Analysing trajectory similarity and improving graph dilation

Sampson Wong

School of Computer Science, University of Sydney

A thesis submitted to fulfil requirements for the degree of Doctor of Philosophy

Abstract

In this thesis, we focus on two topics in computational geometry.

The first topic is analysing trajectory similarity. A trajectory tracks the movement of an object over time. A common way to analyse trajectories is by finding similarities. The Fréchet distance is a similarity measure that has gained popularity in the theory community, since it takes the continuity of the curves into account. One way to analyse trajectories using the Fréchet distance is to cluster trajectories into groups of similar trajectories. For vehicle trajectories, another way to analyse trajectories is to compute the path on the underlying road network that best represents the trajectory. In Chapters 2, 3 and 4, we focus on trajectory similarity problems.

The second topic is improving graph dilation. Dilation measures the quality of a network in applications such as transportation and communication networks. Spanners are low dilation graphs with not too many edges. Most of the literature on spanners focuses on building the graph from scratch. We instead focus on adding edges to improve the dilation of an existing graph. In Chapters 5 and 6, we focus on graph dilation problems.

Statement of Attribution

The contents of this thesis are based on four published papers and one unpublished manuscript. I was the corresponding author and the main contributor on all five papers. As per convention in theoretical computer science, authors are listed alphabