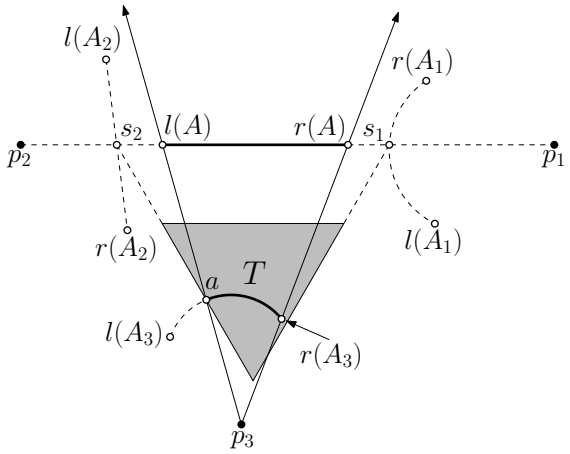


FIGURE 3.20: The Steiner point $s_3 \in \widehat{l(A_3)r(A_3)}$ must lie inside the triangle T defined by vertices s_1 and s_2. We adjust A to be the projection of the part of the source Steiner curve that lies inside T.

Phase 2: Restrictions in the subsequent merge involving $\mathcal{B}$

There are four different cases for this phase depending on the type of merge that occurs after the triple-merge.

Case 1

Suppose that a branch $\mathcal{B}_a$ with pseudoterminal p_a is double-merged with $\mathcal{B}$, resulting in a new branch $\mathcal{B}'$ with a degree-3 Steiner point s' as its root. Then, s' is adjacent to the degree-4 Steiner point s. Hence, it must be restricted to the region demarcated by the Trapezium and Rhombus Theorems, which is defined by vertices s_1 and s_2. We denote this region by T'.

This case is illustrated in Figure 3.21(a). The root of the new branch s' lies on the Steiner arc $\widehat{pp_a}$ and must also lie inside the region T' defined by the vertices s_1 and s_2. Hence, the new root s' is restricted to the subarc $\widehat{uv}$.

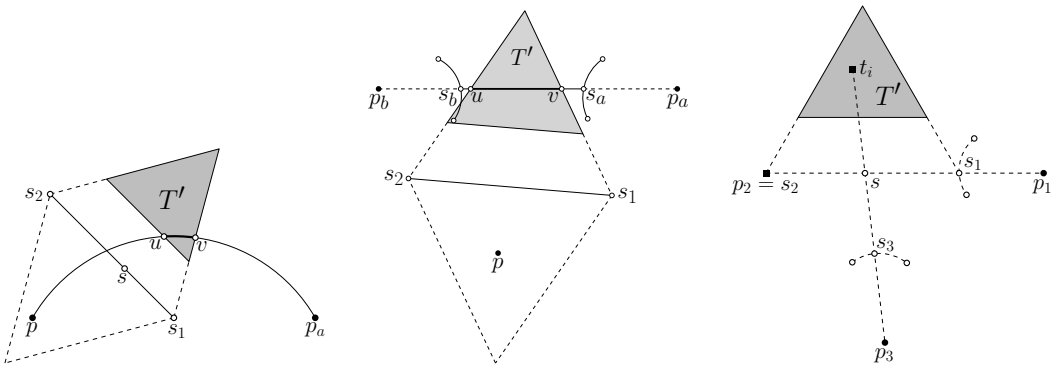
Case 2

Suppose that branches $\mathcal{B}_a$, $\mathcal{B}_b$ and $\mathcal{B}$ are triple-merged, resulting in a new branch $\mathcal{B}'$ with root s'. Let the pseudoterminal and root of branch $\mathcal{B}_a$ be p_a and s_a respectively. The pseudoterminal and root of branch $\mathcal{B}_b$ is p_b and s_b. As in the previous case, s' is adjacent to the degree-4 Steiner point s and hence must lie inside the region T' defined by roots s_1 and s_2. An example where $\mathcal{B}$ is the source branch is shown in Figure 3.21(b).

The new root s' must lie on the Steiner segment $s_a s_b$ and also lie inside the region T'. Therefore, s' is restricted to the subsegment uv.

Case 3

Suppose that branch $\mathcal{B}$ undergoes a termination merge with terminal t_i to generate an FST F. The terminal t_i is adjacent to the degree-4 Steiner point s in F. Hence, t_i must lie inside the region T' established by the Trapezium and Rhombus Tests, otherwise F cannot be optimal. Figure 3.21(c) shows an example where $t_i \in T'$. If the terminal $t_i \notin T'$, we disallow the termination merge and do not generate the FST.



(a) The new degree-3 Steiner point s' is restricted to the sub-arc $\widehat{uv}$.

(b) The new degree-4 Steiner point s' is restricted to the subsegment uv.

(c) The terminal t_i must lie inside T' for the termination merge to pass the Trapezium and Rhombus pruning tests.

FIGURE 3.21: Examples of how the Trapezium and Rhombus Theorems are applied as pruning tests after a triple-merge has occurred.

Case 4

Suppose that branches $\mathcal{B}_1$ and $\mathcal{B}_2$ are double-merged, where their respective roots s_1 and s_2 were each produced in a triple-merge. Then, the root of the newly generated branch is adjacent to both degree-4 Steiner points s_1 and s_2, and must therefore lie inside T'_1, the region defined by the Trapezium and Rhombus Theorems for s_1 and T'_2, the region for s_2.

Similarly, if branches $\mathcal{B}_1, \mathcal{B}_2$ and $\mathcal{B}_3$ are triple-merged, where at least two of the roots s_1, s_2, s_3 were each produced in a triple-merge, then the new root s must lie in the intersection of the relevant triangles. For example, Figure 3.22 shows an example where $\mathcal{B}_1$, $\mathcal{B}_2$ and $\mathcal{B}_3$ are triple-merged with $\mathcal{B}_3$ as the source branch. In this example, the

roots s_1 and s_3 were each produced in a triple-merge. The root of the new branch, s, is adjacent to both s_1 and s_3 and hence must lie inside both T_1' and T_3', the regions established by the Trapezium and Rhombus Theorems of the previous triple-merges of branches $\mathcal{B}_1$ and $\mathcal{B}_3$ respectively. Hence, s' must lie on the segment vw.

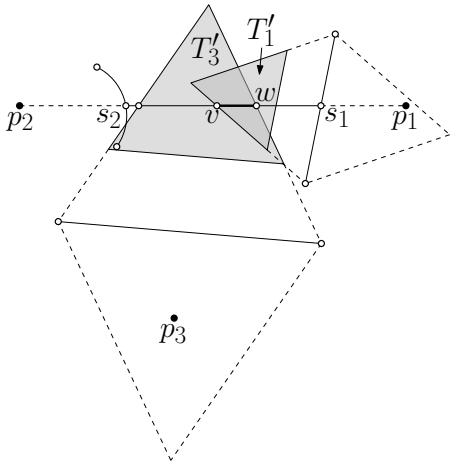


FIGURE 3.22: The new degree-4 Steiner point s' must lie on vw to satisfy the Trapezium and Rhombus Tests.

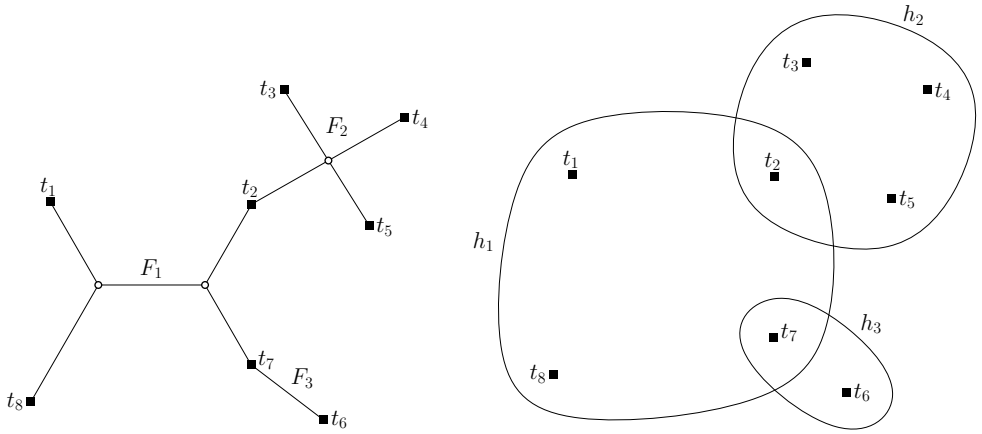
3.4 The concatenation algorithm

In the generation phase, a set of feasible FSTs, $\mathcal{F}$, is obtained. In the concatenation phase, the aim is to obtain a subset $\mathcal{F}^*$ of $\mathcal{F}$, so that $\mathcal{F}^*$ constitutes the set of full components of a minimum length connected network spanning the given terminal set N and containing at most k Steiner points. This is known as the *concatenation problem*.

There are many ways to solve the concatenation problem, including backtrack search, dynamic programming and solving an integer linear programming (ILP) problem [13]. In this section, we provide a cut-formulation ILP model for the concatenation phase, and implement a classical method related to branch-and-cut for finding an optimal solution to the ILP. In order to do this, we transform the concatenation problem into a *minimum connected spanning sub-hypergraph problem*, where each FST is mapped to a hyperedge in a hypergraph. It is this problem which is then modelled and solved as an ILP.

We first establish some key definitions concerning hypergraphs. A *weighted hypergraph* $G = (V, H, w)$ consists of a vertex set V; a set of *hyperedges* H, where each $h \in H$ is a subset of V of cardinality at least two; and a weight function $w : H \to \mathbb{R}$. A *chain*

in a hypergraph is defined to be an alternating sequence of vertices and hyperedges $v_0, h_1, v_1, h_2, v_2, \ldots, h_l, v_l$ such that all vertices and hyperedges are distinct and $v_{i-1}, v_i \in h_i$ for $i = 1, 2, \ldots, l$. For any $H' \subseteq H$, let $G[H']$ be the sub-hypergraph of G induced by H'. A *connected spanning sub-hypergraph* of G is a hypergraph $G[H']$ for some $H' \subseteq H$ such that the vertex-set of $G[H']$ is V and such that there is at least one chain between any two vertices in the hypergraph.



(a) Three FSTs F_1, F_2, F_3.

(b) Three hyperedges $h_1 = \{t_1, t_2, t_7, t_8\}$, $h_2 = \{t_2, t_3, t_4, t_5\}$ and $h_3 = \{t_6, t_7\}$ corresponding to FSTs F_1, F_2 and F_3 respectively.

FIGURE 3.23

We now show how to transform the concatenation problem to a restricted version of finding minimum-weight connected spanning sub-hypergraphs. Let $G = (V, H, w)$ be the following weighted hypergraph. Let $V := N$. Let $H := \{h_i : 1 \leq i \leq |\mathcal{F}|\}$ where $h_i := V(F_i) \cap N$ and where $V(F_i)$ is the set of vertices in F_i (for example, FST F_1 in Figure 3.23(a) is transformed into a hyperedge $h_1 = \{t_1, t_2, t_7, t_8\}$ in Figure 3.23(b)). Let w be the weight function which assigns a weight to each hyperedge, where the weight of hyperedge h_i is defined to be the length of its associated FST F_i, i.e., $w(h_i) = |F_i|$. For any set $H' \subseteq H$ we define $w(H') := \sum_{h_i \in H'} w(h_i)$. To model the bound k on the number of Steiner points, we allocate a *penalty* k_i for each hyperedge h_i. Under the transformation, the penalty k_i is defined to be the number of Steiner points in the associated FST F_i.

We begin by defining the network union of a set of geometric networks (all embedded in the same space).

Definition 3.4.1. Let $\{G_i\}$ be a set of geometric networks where, for each i, G_i has vertex set V_i and edge set E_i. Then, the *network union* of $\{G_i\}$ is defined to be the network $\bigcup_i G_i := (\bigcup_i V_i, \bigcup_i E_i)$.

For any given weighted hypergraph $G = (V, H, w)$ and a non-negative integer k, we aim to solve the following.

Definition 3.4.2 (Problem P*)**.** The *k-restricted minimum-weight connected spanning sub-hypergraph problem* is the problem of finding a set $H^* \subseteq H$ (if such a set exists) such that: $G[H^*]$ is a connected spanning sub-hypergraph of G; $\sum_{h_i \in H^*} k_i \leq k$; and H^* has minimum total weight.

In the context of the concatenation problem, such an H^* always exists, as $\mathcal{F}$ contains a set of FSTs whose network union is a minimum k-Steiner tree. Henceforth, we refer to the above problem as P*.

Theorem 3.4.3. Let H' be a subset of H. Let $\mathcal{F}' := \{F_i \in \mathcal{F} : V(F_i) \cap N \in H'\}$. Then H' is an optimal solution to P^* if and only if the network

$$F' := \bigcup_{F_i \in \mathcal{F}'} F_i$$

(under network union) is a solution to the minimum k-Steiner tree problem.

Proof. Firstly, we show that $\mathcal{F}'$ is feasible for the concatenation problem if and only if H' is feasible for P*. Since each FST F_i contains k_i Steiner points, the total number of Steiner points in $\mathcal{F}'$ is $\sum_{F_i \in \mathcal{F}'} k_i$. Since the penalty of the associated hyperedge h_i is also k_i, it follows that $\sum_{h_i \in H'} k_i = \sum_{F_i \in \mathcal{F}'} k_i$. Hence,

$$\sum_{F_i \in \mathcal{F}'} k_i \leq k \iff \sum_{h_i \in H'} k_i \leq k.$$

Clearly $G[H']$ and F' span the same terminal-set. Note that F' is a graph, and therefore F' is connected if and only if there exists a path between every pair of distinct vertices of F'. Equivalently, since F' is a union of FSTs, F' is connected if and only if, for every pair x, y of distinct terminals of F', there exists an alternating sequence of FSTs and terminals of F', say, $t_{i_0}, F_{i_1}, t_{i_1}, F_{i_2}, t_{i_2}, ..., F_{i_p}, t_{i_p}$, where $x = t_{i_0}$, $y = t_{i_p}$, and such that $t_{i_0} \in F_{i_1}$,

$t_{i_p} \in F_{i_p}$ and t_{i_j} is contained in both F_{i_j} and in $F_{i_{j+1}}$ for all $j \in \{1, ..., p-1\}$. Suppose first that F' is connected. Then, since $h_i := V(F_i) \cap N$, there exists a corresponding chain $t_{i_0}, h_{i_1}, t_{i_1}, h_{i_2}, t_{i_2}, ..., h_{i_p}, t_{i_p}$ between x and y in $G[H']$. Hence $G[H']$ is connected. Similarly, if $G[H']$ is connected then $\mathcal{F}'$ is also connected. Thus, H' is feasible for P* if and only if F' is a connected graph spanning N and contains at most k Steiner points (i.e., F' is feasible for the concatenation problem).

Since the weight of hyperedge h_i is given by $w(h_i) = |F_i|$, we have

$$\sum_{h_i \in H'} w(h_i) \; = \sum_{F_i \in \mathcal{F}'} |F_i|.$$

Hence, the length of any solution to the concatenation problem will be equal to the weight of the corresponding solution to P*. Therefore, if a solution to the concatenation problem is optimal, then the corresponding solution to P* will be optimal. □

The problem P* can be modelled as an ILP problem. There are numerous ways of doing this. For example, Warme generalised the usage of subtour elimination constraints for the minimum spanning tree problem for graphs to hypergraphs [30].

In contrast, we model our ILP using cut-constraints. Essentially, we generalise the classical cut-formulation for connectivity in graphs, i.e., we ensure that at least one (hyper-)edge crosses every partition of the vertex-set. The primary advantage of this method, over the Warme method, for our purposes is that it yields a simpler separation algorithm (discussed below).

Note that the Warme formulation makes an assumption that there is a unique chain between every pair of terminals in an optimal solution. However, our ILP does not make this assumption, and therefore the resulting graph may potentially contain cycles. However, as $\mathcal{F}$ contains FSTs whose network union is a minimum k-Steiner tree, an optimal solution to P* (in the concatenation problem) will always have a unique chain between every pair of terminals. We do not therefore need to enforce this assumption.

To begin, we define the concept of a *hyperedge cut*.

Definition 3.4.4. For any $S \subseteq V$, the *hyperedge cut* of G, associated with S, is denoted $\delta(S) \subseteq H$ and is defined to be the set of all hyperedges in H which contain at least one vertex in S and at least one vertex in $V - S$.

An example of a hyperedge cut is shown in Figure 3.24.

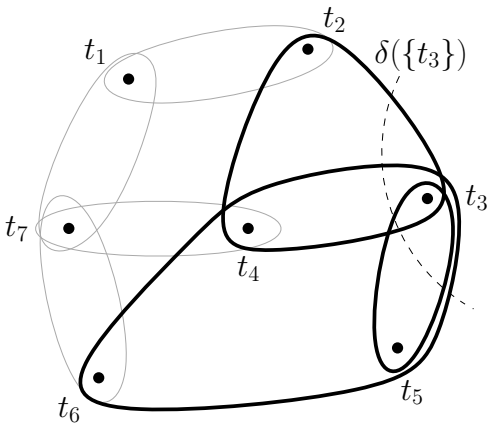


FIGURE 3.24: If $S = \{t_3\}$, then $\delta(S) = \{\{t_2, t_3, t_4\}, \{t_3, t_5\}, \{t_3, t_4, t_5, t_6\}\}$.

Associate a binary variable x_i with each hyperedge $h_i \in H$, such that $x_i = 1$ if and only if the associated hyperedge h_i is present in a solution. Our ILP formulation is as follows:

$$\text{minimise} \sum_{h_i \in H} w(h_i) x_i$$

$$\text{subject to} \sum_{h_i \in \delta(S)} x_i \geq 1, \qquad \emptyset \neq S \subset V \tag{3.1}$$

$$\sum_i k_i x_i \leq k \tag{3.2}$$

$$x_i \in \{0, 1\} \; \forall \; i.$$

The constraint-set in (3.1) is referred to as the set of *cut-constraints*, and consists of an exponential number of constraints. These constraints ensure that the hyperedge cut of a feasible solution is non-empty for every non-empty proper subset $S \subset V$. Constraint (3.2) ensures that any feasible solution is k-restricted.

As the cut-formulation ILP model for hypergraphs is a novel approach to handle the concatenation phase, we first prove that all feasible solutions satisfying the constraints must be k-restricted connected spanning sub-hypergraphs.

Theorem 3.4.5. A set $H' \subseteq H$ induces a connected spanning sub-hypergraph of $G = (V, H, w)$ with a penalty of at most k if and only if Constraints (3.1) and (3.2) are satisfied.

Proof. ($\Rightarrow$) Suppose first that H' induces a connected spanning sub-hypergraph $G[H']$ of $G = (V, H, w)$ with a penalty of at most k. Let $x_i = 1$ if and only if $h_i \in H'$. Clearly, then, Constraint (3.2) is satisfied. To show that Constraint (3.1) is satisfied we must show that H' contains at least one hyperedge in the hyperedge cut of every non-empty proper subset $S \subset V$.

Let $S \subset V$ be some non-empty proper subset of the vertex set. Let u be a vertex in S and let v be a vertex in $V - S$. Then by the definition of a connected spanning sub-hypergraph, there exists at least one chain $C = u, h_1, v_1, h_2, v_2, ..., h_p, v$ between u and v (where each of the hyperedges belongs to H'). Assume that none of the hyperedges in the chain C are in the hyperedge cut of S. Then, $v_1 \in S$ since $h_1 \notin \delta(S)$. Similarly, v_2, v_3, ..., $v_{p-1} \in S$. Then, $h_p \in \delta(S)$ as $v_{p-1} \in S$ and $v \in V - S$. This is a contradiction. Hence, there must be at least one hyperedge from H' in the hyperedge cut $\delta(S)$ for any non-empty proper subset S of V, implying that Constraint (3.1) must also be satisfied.

($\Leftarrow$) Let $x' \in \{0,1\}^{|H|}$ be feasible for Constraints (3.1) and (3.2). Then, we must show that the corresponding set of hyperedges H' induces a connected spanning sub-hypergraph $G[H']$ with $\sum_{h_i \in H'} k_i \leq k$. Clearly, $\sum_{h_i \in H'} k_i \leq k$ because x' must satisfy Constraint (3.2). To show that $G[H']$ is connected and spanning, we assume otherwise, that there exists some $u, v \in V$ such that there exists no chain between u and v. Let S be a subset of V which consists of all the terminals t_i such that there exists at least one chain between t_i and u. By definition, $u \in S$ and $v \notin S$. Due to Constraint (3.1), there must be at least one hyperedge in H', say h_j, such that $h_j \in \delta(S)$. By the definition of a hyperedge cut, there must be at least one terminal $t^* \in h_j$ such that $t^* \notin S$, which means that there exists a chain between u and t^*, leading to a contradiction. Hence, there must be at least one chain between any two terminals in $G[H']$, so it must be connected and must span V. □

We now discuss our solution method for our ILP. Since there are $2^n - 2$ non-empty

strict subsets of N, the number of cut-constraints is exponential. Including every cut-constraint at once can cause the ILP solver (in our test-instances, Gurobi) to run inefficiently. Therefore only a small number of cut-constraints are added at a time, whereafter a solution is generated and a separation algorithm is used to determine which constraints are violated. This is a well-known approach in ILP solving (see, for instance, [31]).

More specifically, the model is initially solved without any cut-constraints. Then, once the initial solution has been obtained, the number of components in the solution is calculated. If the solution has exactly one component, then the solution is optimal since all cut-constraints are satisfied by such a solution. Otherwise, the separation algorithm adds one constraint per component in the current solution (unless there are exactly two components, in which case there is exactly one constraint added) to ensure that there is at least one hyperedge in the hyperedge cut of each component. The new ILP model is then re-solved with the added cut-constraints and a potentially new solution is obtained. This process is repeated until there is exactly one component in the solution. Practically, this process is implemented in the form of *lazy constraints* in Gurobi.

Next, we present the pseudocode for our ILP algorithm. We then establish the correctness of the algorithm.

Algorithm 4: Algorithm for Solving the ILP for P*

Input: A weighted hypergraph $G = (V, H, w)$ and a non-negative integer k.

Output: An optimal solution x^* to the ILP for P*.

```
1  Θ := ∅                    // Θ is the current set of cut-constraints (a subset of
     Constraint (3.1))
2  Solve ILP(∅)              // Solving the ILP where the constraint-set consists of Θ and
     Constraint (3.2)
3  while p ≥ 2 do
4      if p ≥ 3 then
5          for j ∈ {1, ..., p} do
6              Add the constraint Σ_{h_i ∈ δ(C_j)} x_i ≥ 1 to Θ.
7      else
8          Add the constraint Σ_{h_i ∈ δ(C_1)} x_i ≥ 1 to Θ.
9      (x*, C_1, C_2, ..., C_p) := Solve ILP(Θ), where x* is the current solution and C_1,
         C_2, ..., C_p are the vertex sets of the components in x*.
10 return x*
```

Note that there are a finite number of non-empty proper subsets of V. In every iteration, at least one new cut-constraint is added to the ILP. Hence, there are a finite number of iterations in the separation algorithm and therefore Algorithm 4 terminates. Assuming that $\mathcal{F}$ contains a set of FSTs whose network union is a minimum k-Steiner tree, Algorithm 4 terminates successfully, with a final solution x' that satisfies Constraints (3.1) and (3.2). By Theorem 3.4.5, the hypergraph induced by the associated set of hyperedges, $G[H']$, is a minimum-weight connected spanning sub-hypergraph. Hence, by Theorem 3.4.3, the network union of the corresponding FSTs, F', forms a minimum k-Steiner tree.

3.5 Experimental testing

The discussion in this section focuses mainly on the improvements to the initial version of the k-Steiner tree algorithm from [1].

We present experimental results to demonstrate the overall improvement in algorithm performance resulting from our new pruning tests and from certain core improvements

in the algorithm design. We also provide experimental results on the efficiency of the concatenation phase.

3.5.1 Core improvements to the generation phase

Besides our new pruning tests, a number of core improvements were made to the generation phase of the previous version of our algorithm. Firstly, our new generation algorithm utilises terminal indices during termination merges to decrease the number of instances of double-counting, resulting in fewer branches and FSTs. A single FST may be generated in numerous ways by rearranging the order of merges. In order to prevent generating the same FST multiple times, we deem a termination merge between a branch $\mathcal{B}$ and terminal t_i to be feasible only if the terminal with the highest index in $\mathcal{B}$ has an index lower than i.

Furthermore, the new generation algorithm stores branches and FSTs in a more compact format to ameliorate potential memory issues that arise when larger instances are tested. Finally, several improvements were made to the projection test: Propositions 3.3.1 and 3.3.2 allow us to quickly determine when a merge is infeasible, while Theorem 3.3.3 allows additional trimming of the Steiner segment in certain cases. Overall, these allow for quicker and more extensive pruning.

Here, we provide some experimental results to demonstrate the improvement in running time of the current version of our generation phase over the previous version.

All experiments were conducted on a Windows 10 operating system with 4 cores and 16GB RAM. As is observed in Table 3.1, there was a significant decrease in the time duration of the generation phase. The above-mentioned core improvements and improved pruning tests decreased the runtime of the generation algorithm significantly. For example, for $n = 25$ and $k = 3$, the generation algorithm runtime decreased by 98.11%, from 2541.3 seconds to 47.96 seconds. The algorithm speed was improved by a further 28.78% with the inclusion of the Trapezium Test and the extended Rhombus Test.

Number of terminals	5	10	15	20	25
Previous algorithm	0.05236	3.6073	67.706	551.63	2541.3
Current algorithm without extra tests*	0.01916	0.96867	5.8228	20.050	47.960
Current algorithm with extra tests*	0.01770	0.87373	4.7265	15.318	34.156

TABLE 3.1: The average time duration (in seconds) of the generation phase for the previous algorithm and the current algorithm (with and without the Trapezium Test and the extended Rhombus Test) over 20 randomly generated instances for $k = 3$. Here, extra tests* refer to the Trapezium Test and the extended Rhombus Test.

3.5.2 The effect of the new pruning tests

The new pruning tests present in the improved version of the algorithm are the Trapezium Test and the extended Rhombus Test. Here we compare the performance of our algorithm with and without these tests. All major pruning tests from the previous version of the algorithm have been incorporated into the current algorithm, including the lune test and the bottleneck test.

The experiments were conducted on terminal set sizes of $n = 5, 10, 15, ...$ for $k = 3, 4, 5$, where the maximum value of n depends on k. For every value of n and k, the algorithm solved 20 different random instances, where terminal locations were uniformly generated in a $[0, 1] \times [0, 1]$ square. Figure 3.25 illustrates an example of a minimum 2-Steiner tree on 9 terminals that are randomly generated as described.

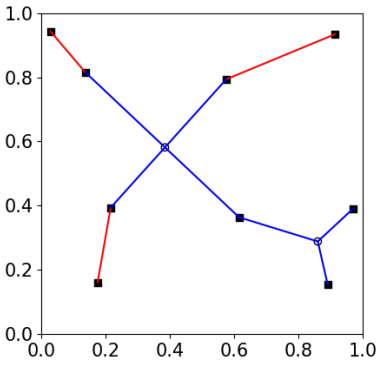


FIGURE 3.25: A minimum 2-Steiner tree.

In Figures 3.26, 3.27 and 3.28, we observe that the extended Rhombus Test together with the new Trapezium Test significantly decrease the completion time of the algorithm.

These improvements in time are more pronounced for larger instances, with higher values of n and k. For example, we note that including these tests results in a 35.66% decrease in total time taken for $k = 3, n = 40$ while it results in a 52.36% time decrease for $k = 4, n = 40$ and 64.21% for $k = 5, n = 40$.

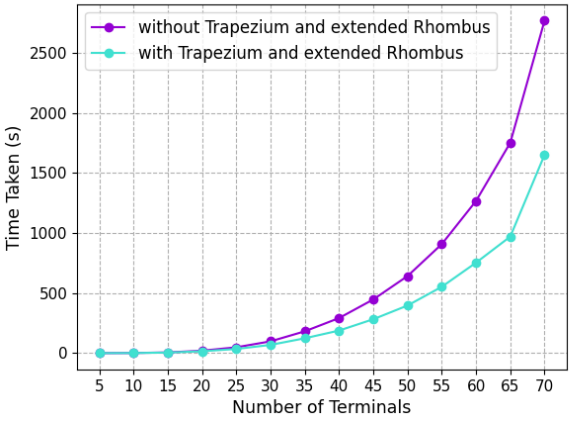


FIGURE 3.26: Improvement in time by including the Trapezium Test and the extended Rhombus Test for $k = 3$.

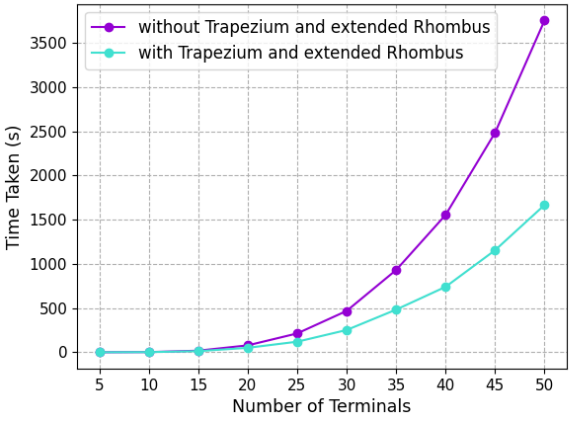


FIGURE 3.27: Improvement in time by including the Trapezium Test and the extended Rhombus Test for $k = 4$.

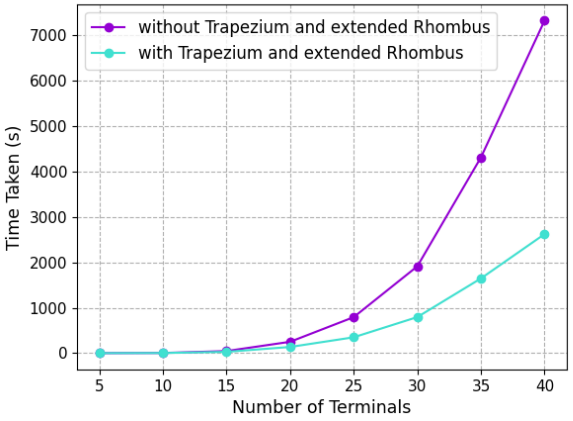


FIGURE 3.28: Improvement in time by including the Trapezium Test and the extended Rhombus Test for $k = 5$.

Both the Trapezium and extended Rhombus Tests are implemented during and after a triple-merge. Figures 3.29, 3.30 and 3.31 reflect this, as the number of FSTs that contain a degree-4 Steiner point significantly decreases when these tests are included. Furthermore, the effectiveness of the tests in reducing the number of FSTs with a degree-4 Steiner point slightly increases as k increases. For example, for $k = 3, n = 40$, the extra pruning tests results in a 92.41% reduction in the number of FSTs with degree-4 Steiner points while it yields a 94.40% decrease for $k = 4, n = 40$. Finally, for $k = 5, n = 40$, the number of FSTs decreases by 96.05%. Note that these results indicate the number of FSTs pruned during generation only, i.e., prior to post-processing.

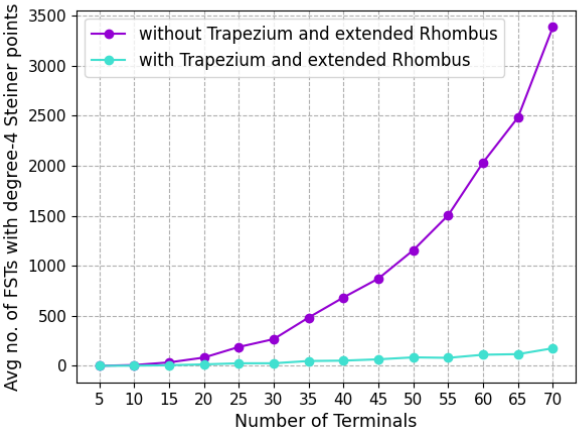


FIGURE 3.29: The decrease in the number of FSTs with degree-4 Steiner points by using the Trapezium and extended Rhombus Tests for $k = 3$.

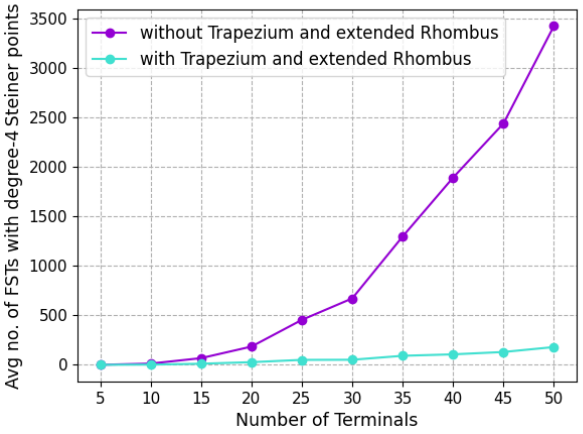


FIGURE 3.30: The decrease in the number of FSTs with degree-4 Steiner points by using the Trapezium and extended Rhombus Tests for $k = 4$.

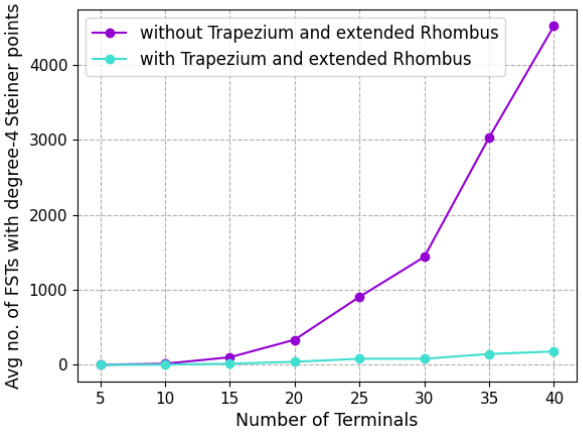


FIGURE 3.31: The decrease in the number of FSTs with degree-4 Steiner points by using the Trapezium and extended Rhombus Tests for $k = 5$.

3.5.3 The concatenation phase

In this subsection, we test the efficiency of our novel concatenation algorithm. A total of 20 random instances, i.e., terminal sets, were generated for each value of k and n. For each instance, Figure 3.32 plots the number of FSTs that remained after generation and post-processing (including the extra tests) in the new algorithm against the completion time of the concatenation phase.

In the figure, we see that the time taken for the concatenation phase increases rapidly as the number of FSTs increases. This is expected, as the number of binary variables in

the ILP of the concatenation phase corresponds to the number of FSTs created during the generation phase. Furthermore, the number of cut-constraints in the ILP is given by $2^n - 2$, where n is the number of terminals. This demonstrates the value of including strong pruning tests to reduce the total number of FSTs. Lastly, we note that the concatenation time appears to have higher variance as the size of the instances get larger.

Finally, we observed that for instances of the size that we analysed in our experiments, concatenation tended to be significantly faster than generation. For example, when $n = 40$ and $k = 5$, the concatenation algorithm accounted for an average of 1.24% of the total runtime. We note, however, that neither algorithm's code has been optimised for performance. The concatenation time increases much more rapidly than the generation time, and therefore we expect concatenation time to eventually dominate for larger instances.

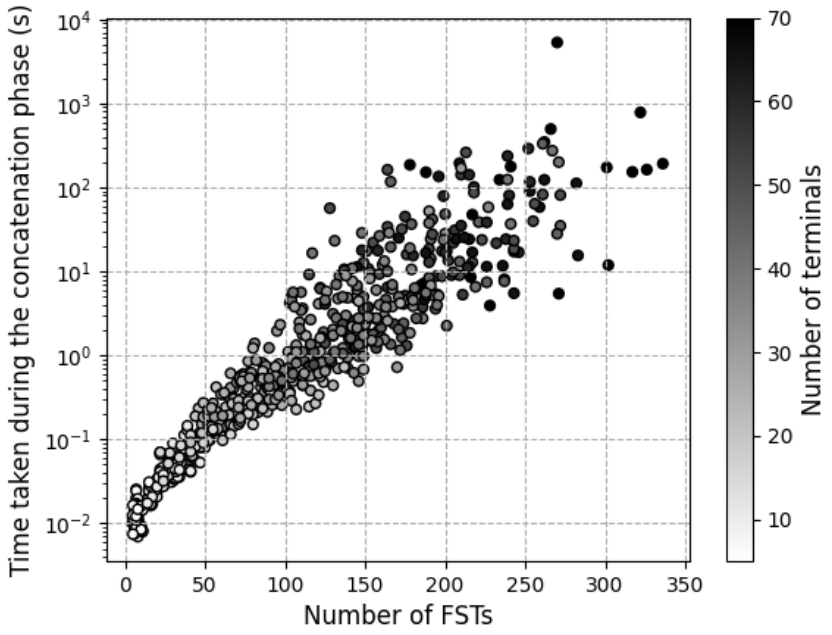


FIGURE 3.32: The relationship between the number of FSTs and the corresponding time duration of the concatenation phase (for $k = 3, 4, 5$); note, log-scale is employed.

Chapter 4

The Minimum Bottleneck k-Steiner Network Problem

As stated in Section 1.1, the content for Chapter 4 will be submitted as a third paper.

4.1 Background and Preliminaries

In this section, we provide basic definitions, theorems and results required for this chapter.

Definition 4.1. Let $N = \{t_1, t_2, ..., t_n\}$ be a set of $n \geq 2$ distinct points in the Euclidean plane and k be a non-negative integer. A *k-Steiner network* on N is defined to be a connected (straight-line) geometric network $T = (V(T), E(T))$ in $\mathbb{R}^2$, where:

1. $N \subseteq V(T)$, and
2. $|V(T)| \leq n + k$.

The points of N are called *terminals* and the points in $T \setminus N$ are called *Steiner points*. A longest edge of T is called a *bottleneck edge* of T.

The *minimum Euclidean bottleneck k-Steiner network problem* may be stated as follows: find a k-Steiner network T^* on N such that

$$|T^*| := \max_{e \in E(T^*)} |e|$$

is minimum, where $|e|$ denotes the Euclidean length of e. We refer to $|T^*|$ as the *cost* of T^* and T^* is called a *minimum bottleneck k-Steiner network.*

In general, there are an infinite number of solutions to the above problem for a given set of terminals and integer k. Hence, we begin by defining solutions that satisfy certain desirable properties.

Definition 4.2. Let T be a k-Steiner network on N. The *edge spectrum* of T is the non-increasing list of all its edge lengths.

The set of k-Steiner networks on N can be ordered with respect to their edge spectra as follows. Let T_a and T_b be k-Steiner networks on N and let $(l_a^1, l_a^2, ..., l_a^p)$ and $(l_b^1, l_b^2, ..., l_b^q)$ be their edge spectra respectively. Let r be the number such that $l_a^1 = l_b^1$, $l_a^2 = l_b^2$, ..., $l_a^r = l_b^r$ and either $l_a^{r+1} \neq l_b^{r+1}$ or r is equal to either p or q. If $l_a^1 \neq l_b^1$, then r is zero. Then, we say that T_a is *lexicographically smaller* than T_b if either:

1. $l_a^{r+1} < l_b^{r+1}$, or
2. $p = r$ and $q > r$.

Definition 4.3. Given a set of terminals N and a non-negative integer k, a *bottleneck k-Steiner tree* (k-BST) is a k-Steiner network T on N that is *lexicographically minimal* with respect to its edge spectrum. In other words, there exists no other k-Steiner network on N with an edge spectrum that is lexicographically smaller than the edge spectrum of T.

We note that a k-BST must be a tree, as removing any edges that do not disconnect the network results in a lexicographically smaller edge spectrum. Furthermore, it must contain exactly k Steiner points as an extra Steiner point can always produce a lexicographically smaller edge spectrum. For example, let $(l^1, l^2, ..., l^p)$ be the edge spectrum of a k-Steiner network T with fewer than k Steiner points. Then, adding a Steiner point at the midpoint of any edge would result in a lexicographically smaller edge spectrum. Lastly, a k-BST is a minimum bottleneck k-Steiner network. For assume that there exists a k-BST T that is not a minimum bottleneck k-Steiner network, i.e., that there exists a k-Steiner network with a strictly shorter bottleneck edge. This is a contradiction as this network is lexicographically smaller than T.

Next we show that there exists at least one k-BST for any given set of terminals N and any non-negative integer k. First, let $\mathcal{T}_1$ be the set of all k-Steiner networks that minimise the longest edge and let l_1 be this length. Let $\mathcal{T}_2 \subseteq \mathcal{T}_1$ be the set of all trees in $\mathcal{T}_1$ that minimise the second longest edge and let l_2 be this length. Note that we may have $l_1 = l_2$. We continue this process for p steps, where p is the smallest index such that every network in $\mathcal{T}_p$ has the same edge spectrum.

By construction, every network in $\mathcal{T}_p$ is lexicographically minimal, and therefore a k-BST. By the above discussion, every network in $\mathcal{T}_p$ is a tree and $p = n + k - 1$.

We now prove two additional properties of k-BSTs. In [18], Bae et al. purposefully restricted their attention to Steiner trees with useful properties. This includes the property described in Proposition 4.4 below, and a weaker version of Proposition 4.5. However, with our definition of lexicographic minimality, these properties follow naturally.

Proposition 4.4. *Every Steiner point of a k-BST must lie at the centre of the smallest disc containing its neighbours.*

Proof. Let T be a k-BST on the set of terminals N. Now, suppose that there exists a Steiner point $s \in V(T)$, with neighbours $N(s)$, that does not lie at the centre of the smallest disc, D, containing its neighbours. Let c and r be the centre and radius of D. We relocate s to c, maintaining its incident edges (and hence its topology), and denote the new Steiner point by s'. In this process we therefore generate a new k-Steiner network, say T'. We show that T' is a lexicographically smaller k-Steiner network than T.

Since $N(s') \in D$, the longest edge incident to s' must have a length of at most r. In fact, its length must be exactly r because otherwise, $|s'x| < r \; \forall \; x \in N(s')$ which implies that D is not the smallest disc containing $N(s') = N(s)$.

We now show that the longest edge incident to s in T is strictly greater than r, which implies that T' is a lexicographically smaller k-Steiner network and hence we get a contradiction.

Assume otherwise, i.e., that every edge incident to s has a length of at most r. This implies that every neighbour of s lies inside a disc of radius r centred at s. Note that a smaller disc centred at s cannot contain every neighbour of s as this contradicts the

minimality of D. Hence, the disc with radius r centred at s is a smallest disc containing $N(s)$. However, this is a contradiction as the smallest disc containing a set of points in the plane is unique [32]. □

Proposition 4.5. *Let T be a k-BST. Then, T is an MST on its set of vertices, $V(T)$.*

Proof. Let T be a k-BST. Assume that T is not an MST on the set of its vertices $V(T)$. Then, by the matroid property of MSTs, there exists some edge in T that can be replaced by a strictly shorter edge, whilst maintaining connectivity. However, this replacement results in a strictly lexicographically smaller edge spectrum, which is a contradiction. □

Next we define a general type of k-Steiner network which we call *canonical k-Steiner trees*. These are defined as k-Steiner networks that are trees and every Steiner point lies at the centre of the smallest disc containing its neighbours. Hence, k-BSTs are canonical k-Steiner trees. An example of a canonical 5-Steiner tree is shown below in Figure 4.1. The purpose of defining this class of network is to have a defined set of feasible solutions, which are produced when generating potential solutions to the minimum Euclidean bottleneck k-Steiner network problem.

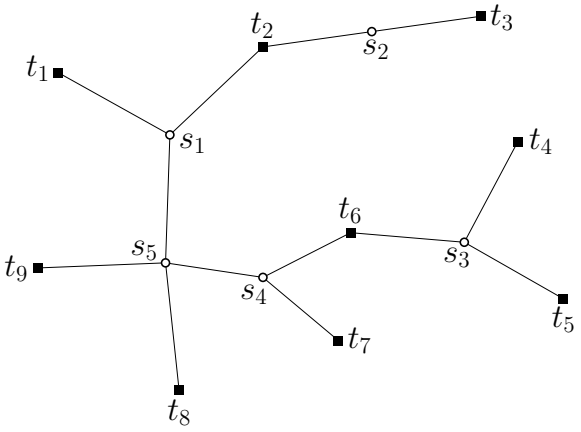


FIGURE 4.1: An example of a canonical 5-Steiner tree.

Every canonical k-Steiner tree can be decomposed into so called *cluster subtrees*, based on the work of Bae et al. [16], [18]. We need a number of basic concepts from their papers.

Firstly note that, in a canonical k-Steiner tree, for any Steiner point s, the boundary of a smallest disc C containing the neighbours of s is determined either by two diametrically

opposite points, or by three points on the boundary of C. If C is defined by three points then the triangle formed by these 3 points encloses s. We call this set of two or three points the *determinators* of s. If there are more than three points on C then we break ties and arbitrarily pick any three points on C that form a triangle enclosing s and assign these points as the determinators of s.

Let $\mathcal{T}$ be a canonical k-Steiner tree with Steiner point set $S = \{s_1, ..., s_p\}$ and let $\mathcal{D} = \{D_1, ..., D_p\}$ be such that D_i is the set of determinators of s_i for each $1 \leq i \leq p$. As in [18] we first construct a directed graph G on S, where G has an arc from s_i to s_j if and only if $s_j \in D_i$. We now define an equivalence relation $\equiv$ as follows: for every $s_i, s_j \in S$ we have $s_i \equiv s_j$ if and only if there exists a directed path in G from s_i to s_j and a directed path in G from s_j to s_i.

Then, the equivalence class of Steiner point s_i, denoted $[s_i]$, is defined to be the set of Steiner points in S that are equivalent to s_i, i.e., $[s_i] := \{s \in S | s \equiv s_i\}$. Finally, let $[S] := \{[s_i] \,|\, s_i \in S\}$. We refer to the elements of $[S]$ as *clusters*.

As demonstrated in [18], there is a natural partial ordering $\preceq$ on $[S]$. Let r_i be the distance between s_i and its determinators. The relation $\preceq$ has the property that if $s_i \in [s_j]$ then $r_i = r_j$, and if $[s_j] \preceq [s_i]$ then $r_j \leq r_i$. If $[s_i]$ is a maximal element of S with respect to the partial order, then $[s_i]$ is referred to as a *primary cluster*. More detail can be found in [18].

Using the partial order defined above, we now present a method to partition the canonical k-Steiner tree $\mathcal{T}$ into subtrees that inherit the partial order.

Definition 4.6. For any $s_i \in S$, let T_i be the subtree of $\mathcal{T}$ induced by the node set $V_i := \bigcup\{D_j : s_j \in [s_i]\} \cup [s_i]$. Tree T_i is referred to as a *cluster subtree* of $\mathcal{T}$. If $[s_i]$ is a primary cluster then T_i is called a *primary* cluster subtree.

Note that every edge in a cluster subtree is of the same length. It follows from [18] that the leaves of a primary cluster subtree are all terminals.

Definition 4.7. An *undetermined edge* of $\mathcal{T}$ is any edge vw such that v is not a determinator of w and w is not a determinator of v.

Note that undetermined edges are not part of any cluster subtree. Consequently, the edge-induced forest of $\mathcal{T} - E'$, where E' is the set of undetermined edges of $\mathcal{T}$, is a graph union of cluster subtrees.

We now provide an example. Figure 4.2 shows a directed graph where the underlying undirected graph is a canonical k-Steiner tree $\mathcal{T}$. An arc vw indicates that w is a determinator of v and an undirected edge vw indicates that vw is an undetermined edge; for example, t_1s_1, s_6t_9 are the two undetermined edges of $\mathcal{T}$. There are two primary clusters in $\mathcal{T}$, which are $\{s_0, s_4\}$ and $\{s_6\}$. As previously mentioned, these clusters induce primary cluster subtrees, with leaves that are all terminals (t_0, t_6, t_7, t_{11} and t_8, t_{10} respectively). There are three additional clusters $\{s_1, s_2\}$, $\{s_3\}$ and $\{s_5\}$ in $\mathcal{T}$.

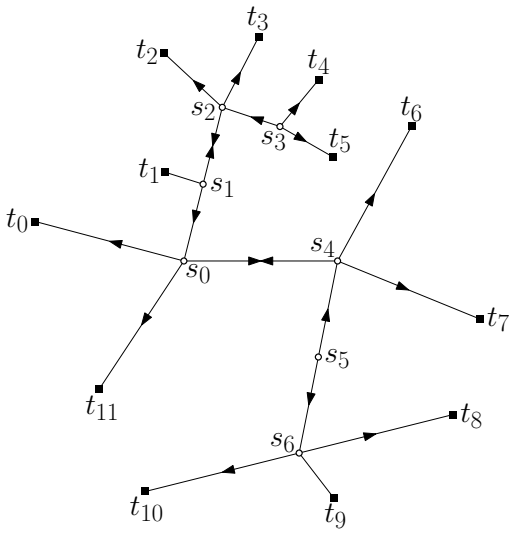


FIGURE 4.2: A canonical 7-Steiner tree where the arrows indicate the determinators of each Steiner point.

In the example of Figure 4.2, the clusters of $\mathcal{T}$ can be arranged according to the partial ordering, as shown below in Figure 4.3. If there exists a directed path from cluster $[s_i]$ to cluster $[s_j]$ in this figure, then $r_i \leq r_j$.

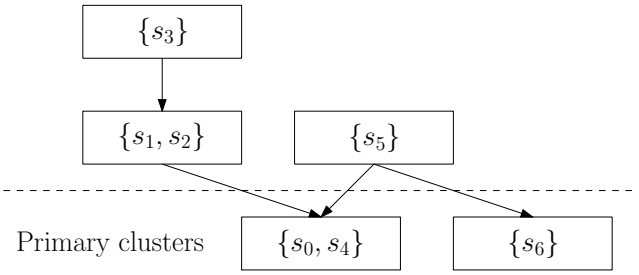


FIGURE 4.3: Hierarchy of the clusters in the canonical 7-Steiner tree in Figure 4.2.

Cluster subtrees have a very specific form, which we now detail.

Definition 4.8. Let T_i, T_j be two distinct cluster subtrees of $\mathcal{T}$ such that $[s_i] \preceq [s_j]$. Then T_j is called an *ancestor* of T_i. If T_i and T_j also have a Steiner point s in common then T_j is called a *parent* of T_i and s is called a *quasi-terminal of* T_i.

For example, consider the canonical k-Steiner tree shown in Figure 4.2. As can be seen from Figure 4.3, the cluster subtree induced by $\{s_0, s_4\}$ is an ancestor of the cluster subtree induced by $\{s_3\}$. It is also the ancestor and the parent of the cluster subtree induced by $\{s_1, s_2\}$. Hence, the Steiner point s_0 shared between the cluster subtrees induced by $\{s_1, s_2\}$ and $\{s_0, s_4\}$ is the quasi-terminal of the cluster subtree induced by $\{s_1, s_2\}$.

Definition 4.9. A *cluster topology* $\mathbb{T}$ is a graph with tree topology with the following properties:

1. Every leaf of $\mathbb{T}$, except for at most one, is a terminal or a quasi-terminal. Any leaf that is not a terminal or a quasi-terminal is called a *leaf bud*.
2. Every node of $\mathbb{T}$ that is not a leaf is a Steiner point.
3. Every Steiner point of $\mathbb{T}$ that is not a quasi-terminal has degree 2 or 3.

Leaf buds, mentioned in the first property of cluster topologies, will be used during the generation phase of our algorithm and will be discussed in detail in Section 4.3.1. A cluster topology where every leaf is a terminal is called a *primary* cluster topology.

Observation 4.10. Cluster subtrees have a cluster topology, and in particular, primary cluster subtrees have a primary cluster topology.

Our proposed algorithm iteratively generates the cluster subtrees of a canonical k-Steiner tree by starting with the primary cluster subtrees and then working down the list of direct ancestors. The details will be given in Section 4.3.

4.2 Algorithm overview

Note that, given any set of points N' in the plane, a minimum spanning tree on N' always minimises the length of a longest edge in a network spanning N'. This follows directly

from the matroid property of minimum spanning trees (or, as a direct consequence of our definition of k-BSTs). Therefore, the key challenge in the minimum bottleneck k-Steiner network problem is to find an optimal set of Steiner points. For general k, this is an NP-hard problem.

The algorithm we propose consists of two main stages, a Steiner point generation phase and an integer programming phase. In the first stage, the *generation phase*, a set $\mathcal{S}$ of Steiner points in $\mathbb{R}^2$ is produced. This set has the property that there exists a subset of it, say $\mathcal{S}^* \subseteq \mathcal{S}$, where $|\mathcal{S}^*| \leq k$, such that a minimum spanning tree on $N \cup \mathcal{S}^*$ is a minimum bottleneck k-Steiner network on N. The principal aim of the first stage is to produce an $\mathcal{S}$ with as small a cardinality as possible. This allows the second stage, which uses an integer linear program (ILP) to select $\mathcal{S}^*$ from $\mathcal{S}$, to proceed as efficiently as possible.

To produce the set $\mathcal{S}$ we effectively generate every possible cluster subtree (with at most k Steiner points) on nodes in N, but we use certain *pruning* conditions to eliminate as many sub-optimal cluster subtrees as possible (specifically, cluster subtrees which violate properties of k-BSTs). The set $\mathcal{S}$ is then produced by collecting together all Steiner points of all the cluster subtrees that remain at the end of the generation process. The generation phase will be described in more detail in Section 4.3.

In the second phase of the algorithm, which we call the *ILP phase*, we are required to solve the minimum bottleneck k-Steiner network problem with Steiner points selected from $\mathcal{S}$. This is a combinatorial problem and can therefore be solved exactly by integer programming methods. The ILP phase will be discussed in Section 4.4.

4.3 Generation Phase

Recall that the purpose of the generation phase is to construct a set $\mathcal{S}$ containing an optimal set of Steiner points. Starting from terminals, the generation algorithm first iteratively "grows" primary cluster topologies. Each growing step can occur in one of two ways: (1) by combining (*merging*) two partially completed primary cluster topologies (referred to as *branches*) and thereby producing a new degree-3 Steiner point; or (2) by *beading* a branch, which introduces a new degree-2 Steiner point (a *bead*). The growth continues until all leaves of the topology are terminals, where the final step is known

as *termination*. When the primary cluster topology is complete, an embedding process is initiated which finds the optimal locations of all Steiner points with respect to the generated topology. This results in an (embedded) primary cluster subtree where every edge is of the same length. Various *pruning tests* for optimality are then performed in order to determine whether the cluster subtree can be discarded.

Once all possible primary cluster subtrees have been generated, the algorithm proceeds similarly for the (non-primary) cluster topologies. Since all Steiner points of the primary cluster trees have known locations at this juncture, these are designated as potential quasi-terminals for the next round of cluster topology generation. As before, starting from terminals and/or quasi-terminals, *branches* are iteratively grown and then *terminated*, before an embedding process produces a cluster subtree. This process repeats until effectively all potential k-BSTs have been considered (minus the trees that are discarded by pruning). The set $\mathcal{S}$ is then produced by collecting together all Steiner points in all cluster trees that survived the pruning process.

The main concepts and processes during generation, which we now discuss in more detail, are therefore *branches*, *merging*, *beading*, *termination*, *pruning*, and the process of optimally embedding a cluster topology.

Firstly, we require the following definition.

Definition 4.11. For a given set of terminals N, the *instance upper-bound* for N is a non-negative real number $b_{\max}$ such that, for any minimum bottleneck k-Steiner network T^* on N, we have $|T^*| \leq b_{\max}$.

The longest edge of an MST on the set of terminals N provides a crude upper bound, but a better upper bound will be developed in 4.3.7.

4.3.1 Branches

We first introduce the concept of branches, which are the building blocks used to generate cluster subtree topologies. An analogous concept is used in the k-Steiner tree algorithm in Chapter 3.

Definition 4.12. A *branch* β_i is a cluster topology with an associated region R_i and a leaf bud r_i which is constrained to lie within R_i. In addition, every terminal and quasi-terminal in β_i has a known location in the plane.

The leaf bud r_i is a vertex that develops into either a degree-2 Steiner point (a bead), a degree-3 Steiner point or a terminal in the subsequent bead, merge or termination operation, respectively. The region R_i containing r_i is called the *leading region* of β_i.

Branches are grown starting from terminals and quasi-terminals. Therefore we introduce the following notion.

Definition 4.13. A *base* branch is a branch β_0 with leaf bud r_0 and leading region R_0, such that β_0 contains exactly one vertex t_0 besides r_0. Node t_0 is a terminal or quasi-terminal and R_0 is the disc of radius $b_{\max}$ centred at t_0.

Figure 4.4 shows an example of a base branch. The leaf bud of this branch must lie within a distance of $b_{\max}$ from t_0.

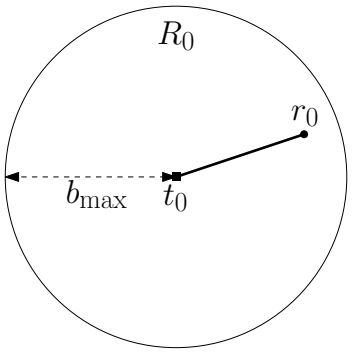


FIGURE 4.4: The leading region R_0 of a base branch β_0 with a terminal t_0 and root bud r_0.

The main purpose of the leading region is determine whether two branches can be merged together, or if a branch can be terminated at a given terminal. Essentially, two branches can be merged if their leading regions have non-empty intersection; and a branch can be terminated at a terminal t if t is contained in the leading region of the branch. The leading region of a branch is constructed by iteratively using the Minkowski sum operation, beginning with discs of radius $b_{\max}$ centred around terminals or quasi-terminals. Whenever two branches β_1, β_2 are merged, the leading region of the resulting branch will be the Minkowski sum of a disc of radius $b_{\max}$ and the intersection of R_1 and R_2.

We will present these concepts in more detail in the sections below. For now, we simply provide a figurative example in Figure 4.5.

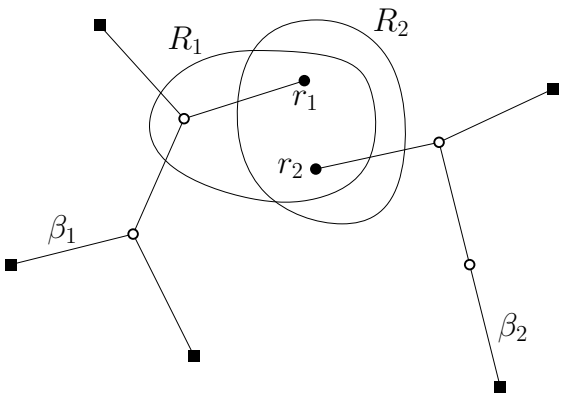


FIGURE 4.5: Two branches β_1 and β_2 with intersecting leading regions R_1 and R_2.

4.3.2 Merging

When *merging* two branches, say β_1, β_2, the leaf buds r_1, r_2 are replaced by a single new degree-3 Steiner point s, thereby combining β_1 and β_2 into a single branch β. A new leaf bud r for β is introduced, which is adjacent to s. The leading region of β, which contains s, is constructed by a Minkowski operation on the intersection of R_1 and R_2. We now describe this process in more detail and provide some examples. We begin with a definition.

Definition 4.14. Let B be the disc of radius $b_{\max}$ centred at the origin and let R be a region in $\mathbb{R}^2$. The region R^+ is the Minkowsi sum of R and B. In other words, $R^+ := \{a + b \mid a \in R, b \in B\}$.

Let β_1 and β_2 be two branches with leaf buds r_1, r_2 and leading regions R_1, R_2, respectively. Let k_1, k_2 be the number of Steiner points in β_1 and β_2 respectively. If $k_1 + k_2 \leq k - 1$ and $R_1 \cap R_2 \neq \emptyset$ then it is possible to merge β_1 and β_2. The merged branch of β_1 and β_2, say β, has the following properties.

1. The topology of β is obtained by taking the graph union of β_1 and β_2, where r_1 and r_2 are identified as a common Steiner point, denoted s; and by adding leaf bud r and edge rs.

2. Since s is constrained to lie in $R_1 \cap R_2$, it follows that any nodes adjacent to s in branch β, including r, lies in $(R_1 \cap R_2)^+$, which is the leading region of β.

3. β contains $k_1 + k_2 + 1$ Steiner points.

Figures 4.6 - 4.9 show an example of two subsequent merges that may arise during the generation stage. We begin with two base branches β_1 and β_2 (Figure 4.6). The regions have a nonempty intersection and hence the base branches can be merged together. The leading region R' of the merged branch is $(R_1 \cap R_2)^+$ (Figure 4.7). The resulting branch can be merged with another base branch β_3 since $R' \cap R_3 \neq \emptyset$ (Figure 4.8). The branch resulting from this new merge is shown in Figure 4.9.

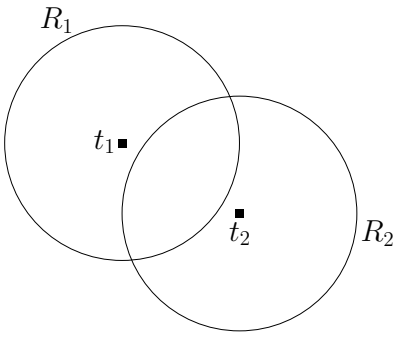


FIGURE 4.6: The leading regions R_1 and R_2 intersect and hence branches β_1 and β_2 can be merged.

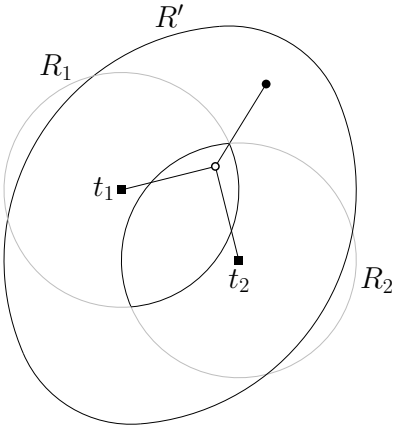


FIGURE 4.7: Merging β_1 and β_2 generates a new branch β' with leading region R'.

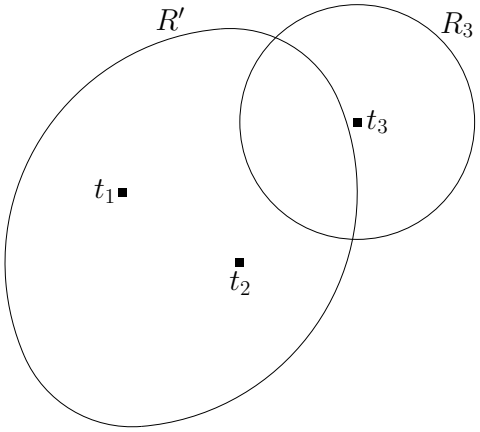


FIGURE 4.8: The leading region R' intersects the leading region R_3 of β_3 and hence can be merged.

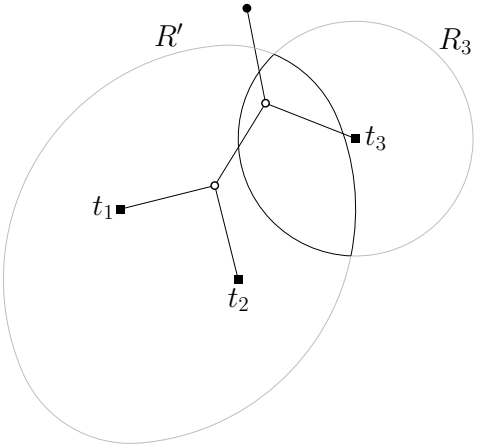


FIGURE 4.9: Merging β' and β_3.

4.3.3 Beading

In this subsection we discuss *beading*, which is a process that introduces a new degree-2 Steiner point to a branch. When a branch is beaded, the leaf bud forms a degree-2 Steiner point and a new leaf bud adjacent to the Steiner point is generated.

Let β_i be a branch with $k-1$ or fewer Steiner points, with a leading region R_i and leaf bud r_i. Suppose that β_i is beaded to generate a new branch β with a new degree-2 Steiner s, leading region R, and leaf bud r. Branch β has the following properties:

1. The topology of β is defined by converting r_i into the new degree-2 Steiner point s. A new node r is then introduced in addition to the edge rs. Node r is designated as the leaf bud of β.

2. The new leaf bud r must lie within a distance of $b_{\max}$ from s. Therefore, since s lies in R_i, we define the leading region of β as $R := R_i^+$. This is shown in Figure 4.10.

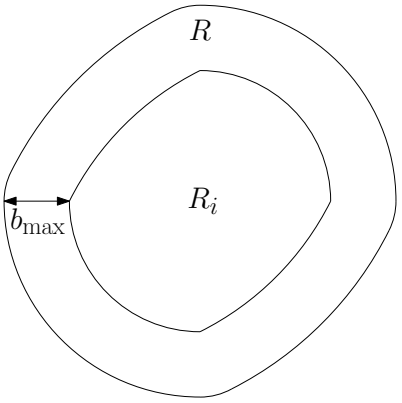


FIGURE 4.10: The leading region $R = R_i^+$.

Figure 4.11 shows a branch β_i with leaf bud r_i. This branch is beaded to generate a new branch β with leaf bud r (shown in Figure 4.12). The previous leaf bud, r_i, becomes a degree-2 Steiner point s in β.

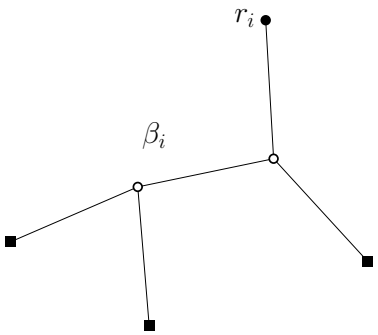


FIGURE 4.11: A branch β_i with a leaf bud r_i.

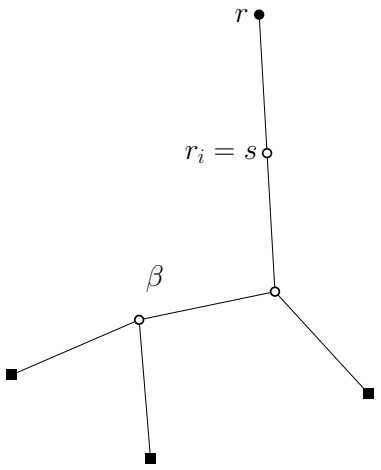


FIGURE 4.12: Branch β_i is beaded to generate a new branch β. In the process, r_i becomes a degree-2 Steiner point s, while a new leaf bud r and the edge rs are introduced.

4.3.4 Termination

The termination of a branch results in the generation of a cluster topology where the location of all its leaves are known. For a primary cluster topology, all its leaves after termination will be terminals. In general, after termination, a cluster topology's leaves will be terminals or quasi-terminals with known locations, and its internal nodes will be Steiner points with unknown locations. The locations of the Steiner points are determined during the embedding process, described below in Subsection 4.3.5.

Let β be a branch with a leading region R and leaf bud r. If a terminal (or quasi-terminal) t lies inside R, then β can be terminated to produce a cluster topology, say $\mathbb{T}$, where t replaces the leaf bud r. We say that β has been *terminated at* t *to produce* $\mathbb{T}$.

As an example, let β be the branch resulting from merging base branches β_i and β_j (with leading regions R_i and R_j respectively). Then, the leading region of β is $R := (R_i \cap R_j)^+$. Suppose that there exists a terminal $t \in R$. Then, branch β can be terminated at t, resulting in a cluster subtree topology whose leaves are t_i, t_j and t. This process is shown below in Figure 4.13.

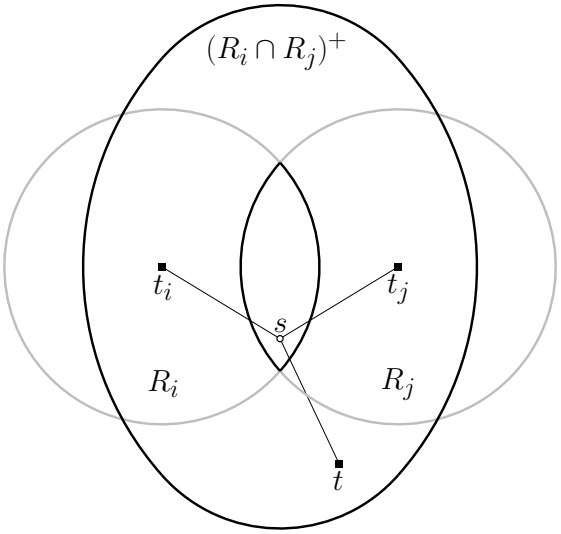


FIGURE 4.13: Branch β with leading region $R = (R_i \cap R_j)^+$ can be terminated at t, since $t \in R$.

4.3.5 Optimally embedding a cluster topology

Once a branch has been terminated to produce a cluster topology $\mathbb{T}$, the next step is to find the optimal locations of the internal points of $\mathbb{T}$, which are all Steiner points. If successfully done, this will result in a cluster subtree T. Note that T should have a number of specific properties inherited from k-BSTs. For instance, every edge in T should have the same length. However, not every cluster topology produced during generation has a valid embedding satisfying even this property. Clearly in such cases the cluster topology can be discarded, since it cannot be part of an optimal solution on N.

We now formalise the embedding process.

Definition 4.15. Let $\mathbb{T}$ be a cluster topology where every leaf of $\mathbb{T}$ has a fixed location in the plane. An *optimal embedding of* $\mathbb{T}$, if one exists, is a tree T with the same topology as $\mathbb{T}$, such that (1) T is embedded in the plane (i.e., every internal node of T has a known location); (2) every edge of T has the same length $l \leq b_{\max}$ and; (3) for all trees of topology $\mathbb{T}$ satisfying these two properties, T has the minimum cost.

To find an optimal embedding of $\mathbb{T}$ we employ the the procedure developed by Bae et al. in [18]. We briefly describe the method here.

Let q be the number of degree 3 Steiner points in $\mathbb{T}$. Since the locations of beads can be found by introducing weights to edges, this gives us $2q+1$ unknowns (namely, a coordinate pair for each of the q Steiner points and the edge length l). Note that a cluster topology with q degree-3 Steiner points has $q+2$ leaves, and therefore has $q+2+q-1$ (beaded) edges. Therefore, a total of $2q+1$ equations, one for each edge, can be created and the system can be solved as a series of quadratic equations. If there is more than one solution, we pick a solution with smallest l. Once the degree-3 Steiner points are located, beads are then evenly distributed along each of the beaded edges. The complexity of the embedding process is $2^{O(m)}$ where m is the number of edges [18].

We now provide a simple example. The cluster subtree topology in Figure 4.14 has five unknowns: $s_0^x, s_0^y, s_1^x, s_1^y, l$ where $s_i = (s_i^x, s_i^y)$ for $i = \{0,1,2\}$. The five equations come from the five edges t_0s_0, t_3s_0, s_0s_1, t_1s_1 and s_1t_2. Equations 4.1 - 4.4 correspond to the edges with no beads and Equation 4.5 corresponds to the edge with a single bead.

$$(t_0^x - s_0^x)^2 + (t_0^y - s_0^y)^2 = l^2 \tag{4.1}$$

$$(t_3^x - s_0^x)^2 + (t_3^y - s_0^y)^2 = l^2 \tag{4.2}$$

$$(s_1^x - s_0^x)^2 + (s_1^y - s_0^y)^2 = l^2 \tag{4.3}$$

$$(s_1^x - t_1^x)^2 + (s_1^y - t_1^y)^2 = l^2 \tag{4.4}$$

$$(s_1^x - t_2^x)^2/4 + (s_1^y - t_2^y)^2/4 = l^2 \tag{4.5}$$

Once the values of $s_0^x, s_0^y, s_1^x, s_1^y, l$ are calculated, we find the location of the degree-2 Steiner point s_2 by letting $s_2^x = (s_1^x + t_2^x)/2$ and $s_2^y = (s_1^y + t_2^y)/2$

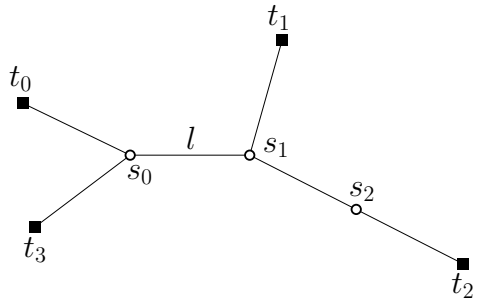


FIGURE 4.14: A cluster topology with four terminals, two degree-3 Steiner points and a single degree-2 Steiner point.

4.3.6 Pruning tests

Here, we introduce various optimality tests designed to eliminate cluster subtrees that cannot form a minimum k-BST.These tests are employed after the cluster subtree topologies have been embedded.

4.3.6.1 Lune Test

The lune test is an example of an empty region test; for every edge of a cluster subtree, it defines a region around the edge (called a *lune*) that cannot contain any vertices. Firstly, we define a lune.

Definition 4.16. Let uv be an edge. Then, we define the open region consisting of all points x such that $|xu| < |uv|$ and $|xv| < |uv|$ as the *lune of* uv and denote it by $L(u, v)$.

As a k-BST is an MST on the set of its vertices (Theorem 4.5), it immediately follows that it must satisfy the lune test.

Corollary 4.17. *Let T be a k-BST. Then, for any edge $uv \in E(T)$, its lune $L(u, v)$ does not contain any vertices in $V(T)$.*

When a cluster subtree topology $\mathbb{T}$ is successfully embedded, a cluster subtree T is generated. Every vertex of T has a known location and hence, we can check the lune of every edge and verify whether it contains any terminals or any Steiner points of T. If it does contain any of the aforementioned vertices, then the cluster subtree does not satisfy the lune test and is therefore discarded.

4.3.6.2 Cross Lune Test

The cross lune test extends the lune test as follows. When a nonprimary cluster subtree is generated, if any of its internal Steiner points lie inside any of the lunes of its parent cluster subtree or vice versa, then the two cluster subtrees cannot be part of the same minimum k-BST. As any nonprimary cluster subtree cannot exist without its parent cluster subtree, we can therefore discard the nonprimary cluster subtree.

For example, the FBST shown in Figure 4.15 does not satisfy the cross lune test as s_3, an internal Steiner point of a nonprimary cluster subtree, lies inside the lune $L(s_1, s_2)$ of its parent cluster subtree.

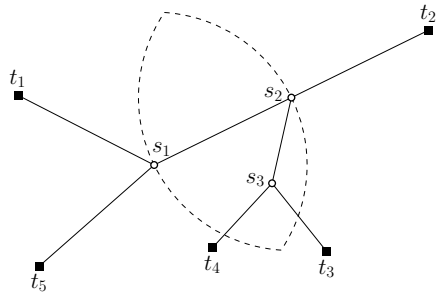


FIGURE 4.15: An example of a nonprimary cluster subtree that does not satisfy the cross lune test.

4.3.6.3 Acute Angle Test

As discussed in Section 4.1, since every Steiner point s of a k-BST lies at the centre of the smallest disk spanning its neighbours, it follows that s lies in the acute angled triangle formed by the determinators of s (or, it coincides with the midpoint of the determinators of s if s has exactly two determinators).

Now, the generation process of cluster subtree topologies guarantees that, for every such topology $\mathbb{T}$, there exists an embedding of $\mathbb{T}$ where every edge has a length of at most $b_{\max}$. However, when creating an optimal embedding of $\mathbb{T}$ we enforce that every edge has the same length. This may create an embedding in which at least one Steiner point is not at the centre of the smallest disk spanning its neighbours, and hence the triangle formed by the neighbours of the Steiner point is obtuse.

We now provide an example in Figure 4.16. Suppose that we have two base branches β_i and β_j with leading regions R_i and R_j respectively. Since $R_i \cap R_j \neq \emptyset$, we can merge these branches and obtain a new branch β with leading region $R = (R_i \cap R_j)^+$. As terminal $t \in R$, we can terminate branch β at t, generating a cluster subtree topology C. However, as illustrated in Figure 4.16, forcing the edges ts, t_is and t_js to have equal lengths places the Steiner point s outside the $\triangle tt_it_j$, which is obtuse. Hence, it cannot be a k-BST.

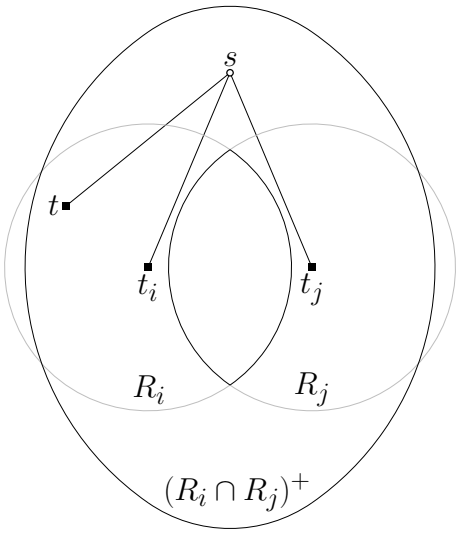


FIGURE 4.16: As $\triangle tt_it_j$ is obtuse, forcing the edges st, st_i and st_j to have equal lengths means that the Steiner point s does not coincide with the centre of the smallest disc containing t, t_i and t_j.

Hence, we implement a pruning test which we call the *acute angle test* to ensure that the neighbours of every internal Steiner point of an embedded cluster subtree forms an acute-angled triangle.

4.3.6.4 Convex Hull Test

A weaker (but more easily implemented) test is the *convex hull test*. It follows directly from the above acute triangle property that every Steiner point s must lie in the convex hull of the leaves of the cluster subtree containing s internally.

In Figure 4.17 we provide an example to show that the convex hull test is strictly weaker than the acute angle test.

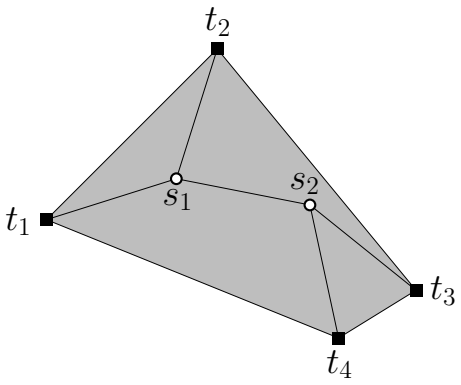


FIGURE 4.17: An example of a cluster subtree that does not satisfy the acute angle test but satisfies the convex hull test. The Steiner point s_2 does not lie inside the convex hull of s_1, t_3 and t_4 but the internal Steiner points s_1 and s_2 lie inside the convex hull of the terminals t_1, t_2, t_3 and t_4.

4.3.7 Calculating $b_{\max}$

In this section we aim to find an upperbound $b_{\max}$ for the cost of an optimal bottleneck k-Steiner network on N. We use two heuristic algorithms (Algorithm 5 and Algorithm 6) to calculate two upper bounds and then select the smaller of the two.

In Algorithm 5, we first find an MST that spans N. Then, we bead the edge with the longest length by introducing a new bead and then evenly distributing all beads along the beaded edge. We repeat this process $k-1$ more times and then find the length of the longest edge. This is a well-known approximation algorithm with a performance ratio of 2 (the proof is covered in [15]).

Below is the pseudocode for Algorithm 5.

Algorithm 5: Beaded MST heuristic

Input: A set of terminals N and a non-negative integer k.

Output: An upper bound $b^1_{\max}$.

1 Obtain an MST T with edges $E(T) = \{e_1, e_2, ..., e_{n-1}\}$ on the set of terminals N.

2 Let l_i be the length of edge e_i.

3 Let $z_i := 1 \quad \forall\ i \in \{1, 2, ..., n-1\}$.

4 **for** $j = 1$ *to* k **do**

5 $\quad i^* := \underset{i \in \{1,2,...,n-1\}}{\operatorname{argmax}} \frac{l_i}{z_i}$

6 $\quad$ Let $z_{i^*} := z_{i^*} + 1$.

7 Let $b^1_{\max} := max(\{l_1/z_1, l_2/z_2, ..., l_{n-1}/z_{n-1}\})$.

As shown in [15], the complexity of Algorithm 5 is $O((n+k)\log_2 n)$.

The next algorithm was first used as a heuristic for the classical Steiner tree problem in [14]. Here, we tailor this algorithm for the minimum bottleneck k-Steiner network problem by modifying the cost of a tree to be the length of the longest edge, as opposed to the sum of the length of its edges.

In Algorithm 6, we first find an MST that spans $V := N$. Then, for every pair of terminals $a, b \in N$, we calculate its midpoint v_{ab} and for every triple $d, e, f \in N$, we calculate its circumcentre v_{def}. For evey pair and every triple of nodes in N we then construct an MST that spans $V \cup \{v_{ab}\}$ and an MST spanning $V \cup \{v_{def}\}$. Amongst all such pairs and triples, let $v_1 \in \{v_{ab}, v_{def}\}$ be a vertex such that the length of the longest edge of an MST spanning $V \cup \{v_1\}$ is the smallest. Now, let $V := V \cup \{v_1\}$. Next, we repeat the process $k-1$ more times until we obtain $V = N \cup \{v_1, v_2, ..., v_k\}$. An MST spanning V is a k-Steiner tree on N, hence, the length of the longest edge of an MST spanning V is an upper bound on the length of an edge in a minimum k-BST.

Algorithm 6: Iterative 1-Steiner heuristic

Input: A set of terminals N and a non-negative integer k.

Output: An upper bound $b^2_{\max}$.

```
1  V := N
2  for i ∈ {1, 2, ..., k} do
3      Let M be some large number.
4      for every pair of vertices {v_a, v_b} in V do
5          Let v' be the midpoint of points v_a and v_b.
6          Obtain an MST T on the set of points V ∪ {v'}.
7          Let e be a longest edge in E(T).
8          if |e| < M then
9              M := |e|
10             v* := v'
11     for every triple of vertices {v_a, v_b, v_c} in V do
12         Let v'' be the circumcentre of points v_a, v_b and v_c.
13         Obtain an MST T on the set of points V ∪ {v''}.
14         Let e be a longest edge in E(T).
15         if |e| < M then
16             M := |e|
17             v* := v''
18     V := V ∪ {t*}
19 b^2_max := M
```

It takes $O(n \log n)$ time to calculate an MST on N. In each iteration, we must find an MST for each every pair or triple of vertices, resulting in selecting from a set of $n + k - 1$ vertices in the final iteration. This results in $O((n + k)^3 n \log n)$ time. There are a total of k iterations in the algorithm, so the total complexity of the algorithm is $O((n + k)^3 kn \log n)$.

We now let

$$b_{\max} := \min\{b^1_{\max}, b^2_{\max}\}.$$

4.3.8 Formal description of the generation phase

In this section we describe the generation phase in more detail, as well as provide pseudocode. The first step of the generation phase is to calculate an upper bound $b_{\max}$ for the length of any edge in a cluster subtree. This is achieved by the method detailed in Section 4.3.7. Next, the primary cluster subtrees are generated by first producing base branches at every terminal in N and then iteratively employing the bead, merge and termination operations defined above. Whenever a branch is terminated to produce a cluster topology $\mathbb{T}$, an optimal embedding of $\mathbb{T}$ (if one exists), say T, is produced using the method described in Section 4.3.5. The tree T is then subjected to pruning tests. If T passes all the tests then it is stored in a member of a set $\mathcal{C}$. The set $\mathcal{C}$ will be defined formally below, but for now it suffices that $\mathcal{C}$ is a set of subsets of cluster subtrees.

Next, the nonprimary cluster subtrees are generated in a similar manner. Instead of initiating the branches at terminals only, during nonprimary generation we also create base branches at internal Steiner points of "feasible" trees in the members of $\mathcal{C}$. These Steiner points will therefore become quasi-terminals of the next generation of branches. As with primary cluster generation, we then iteratively perform the merge, bead and termination procedures, followed by optimal embedding and pruning operations.

We now describe one more property of nonprimary cluster subtree generation. Nonprimary cluster subtree generation takes places in a series of iterations, where all branches grown at a given iteration terminate at the internal Steiner points of a fixed tree in a member of $\mathcal{C}$. We refer to this tree as a *termination cluster* for that iteration. The purpose of termination clusters is to avoid duplicate generation of cluster subtrees. Termination clusters also ensure that any generated cluster subtree has at least one leaf which is a quasi-terminal (therefore the cluster subtree is not primary), while other leaves can be terminals.

Once the termination cluster, say T', has been chosen, we then grow branches as described above and ensure that all branches terminate at an internal Steiner point of T'. These Steiner points are then designated as quasi-terminals of the newly generated cluster topologies. After embedding these topologies and running the pruning tests, the resulting cluster subtrees are also placed in a member of $\mathcal{C}$. We then select the next termination cluster from a member of $\mathcal{C}$ and repeat.

We now make the above description of non-primary generation more rigorous.

Non-primary generation

Let C_i, $1 \leq i \leq k$ be all embedded cluster subtrees with i Steiner points generated during primary generation. Let $\mathcal{C} := \{C_1, ..., C_k\}$. The set $\mathcal{C}$ is then updated during non-primary generation. The non-primary generation algorithm loops through each $i \in \{1, ..., k-1\}$ and, for every such i, loops through every $T_j \in C_i$. For given i and j, the tree T_j is fixed as the termination cluster. Therefore all branches generated for the i, j iteration pair will terminate at an internal Steiner point of T_j. For each i, j, our algorithm initiates the generation process by creating base branches at terminals and internal Steiner points of existing cluster subtrees in members of $\mathcal{C}$. However, it limits the choice of these terminals and potential quasi-terminals to the *base vertices* for T_j, defined as follows.

Definition 4.18. Let T_j be a termination cluster. Then the *base vertices* for T_j, denoted N_j, contains all terminals in $N \setminus V(T_j)$, as well as all internal Steiner points of every tree H in a member of $\mathcal{C}$ such that:

1. H contains at most $k - k(T_j) - 1$ Steiner points, where $k(T_j)$ is the number of Steiner points in T_j.

2. H does not share any terminals with T_j.

3. H has been selected as the termination cluster at a previous iteration of nonprimary generation.

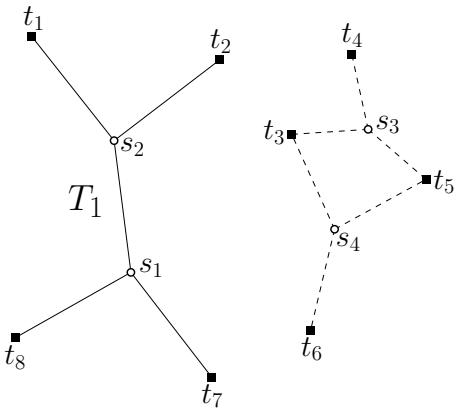


FIGURE 4.18: The set of base vertices N_1 contains the degree-3 Steiner points s_3 and s_4.

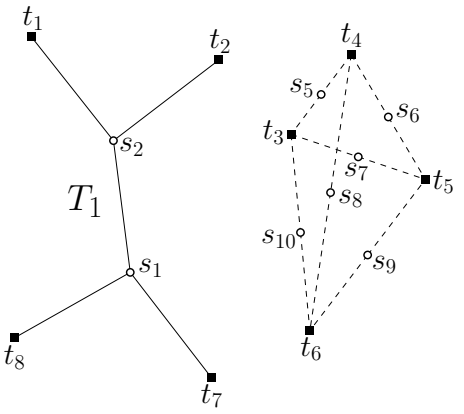


FIGURE 4.19: The set of base vertices N_1 contains the degree-2 Steiner points $s_5, s_6, ..., s_{10}$.

Suppose that T_1 is a primary cluster subtree for an instance with $N = \{t_1, t_2, ..., t_8\}$ and $k = 4$, as illustrated in Figures 4.18 and 4.19. By Definition 4.18, the base vertices for T_1, denoted by N_1, consists of the terminals t_3, t_4, t_5, t_6 and internal Steiner points of cluster subtrees that satisfy properties 1 - 3. Since $k - k(T_1) - 1 = 1$, the set $\mathcal{C}$ consists of cluster subtrees with exactly one Steiner point that do not share any terminals with T_1. Since $k(T_1) = 2$, every cluster subtree with one Steiner point has been previously selected as the termination cluster. Hence, the set of base vertices for T_1 is $N_1 = \{s_3, s_4, ..., s_{10}\}$. Note that there are no more feasible degree-3 Steiner points in N_1 as $\angle t_4 t_3 t_6, \angle t_4 t_5 t_6 \geq \pi/2$.

Note that, by Condition 3, in the first iteration of non-primary generation, only terminals in $N \setminus V(T_j)$ are base vertices for T_j. Once the base vertices for T_j have been calculated, the non-primary generation procedure initiates base branches at each base vertex, and employs the merge and bead operations to grow branches. All branches are terminated at internal Steiner points of T_j.

The purpose of defining the base vertices for T_j in the manner above is as follows: Condition 1 of the definition ensures that we do not commence branches at internal Steiner points that would directly lead to a tree containing more than k Steiner points. Condition 2, as well as the condition that the base vertices do not contain terminals of T_j, ensures that we do not commence branches at points that would directly lead to an network containing cycles. Condition 3 mitigates double-counting, and will be discussed in more detail next.

Recall that once primary generation is complete, the set $\mathcal{C} = \{C_1, ..., C_k\}$ is such that, for each i, C_i contains primary cluster subtrees with i Steiner points. During non-primary generation, $\mathcal{C}$ continues to be populated by cluster subtrees, but the index i in C_i is extended as follows. For every i, every member of C_i is a cluster subtree where i is the total number of Steiner points in the cluster subtree *and all its ancestors*. The purpose of maintaining this extended index definition during non-primary generation is to keep track of which branches are able to merge with a cluster subtree in C_i (without violating the contraint of k Steiner points).

As a result of this indexing, nonprimary cluster subtrees are always allocated to a member of $\mathcal{C}$ with a higher index than its parent cluster subtree. To ensure that every cluster

subtree with at most $k-1$ Steiner points is selected as a termination cluster, we begin with those in C_1 and continue in ascending order.

Now, suppose two cluster subtrees T_i and T_j contain internal Steiner points s_i and s_j respectively such that Conditions 1 and 2 in Definition 4.18 are satisfied. Without Condition 3, s_i is a base vertex of T_j and s_j of T_i. However, this can result in duplicate generation of the same cluster subtree. For example, the nonprimary cluster subtree T' in Figure 4.20 may be generated twice, once with T_i as the termination cluster and once with T_j.

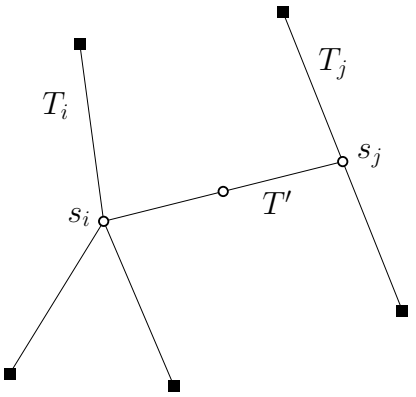


FIGURE 4.20: A cluster subtree T' with parent cluster subtrees T_i and T_j.

Pseudocode

We present the pseudocode of the primary generation algorithm in Algorithm 7.

Firstly, in Lines 1 and 2, we calculate $b^1_{\max}$ and $b^2_{\max}$ using Algorithms 5 and 6. We choose the smaller of the two bounds to be $b_{\max}$ in Line 3. In Lines 4 and 5, we initialise $B_0, ..., B_k$ and $C_1, ..., C_k$ to be empty. Sets B_i and C_i will be populated with all the branches and cluster subtrees with i Steiner points, respectively.

We utilise the index of the terminals to avoid repeated generation of primary cluster subtrees. A branch β is allowed to terminate with terminal, say t_i, only if the terminal with the largest index in β has a lower index than i. Therefore, in Lines 6 - 7 we generate base branches at every terminal except at the terminal with the largest index and add it to the set B_0, the set of branches with no Steiner points.

Throughout generation, to prevent merging the same pair of branches multiple times, a branch is only merged with another branch that contains a fewer number of Steiner

Algorithm 7: Primary Generation Algorithm

Input: A set of terminals $N = \{t_1, t_2, ..., t_n\}$ and a non-negative integer k.

Output: A set $\mathcal{C} = \{C_1, ..., C_k\}$ where C_i contains primary cluster subtrees with i Steiner points.

```
1  Run Algorithm 5 to obtain b^1_max.
2  Run Algorithm 6 to obtain b^2_max.
3  b_max := min{b^1_max, b^2_max}
4  Let C_i := ∅ for i = {1, 2, ..., k}.
5  Let B_i := ∅ for i = {0, 1, ..., k}.
6  for t_p ∈ N and p ≠ n do
7      Generate a base branch on t_p and add it to B_0.
8  for β_p ∈ B_0 do
9      Bead β_p and add it to B_1.
10     for β_q ∈ B_0 where q < p do
11         Merge β_p and β_q then add the resulting branch to B_1.
12 for i ∈ {1, 2, ..., k − 1} do
13     for β_p ∈ B_i do
14         Bead β_p and add it to B_{i+1}.
15         for t ∈ N do
16             Terminate β_p at t to generate a cluster subtree topology 𝕋.
17             Embed 𝕋 to generate a cluster subtree T.
18             Run pruning tests on T.
19             if pruning tests are satisfied then
20                 Add T to C_i.
21         for j ∈ {0, 1, ..., min{i, k − i − 1}} do
22             for β_q ∈ B_j where q < p whenever i = j do
23                 Merge β_p and β_q then add the resulting branch to B_{i+j+1}.
24 for β ∈ B_k do
25     for t ∈ N do
26         Terminate β at t to generate a cluster subtree topology 𝕋.
27         Embed 𝕋 to generate a cluster subtree T.
28         Run pruning tests on T.
29         if pruning tests are satisfied then
30             Add T to C_k.
```

points or an equal number of Steiner points but has a lower index. For example, a branch $\beta_i \in B_p$ can only be merged with branches $\beta_j \in B_q$ where $q < p$, or with branches $\beta_j \in B_p$ when $j < i$.

An example of this is observed in Lines 8 - 11. By employing the bead and merge operations, we generate the set β_1, i.e., the set of all branches with a single Steiner

point. In this case, to avoid double counting, we only merge branches β_q with β_p when $q < p$.

In Lines 12 - 23 we generate all branches of size at most k, and also all cluster subtrees of size at most $k-1$. Lines 14 and 21 - 23 generate the branches. As before, we either use the bead operation to increase the number of Steiner points in the branch by 1 (Line 14), or we employ the merge operation in Lines 21 - 23.In Lines 15 - 20 we terminate branches of size at most $k-1$, and then add the resulting cluster subtree to C_i after the embedding and pruning operations.

Finally, in Lines 24 - 30 we terminate branches of size k and add the resulting cluster subtrees to C_k.

We present the pseudocode of the non-primary generation algorithm in Algorithm 8. In this algorithm we denote the set of internal Steiner points of a cluster T_j by $S[T_j]$.

The input of Algorithm 8 is the set $\mathcal{C} = \{C_1, ..., C_k\}$ where C_i contains cluster trees with i Steiner points. Non-primary generation commences iterating over all $i \in \{1, ..., k-1\}$ and for each i iterating over all possible termination clusters $T_j \in C_i$.

Recall that, in non-primary generation, the set C_i, for $i \in \{1, ..., k\}$, will contain the cluster subtrees whose cumulative Steiner point count - including those from its ancestor cluster subtrees - is i. The algorithm begins by initialising $B_0, ..., B_k$ to be empty. Similarly to the indexing of the sets in $\mathcal{C}$, each set B_i will be populated with branches such that every branch in B_i has a cumulative (across ancestors) Steiner point count of i.

In Lines 5 - 10 the algorithm generates a set of base branches at each candidate vertex $v \in N_j$. Base branches at the terminals are added to B_0, but the base branches at Steiner points are added to B_q where q is the total number of Steiner points in the cluster subtree containing the Steiner point as well as all ancestors of that tree.

The remaining lines of code are similar to the corresponding code in Algorithm 7. Specifically, Lines 11 - 14 create branches with a single Steiner point using the bead and merge operations and Lines 17, 24 - 26 create branches of size up to k. Lines 27 - 33 terminate branches of size k to produce clusters with k Steiner points.

Algorithm 8: Nonprimary Generation Algorithm

Input: A set of terminals $N = \{t_1, t_2, ..., t_n\}$, a non-negative integer k and the set $\mathcal{C}$.

Output: An updated set $\mathcal{C} = \{C_1, ..., C_k\}$ where C_i is the set of cluster subtrees whose cumulative Steiner point count - including those from its ancestor cluster subtrees - is i.

```
1  for i ∈ {1, 2, ..., k − 1} do
2      for T_j ∈ C_i do
3          Let B_i := ∅ for i = {0, 1, ..., k}.
4          Generate the set of candidate vertices, N_j for T_j.
5          for v ∈ N_j do
6              if v is a Steiner point then
7                  Let q be the total number of Steiner points in the cluster subtree
                   containing v, including all Steiner points from its ancestor cluster
                   subtrees.
8                  Generate a base branch on v and add it to B_q.
9              else
10                 Generate a base branch on v and add it to B_0.
11         for β_p ∈ B_0 do
12             Bead β_p and add it to B_1.
13             for β_q ∈ B_0 where q < p do
14                 Merge β_p and β_q then add the resulting branch to B_1.
15         for r ∈ {1, 2, ..., k − i − 1} do
16             for β_p ∈ B_r do
17                 Bead β_p and add it to B_{r+1}.
18                 for s ∈ S[T_j] do
19                     Terminate β_p at s to generate a cluster subtree topology 𝕋.
20                     Embed 𝕋 to generate a cluster subtree T.
21                     Run pruning tests on T.
22                     if pruning tests are satisfied then
23                         Add T to C_{r+i}.
24                 for c ∈ {0, 1, ..., min{r, k − r − i − 1}} do
25                     for β_q ∈ B_c where q < p whenever r = c do
26                         Merge β_p and β_q then add the resulting branch to B_{r+c+1}.
27         for β ∈ B_{k−i} do
28             for s ∈ S[T_j] do
29                 Terminate β at s to generate a cluster subtree topology 𝕋.
30                 Embed 𝕋 to generate a cluster subtree T.
31                 Run pruning tests on T.
32                 if pruning tests are satisfied then
33                     Add T to C_k.
```

Termination occurs in Lines 19, where cluster trees having a cumulative Steiner point count of less than k are produced) and in Lines 29 (where cluster trees having a cumulative Steiner point count of exactly k are produced).

Note that, whenever termination takes place, it does so at vertices selected from the set $S[T_j]$, which are the internal Steiner points of T_j.

4.4 The ILP phase

4.4.1 Flow formulation

The generation algorithm outputs a set $\mathcal{S}$ of Steiner points, which is the set of all Steiner points of every cluster subtree surviving all pruning tests. Therefore, there exists a set $\mathcal{S}^* \subseteq \mathcal{S}$, with $|\mathcal{S}^*| \leq k$, such that a minimum spanning tree on $N \cup \mathcal{S}^*$ is a minimum bottleneck k-Steiner network on N. In this section we present a flow formulated ILP to find $\mathcal{S}^*$.

Let $\mathcal{S} = \{s_0, s_1, s_2, ..., s_{p-1}\}$ be the set of p Steiner points output by the generation algorithm and let $N = \{t_0, ..., t_{n-1}\}$ be the given set of terminals. Let $I := \{0, 1, ..., p+n-1\}$ be the combined index set of the terminals and Steiner points, where $i \in \{0, 1, 2, ..., p-1\}$ refers to Steiner point s_i and $j \in \{p, p+1, p+2, ..., p+n-1\}$ refers to terminal t_{j-p}.

Firstly, we let t_q, for some $q \in \{0, ..., n-1\}$, be an arbitrary terminal in N, designated as the root. For each $i \in \{0, ..., p-1\}$, we create a binary variable $x_i \in \{0, 1\}$, where $x_i = 1$ if and only if s_i is present in the solution. For every pair of indices $i, j \in I$ (where $i < j$), we define the binary edge variables $e_{i,j} \in \{0, 1\}$, where $e_{i,j} = 1$ if and only if the edge between vertex i and vertex j is present in a solution. Note that an edge between two vertices is feasible only if each endpoint is either a terminal or a Steiner point that has an associated binary variable value of 1. This will be enforced via a constraint in the ILP.

The flow variables are denoted by $f_{i,j}$, for every $i, j \in I$. These indicate the amount of flow on edge $e_{i,j}$ in the direction from i to j. Note that a nonzero flow may occur along edge $e_{i,j}$ only if this edge is present in a solution. As before, this is enforced by a constraint in the ILP.

Finally, the Euclidean length of the edge $e_{i,j}$ is denoted by $l_{i,j}$ and is pre-processed prior to the ILP.

We now present our ILP model, which is based on a classical flow model for connectivity.

$$
\begin{aligned}
\text{minimise} \quad & w & & \\
\text{subject to} \quad & w \geq l_{i,j} e_{i,j} & \forall\ i,j \in I,\ i<j & \quad (4.6)\\
& \sum_{i \in \{0,1,\ldots,p-1\}} x_i \leq k & & \quad (4.7)\\
& \frac{f_{i,j}}{n-1} \leq x_j & \forall\ i \in I,\ j \in \{0,1,\ldots,p-1\} & \quad (4.8)\\
& e_{i,j} \geq \frac{f_{i,j}+f_{j,i}}{n-1} & \forall\ i,j \in I,\ i<j & \quad (4.9)\\
& \sum_{i \in I} f_{i,u} - \sum_{j \in I} f_{u,j} = 0 & \forall\ u \in \{0,1,\ldots,p-1\} & \quad (4.10)\\
& \sum_{i \in I} f_{i,u} - \sum_{j \in I} f_{u,j} = 1 & \forall\ u \in \{p,p+1,\ldots,p+n-1\} \setminus \{p+q\} & \quad (4.11)\\
& \sum_{i \in I} f_{(p+q),i} = n-1 & & \quad (4.12)\\
& e_{i,j} \in \{0,1\} & \forall\ i,j \in I, i<j & \\
& f_{i,j} \geq 0 & \forall\ i,j \in I & \\
& x_i \in \{0,1\} & \forall\ i \in \{0,\ldots,p-1\} &
\end{aligned}
$$

The objective of the ILP is to minimise the length of the longest edge in a solution. This is achieved by defining a variable w which is greater than or equal to the length of every edge present in a solution (Line 4.6) and minimising it. Line 4.7 forces the total number of Steiner points in a solution to be at most k.

Line 4.8 ensures that there is zero flow into a Steiner point not in a solution. Line 4.9 ensures that if there exists a nonzero flow from vertex i to vertex j (or vice versa), that $e_{i,j} = 1$. Lines 4.10 and 4.11 force the net flow into a Steiner point to be zero and the net flow into a terminal that is not the root (t_q) to be 1. Finally Line 4.12 ensures that the net flow out of the root to is $n-1$ (the number of non-root terminals).

4.4.2 Edge-cut constraints

We now present our second ILP model, which employs edge-cut constraints to achieve connectivity.

Let $\mathcal{S} = \{s_0, s_1, s_2, ..., s_{p-1}\}$ be the set of p Steiner points output by the generation algorithm and $N = \{t_0, ..., t_{n-1}\}$ the set of terminals. We define $V := \mathcal{S} \cup N$ as the set of all vertices. As previously defined in Subsection 4.4.1, we define $I := \{0, 1, ..., p+n-1\}$ to be the combined index set of the terminals and Steiner points, where $i \in \{0, 1, 2, ..., p-1\}$ refers to Steiner point s_i and $j \in \{p, p+1, p+2, ..., p+n-1\}$ refers to terminal t_{j-p}.

For each Steiner point $s_i \in \mathcal{S}$, we create an associated binary variable x_i. The binary variable takes a value of 1 if the corresponding Steiner point is present in a solution and is 0 otherwise. Furthermore, we define the binary edge variables $e_{i,j}$ for $i, j \in I$ where $i < j$. This binary variable has a value of 1 if the edge between vertices with indices i and j is present in a solution and is 0 otherwise.

Let $V' \subset V$ be a subset of vertices such that $V' \cap N \neq \emptyset$ and $(V - V') \cap N \neq \emptyset$. Then, we define its edge cut to be $\delta(V') := \{(i, j) \mid v_i \in V' \text{ and } v_j \in V - V'\}$ where $v_i = s_i$ if $i \in \{0, 1, ..., p-1\}$ and $v_i = t_{i-p}$ if $i \in \{p, p+1, ..., p+n-1\}$. In other words, the edge cut $\delta(V')$ is the set of all edges with one endvertex in V' and the other in $V - V'$.

Now that we have defined the edge cut for a subset of vertices V', we now present our ILP formulation.

$$
\begin{aligned}
&\text{minimise} && w && && \\
&\text{subject to} && w \geq l_{i,j} e_{i,j} && \forall\ i,j \in I,\ i<j && (4.13)\\
& && \sum_{i \in \{0,1,\ldots,p-1\}} x_i \leq k && && (4.14)\\
& && e_{i,j} \leq x_i && \forall\ i \in \{0,1,\ldots,p-1\},\ j \in I,\ \ i<j && (4.15)\\
& && e_{i,j} \leq x_j && \forall\ i,j \in \{0,1,\ldots,p-1\},\ i<j && (4.16)\\
& && \sum_{(i,j)\in\delta(V')} e_{i,j} \geq 1 && \forall\ \emptyset \neq V' \subset V \text{ where } V' \cap N \neq \emptyset \text{ and } (V-V') \cap N \neq \emptyset && \\
& && && && (4.17)\\
& && e_{i,j} \in \{0,1\} && \forall\ i,j \in I, i<j && \\
& && x_i \in \{0,1\} && \forall\ i \in \{0,\ldots,p-1\} &&
\end{aligned}
$$

Similarly to the flow formulation described in Subsection 4.4.1, the objective of the ILP is to minimise the length of the longest edge in a solution. To do this, we define a variable w which is greater than or equal to the length of every edge present in a solution (Line 4.13) and then minimising it. Line 4.14 forces the number of Steiner points in a solution to be at most k.

Any edge whose endvertex is a Steiner point may be present in a solution only if the Steiner point is also present (Lines 4.15 and 4.16). Finally, for any subset of vertices $V' \subset V$ where V' contains at least one but not all the terminals, we force at least one edge in its edge cut to be present in a solution.

4.4.3 Comparison

In this subsection, we briefly compare the runtime of the ILP phase using the flow formulation versus the edge-cut formulation.

For 20 randomly generated instances, we first run the generation algorithm to produce a set of potentially optimal Steiner points. We then solve the ILP using both formulations with this identical set of Steiner points. Table 4.1 shows the average ILP runtime for $k = 3$ and $n = 5, 6, 7$.

Due to the long runtime of the flow formulation, testing for larger values of n was not feasible. The edge-cut formulation outperforms the flow formulation significantly, requiring only 101.88 seconds for $n = 7, k = 3$, compared to 6011.7 seconds with the flow formulation.

Number of terminals	5	6	7
Flow formulation	35.146	185.48	6011.7
Edge-cut formulation	7.5217	17.773	101.88

TABLE 4.1: ILP runtimes (in seconds) for the flow formulation and the edge-cut formulation with $k = 3$.

Therefore, we employ only the edge-cut formulation in the computational experiments presented in Section 4.5.

4.5 Experiments

In this section, we present computational experiments to demonstrate the performance of our algorithm and the effectiveness of the pruning tests.

4.5.1 Algorithm performance

For each terminal set size where $k = 3$, we randomly generate 20 instances, where each terminal is sampled from a uniform $[0, 1] \times [0, 1]$ distribution. We then use our algorithm to obtain a minimum bottleneck k-Steiner network for each instance.

Figure 4.21 presents the average runtime of our algorithm for $n = 5, 6, ..., 15$ and $k = 3$. The average runtime of our algorithm appears to increase exponentially as the number of terminals increases, with instances where $n = 15$ and $k = 3$ taking an average of 6070 seconds.

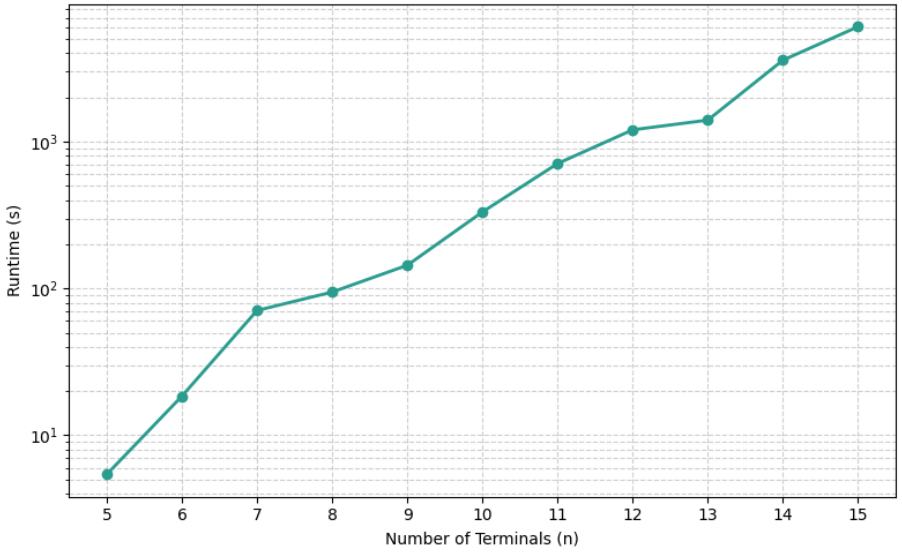


FIGURE 4.21: The average algorithm runtime over 20 randomly generated instances for $k = 3$.

TABLE 4.2: Percentage of Algorithm Time Spent in the ILP Phase for $k = 3$.

Terminals (n)	5	6	7	8	9	10	11	12	13	14	15
ILP Time %	86.91	88.58	93.94	91.56	89.82	91.56	94.85	95.94	94.98	96.92	97.56

We briefly discuss the relative runtime of the generation and ILP phases (shown in Table 4.2). First, we note that the ILP phase dominates the algorithm runtime, taking more than 85% of the algorithm runtime for all tested values of n. Generally, the proportion of time spent in the concatenation phase increases as n increases, with the concatenation phase taking up 97.56% of the algorithm time for $n = 15$.

4.5.2 Effectiveness of pruning tests

As in the previous subsection, we randomly generate 20 instances, with terminals sampled from a uniform distribution. However, in this experiment, we run only the generation phase for $k = 2$ and $k = 3$ to produce Steiner points that are potentially optimal. We then conduct the pruning tests on these Steiner points and compare the effectiveness of the implemented pruning methods in our algorithm.

The grey columns in Figures 4.22 and 4.23 represent the total percentage of the Steiner points that are pruned by at least one of the following pruning tests: the lune test, the acute angle test, the upper bound test and the cross lune test. The lune test appears to

be the most powerful pruning test, except for instances where n is small. This is followed by the acute angle test, the upper bound test and the cross lune test. Every pruning test prunes a larger portion of Steiner points as k increases from 2 to 3. However, their order of effectiveness remains consistent.

As shown in Tables 4.3 and 4.4, the lune test is the most powerful pruning test for all tested instances except when $n = 5$, pruning 97.1% of Steiner points for $n = 55$, $k = 2$ and 98.9% for $n = 40$ and $k = 3$. The cross lune test, despite its limited pruning potential, only prunes Steiner points that are not pruned by the lune test. Hence, the lune and cross lune pruning tests account for 99.1% of all the Steiner points pruned for $n = 40$ and $k = 3$ out of a total pruning rate of 99.8%. The acute angle test is the second most powerful, pruning a total of 94.5% of Steiner points for $n = 40$ and $k = 3$.

Finally, we note that the percentage of pruned Steiner points increases as n increases, with 99.8% of Steiner points being pruned for $n = 40$, $k = 3$ and 98.6% of Steiner points for $n = 55$, $k = 2$.

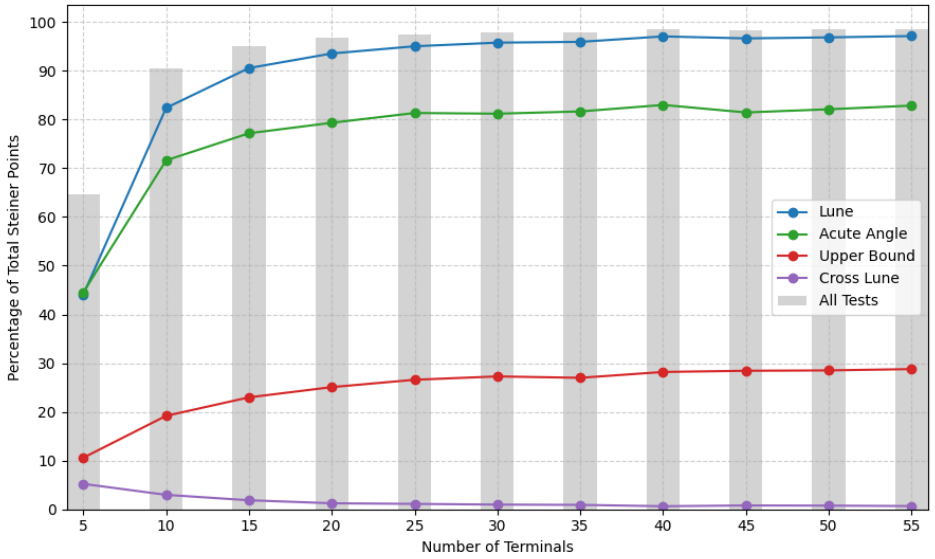


FIGURE 4.22: The percentage of Steiner points pruned by different pruning tests for $k = 2$.

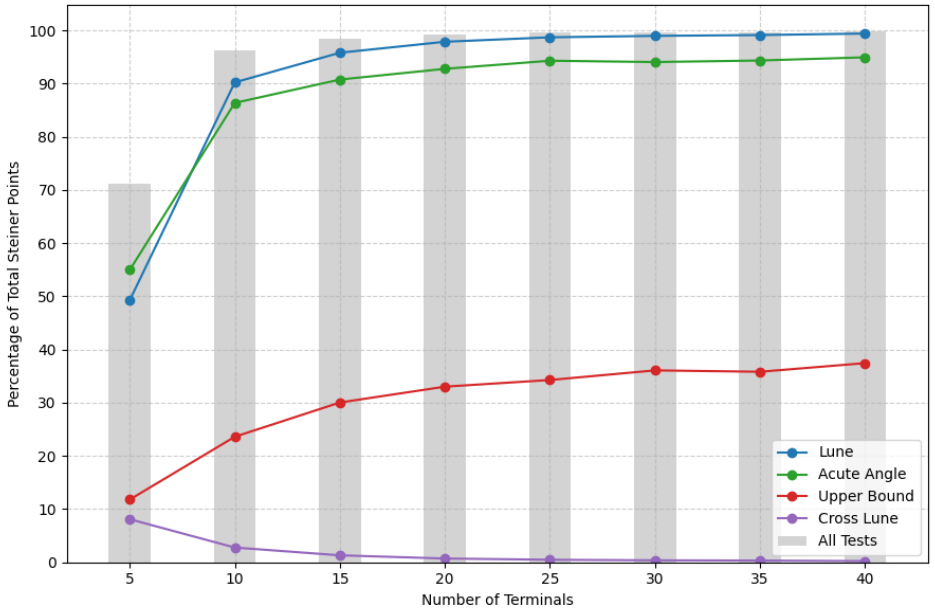


FIGURE 4.23: The percentage of Steiner points pruned by different pruning tests for $k = 3$.

Number of Terminals	5	10	15	20	25	30	35	40	45	50	55
Lune (%)	44.1	82.4	90.5	93.6	95.0	95.8	96.0	97.0	96.6	96.8	97.1
Acute Angle (%)	44.5	71.6	77.2	79.3	81.3	81.2	81.7	83.1	81.4	82.1	82.9
Upper Bound (%)	10.6	19.2	23.0	25.1	26.6	27.3	27.0	28.2	28.4	28.5	28.8
Cross Lune (%)	5.2	3.0	1.9	1.2	1.1	1.0	0.9	0.6	0.8	0.8	0.7
All Tests (%)	64.6	90.5	95.1	96.7	97.6	97.8	98.0	98.5	98.3	98.4	98.6

TABLE 4.3: Percentage of Steiner points pruned by each pruning test for different number of terminals for $k = 2$.

Number of Terminals	5	10	15	20	25	30	35	40
Lune (%)	49.3	90.2	95.7	97.1	97.7	97.9	98.2	98.9
Acute Angle (%)	55.0	86.3	90.7	92.0	93.5	93.9	93.4	94.5
Upper Bound (%)	11.8	23.6	30.0	32.5	34.0	37.5	35.5	37.3
Cross Lune (%)	8.1	2.8	1.3	0.7	0.5	0.4	0.3	0.2
All Tests (%)	**71.2**	**96.2**	**98.5**	**99.3**	**99.6**	**99.6**	**99.7**	**99.8**

TABLE 4.4: Percentage of Steiner points pruned by each pruning test for different number of terminals for $k = 3$.

To conclude, our algorithm can handle up to $n = 15$ for $k = 3$, with 97.56% of the total runtime spent in the ILP phase. When running only the generation phase, we can handle up to 55 terminals for $k = 2$ and 40 terminals for $k = 3$. The pruning tests

are highly effective, removing 99.8% of the generated Steiner points for $n = 40$, $k = 3$. Strong pruning tests are essential, as the ILP runtime appears to grow exponentially with the number of input Steiner points.

Chapter 5

Conclusion

In this thesis, we have presented the first exact algorithms capable of solving the minimum k-Steiner tree problem and the minimum bottleneck k-Steiner network problem in practice.

5.1 Summary of contributions

Chapters 2 and 3 have addressed the minimum k-Steiner tree problem, with Chapter 2 focusing on the geometric and structural properties and Chapter 3 on our algorithm. The minimum bottleneck k-Steiner network problem is explored in Chapter 4.

In Chapter 2, we have established numerous novel properties of a minimum k-Steiner tree, including the edge-lune theorem (Theorem 2.2.2), the 4-lune theorem (Theorem 2.2.4), the Rhombus Theorem (Theorem 2.2.7) and the Trapezium Theorem (Theorem 2.2.8). These theorems focus on degree-4 Steiner points, as they are unique to our problem, where the number of Steiner points is restricted to k.

In Chapter 3, we have presented our exact algorithm for the minimum k-Steiner tree problem, detailing our novel algorithms for both the generation phase and the concatenation phase. We have provided detailed descriptions of the pruning tests, the majority of which have been incorporated into our algorithm, including the projection test, the bottleneck test, the lune test, the 4-lune test and the Trapezium and Rhombus Test. Experimental results have been included to demonstrate the performance of the algorithm and to show the effectiveness of pruning tests.

In Chapter 4, we have presented our exact algorithm for the minimum bottleneck k-Steiner network problem. We have given detailed descriptions of both the generation phase and the ILP phase. Computational results have been provided to demonstrate the algorithm performance and to show the importance of pruning tests.

5.2 Future directions

This thesis provides a solid algorithmic framework for exact algorithms that solve the minimum k-Steiner tree problem and the minimum bottleneck k-Steiner network problem. However, we believe that there are still various ways in which the algorithms could be further improved.

The main difficulty with both problems arises from the proliferation of topologies during the generation of potential solutions. Hence, we implement pruning tests to help mitigate this issue.

For the minimum k-Steiner tree problem, we have established the edge-lune theorem (Theorem 2.2.2) and the 4-lune theorem (Theorem 2.2.4). However, we have not implemented pruning tests based on these theorems (a brief explanation was provided for the 4-lune test at the end of Subsection 3.3.4). These pruning tests could provide additional pruning that may help accelerate our algorithm.

Furthermore, after considerable deliberation and testing, we conjecture the following:

Conjecture 1. Let T be a minimum k-Steiner tree. Then, no two degree-4 Steiner points in T are adjacent.

We believe that this conjecture is not only powerful but also easy to implement, since simply preventing any branch that was previously triple-merged from being triple-merged again eliminates the possibility of adjacent degree-4 Steiner points arising. This prunes potential branches and FSTs very efficiently, without the need to verify the Steiner curves. The conjecture is not yet proven, but we have supporting evidence and a potential approach for proving it (discussed in detail in Appendix A).

In our algorithm for the minimum bottleneck k-Steiner network problem, each pruning test is conducted on cluster subtrees after a successful termination merge and embedding. However, our algorithm for the minimum k-Steiner tree problem employs pruning

tests on branches during every merge, even prior to termination. This provides more aggressive pruning by eliminating potential FSTs earlier on. The main goal of the bottleneck section of this thesis was to develop an exact algorithm that runs effectively in practice. We have not incorporated dynamic pruning tests in our algorithm due to the complexity and challenges involved in implementing pruning tests dynamically during the generation phase. Nevertheless, we firmly believe that the implementation of dynamic pruning tests is a logical next step in further improving our algorithm.

In our current bottleneck k-Steiner algorithm, we use leading regions to ensure that the length of each edge does not exceed the upper bound. Additionally, edge lengths may be constrained to be equal during merges. This results in a reduction in the size of leading regions, thereby decreasing the potential number of merges in the future. However, this was omitted from the thesis due to its complexity and difficulty with implementation.

Finally, the bottleneck k-Steiner algorithm spends the majority of its time in the ILP phase. Our aim was to develop a functional ILP formulation capable of identifying an optimal subset of Steiner points, and we successfully developed two such formulations. However, we believe that there may be a better formulation or ILP cuts specific to this problem that may greatly improve the speed of the ILP phase.

Appendix A

Adjacent degree-4 Steiner points conjecture

The content of this appendix addresses the minimum k-Steiner tree problem. It provides a comprehensive overview on all of our preliminary results on the adjacent degree-4 conjecture (Conjecture 1) so far.

First, we restate our conjecture here.

Conjecture 1 (Restated)**.** Let T be a minimum k-Steiner tree. Then, no two degree-4 Steiner points in T are adjacent.

Here, we discuss the potential advantages of this conjecture, if proven.

Preventing adjacent degree-4 Steiner points from being generated in the algorithm is quite straightforward. Suppose three branches, say $\mathcal{B}_1$, $\mathcal{B}_2$ and $\mathcal{B}_3$, are considered for a triple-merge. For each branch, we must verify whether the most recent merge was a triple-merge, i.e., whether the root is a degree-4 Steiner point. For example, suppose that $\mathcal{B}_1$ has a degree-4 Steiner point as its root s_1. Then, if the merge proceeds, the new degree-4 Steiner point s adjacent to s_1 is generated. This is shown below in Figure A.1.

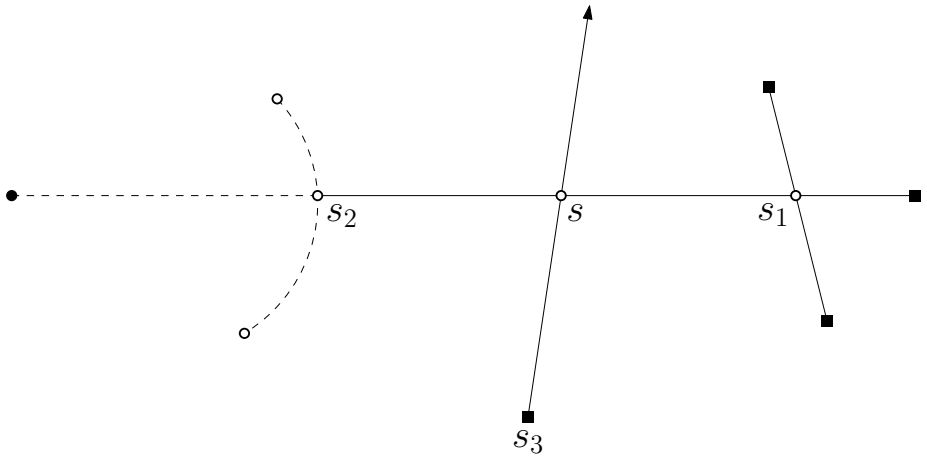


FIGURE A.1: The newly generated degree-4 Steiner point s is adjacent to another degree-4 Steiner point s_1.

This process only requires information about the roots of the branches that are considered for a triple-merge. Hence, if any of the roots are a degree-4 Steiner point, then we disregard the merge before it (and hence also the projection test) occurs. Therefore, if the conjecture holds, we can derive a pruning test that can eliminate a potential triple-merge rapidly and early on before the other pruning tests.

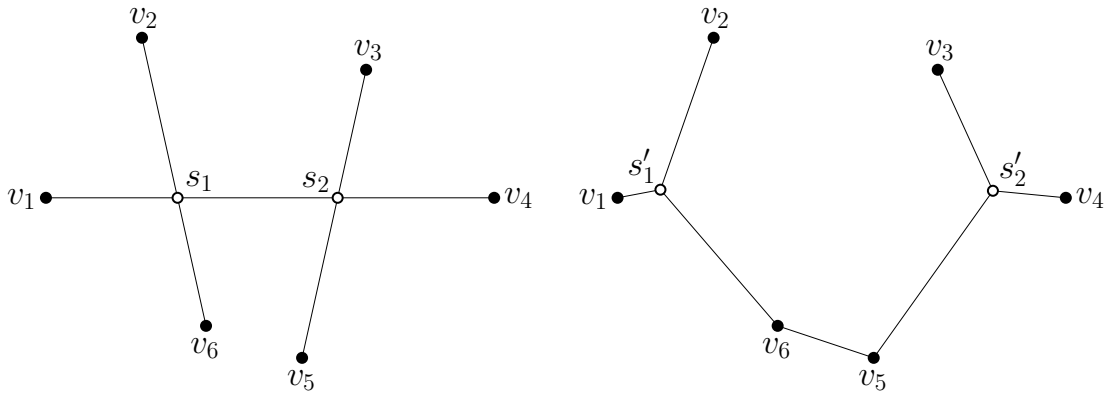
Now, we present an initial argument for Conjecture 1.

First, we generate many random instances where the tree that contains two adjacent degree-4 Steiner points is non-degenerate. Then, we aim to show that there always exists a tree that spans the same set of vertices with fewer or equal number of Steiner points with a shorter length.

Let $V = \{v_1, v_2, ..., v_6\}$ be a set of vertices. We let T_{44} be a 2-Steiner tree spanning V that contains two adjacent degree-4 Steiner points s_1 and s_2. As shown below in Figure A.2(a), the neighbours of s_1 are v_1, v_2, s_2 and v_6 (ordered in a clockwise manner) and the neighbours of s_2 are s_1, v_3, v_4 and v_5 (also ordered in a clockwise manner). We aim to show that for any location of the vertices in V, that there exists a shorter tree spanning V with at most two Steiner points.

Without loss of generality, let $v_1 = (0, 0)$ and $v_4 = (1, 0)$. Now, we deliberately place the other vertices to increase the likelihood of T_{44} being non-degenerate. Vertices v_2 and v_3 are placed above the line $\overline{v_1 v_4}$ and vertices v_5 and v_6 below the line such that $v_2 v_6$ and $v_3 v_5$ intersect $v_1 v_4$ at s_1 and s_2 respectively (where s_1 lies closer to v_1 than s_2). Since

the degree-4 Steiner points must be incident to two pairs of collinear edges, edges v_1s_1, s_1s_2 and s_2v_4 are collinear.



(a) A 44-Topology spanning the terminals $t_1, t_2, t_3, t_4, t_5, t_6$.

(b) A 33-Topology T'' on the same set of terminals.

FIGURE A.2

Now, we aim to generate a shorter tree spanning the vertices in V with at most two Steiner points. The second tree, denoted T_{33}, is the graph union of a minimum Steiner tree spanning $V_1 = \{v_1, v_2, v_6\}$, a minimum Steiner tree spanning $V_2 = \{v_3, v_4, v_5\}$ and a shortest edge with one end-vertex in V_1 and the other in V_2. Note that a minimum Steiner tree spanning three vertices does not contain more than a single Steiner point. Hence, T_{33} spans V and contains at most two Steiner points.

By the Rhombus Theorem, v_2 must lie inside $\triangle v_1s_2e_{v_1s_2}$. Since $s_2 \in v_1v_4$, $\triangle v_1s_2e_{v_1s_2} \subseteq \triangle v_1v_4e_{v_1v_4}$. Hence, if $v_2 \notin \triangle v_1v_4e_{v_1v_4}$, then the Rhombus Theorem is not satisfied and T_{44} is not an optimal 2-Steiner tree spanning V. By symmetry, we randomly place v_2, v_3 uniformly in $\triangle v_1v_4e_{v_1v_4}$ and v_5, v_6 in $\triangle v_4v_1e_{v_4v_1}$.

Out of a million instances where T_{44} was non-degenerate (where $|s_1v_1| < |s_2v_1|$ and $v2v_6$ does not intersect v_3v_5), $|T_{44}| < |T_{33}|$ occurred in 7.2456% of the instances.

In particular, we focused on the instances where the Rhombus Theorem was satisfied for all eight vertices (four neighbours of s_1 and four neighbours of s_2). In all $10,000$ instances where the Rhombus Theorem was satisfied, $|T_{44}| \geq |T_{33}|$. Hence, we conjecture the following result.

Conjecture 2. Let T_{44} and T_{33} be defined as above. Then, if the Rhombus Theorem is satisfied for all eight neighbours of the two degree-4 Steiner points in T_{44}, then $|T_{44}| \geq |T_{33}|$.

If the Rhombus Theorem is not satisfied for any of the neighbours of a degree-4 Steiner point, say s_1, then there exists a shorter 1-Steiner tree spanning the neighbours of s_1 and T_{44} cannot be optimal as a result. Therefore, proving Conjecture 2 implies that Conjecture 1 is true.

To verify its potential usefulness, we ran our algorithm with and without the adjacent degree-4 Steiner points pruning test for instances with $k = 3$. The decrease in the algorithm runtime is shown below in Figure A.3.

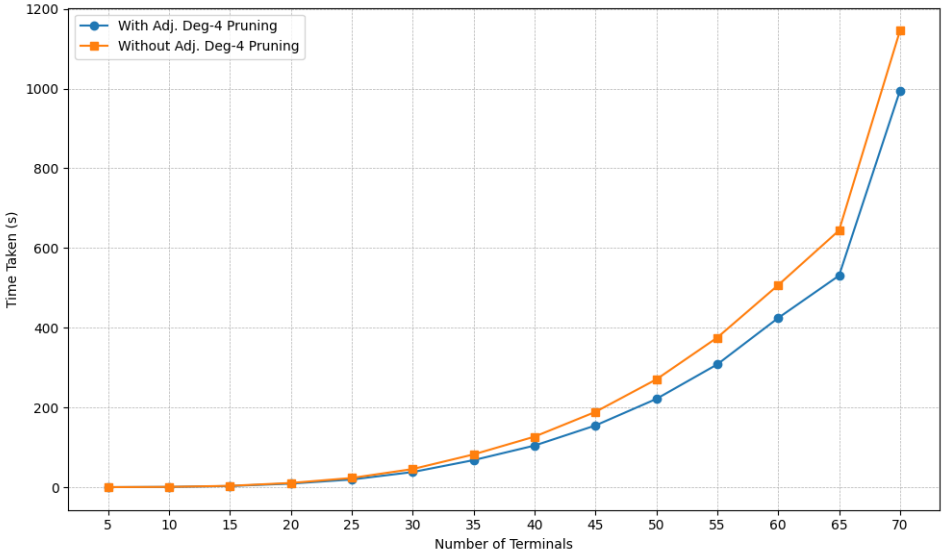


FIGURE A.3: The improvement in time by including the adjacent degree-4 pruning test for $k = 3$.

Bibliography

[1] Marcus Brazil, Michael Hendriksen, Jae Lee, Michael S Payne, Charl Ras, and Doreen Anne Thomas. An exact algorithm for the Euclidean k-Steiner tree problem. *Computational Geometry*, 121:102099, 2024.

[2] Marcus Brazil, Ronald L Graham, Doreen A Thomas, and Martin Zachariasen. On the history of the Euclidean Steiner tree problem. *Archive for history of exact sciences*, 68:327–354, 2014.

[3] Michael R Garey, Ronald L Graham, and David S Johnson. The complexity of computing Steiner minimal trees. *SIAM journal on applied mathematics*, 32(4): 835–859, 1977.

[4] Sookyoung Lee and Mohamed Younis. Recovery from multiple simultaneous failures in wireless sensor networks using minimum Steiner tree. *Journal of Parallel and Distributed Computing*, 70(5):525–536, 2010.

[5] Doreen A Thomas, Marcus Brazil, David H Lee, and Nicholas C Wormald. Network modelling of underground mine layout: two case studies. *International Transactions in Operational Research*, 14(2):143–158, 2007.

[6] J-M Ho, Gopalakrishnan Vijayan, and Chak-Kuen Wong. New algorithms for the rectilinear Steiner tree problem. *IEEE transactions on computer-aided design of integrated circuits and systems*, 9(2):185–193, 1990.

[7] Hao Tang, Genggeng Liu, Xiaohua Chen, and Naixue Xiong. A survey on Steiner tree construction and global routing for VLSI design. *IEEE Access*, 8:68593–68622, 2020.

[8] KG Sirinanda, Marcus Brazil, Peter Alexander Grossman, J Hyam Rubinstein, and Doreen A Thomas. Gradient-constrained discounted Steiner trees II: optimally

locating a discounted Steiner point. *Journal of Global Optimization*, 64(3):515–532, 2016.

[9] George Georgakopoulos and Christos H Papadimitriou. The 1-Steiner tree problem. *Journal of Algorithms*, 8(1):122–130, 1987.

[10] Marcus Brazil, Charl J Ras, Konrad J Swanepoel, and Doreen A Thomas. Generalised k-Steiner tree problems in normed planes. *Algorithmica*, 71(1):66–86, 2015.

[11] Prosenjit Bose, Anthony D'Angelo, and Stephane Durocher. On the restricted k-Steiner tree problem. *Journal of Combinatorial Optimization*, 44(4):2893–2918, 2022.

[12] Charles C Chiang, Majid Sarrafzadeh, and Chak-Kuen Wong. A powerful global router: based on Steiner min-max trees. In *ICCAD*, pages 2–5. Citeseer, 1989.

[13] Marcus Brazil and Martin Zachariasen. Optimal interconnection trees in the plane. *Algorithms and Combinatorics*, 29, 2015.

[14] Andrew B Kahng and Gabriel Robins. A new class of iterative Steiner tree heuristics with good performance. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(7):893–902, 1992.

[15] Wang and Z Du. Approximations for a bottleneck Steiner tree problem. *Algorithmica*, 32:554–561, 2002.

[16] Sang Won Bae, Chunseok Lee, and Sunghee Choi. On exact solutions to the Euclidean bottleneck Steiner tree problem. In *International Workshop on Algorithms and Computation*, pages 105–116. Springer, 2009.

[17] Dingzhu Du, Lusheng Wang, and Baogang Xu. The Euclidean bottleneck Steiner tree and Steiner tree with minimum number of Steiner points. In *International Computing and Combinatorics Conference*, pages 509–518. Springer, 2001.

[18] Sang Won Bae, Sunghee Choi, Chunseok Lee, and Shin-ichi Tanigawa. Exact algorithms for the bottleneck Steiner tree problem. *Algorithmica*, 61:924–948, 2011.

[19] Sayan Bandyapadhyay, William Lochet, Daniel Lokshtanov, Saket Saurabh, and Jie Xue. Euclidean bottleneck Steiner tree is fixed-parameter tractable. In *Proceedings of the 2024 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 699–711. SIAM, 2024.

[20] Marcus Brazil, Charl J Ras, and Doreen A Thomas. The bottleneck 2-connected k-Steiner network problem for $k \leq 2$. *Discrete Applied Mathematics*, 160(7-8): 1028–1038, 2012.

[21] Charl J Ras. Fixed parameter tractability of a biconnected bottleneck steiner network problem. *Networks*, 75(3):310–320, 2020.

[22] Pawel Winter. An algorithm for the Steiner problem in the Euclidean plane. *Networks*, 15(3):323–345, 1985.

[23] David M Warme. Quantitative indicators for strength of inequalities with respect to a polyhedron, part II: Applications and computational evidence. *arXiv preprint arXiv:2403.14540*, 2024.

[24] J Hyam Rubinstein, Doreen A Thomas, and Jia F Weng. Degree-five Steiner points cannot reduce network costs for planar sets. *Networks*, 22(6):531–537, 1992.

[25] Evangelista Torricelli, G Loria, and G Vassura. De maximis et minimis. *Opere di Evangelista Torricelli*, 1:79–97, 1919.

[26] Gabriel Robins and Jeffrey S Salowe. On the maximum degree of minimum spanning trees. In *Proceedings of the tenth annual symposium on Computational geometry*, pages 250–258, 1994.

[27] Marcus Brazil, Marcus Volz, Martin Zachariasen, Charl Ras, and Doreen Thomas. New pruning rules for the Steiner tree problem and 2-connected Steiner network problem. *Computational Geometry*, 78:37–49, 2019.

[28] Jon Lee. *A first course in combinatorial optimization*, volume 36. Cambridge University Press, 2004.

[29] Edgar N Gilbert and Henry O Pollak. Steiner minimal trees. *SIAM Journal on Applied Mathematics*, 16(1):1–29, 1968.

[30] David Michael Warme. *Spanning trees in hypergraphs with applications to Steiner trees*. University of Virginia, 1998.

[31] Ulrich Pferschy and Rostislav Staněk. Generating subtour elimination constraints for the TSP from pure integer solutions. *Central European Journal of Operations Research*, 25(1):231–260, 2017.

[32] Emo Welzl. Smallest enclosing disks (balls and ellipsoids). In *New Results and New Trends in Computer Science: Graz, Austria, June 20–21, 1991 Proceedings*, pages 359–370. Springer, 2005.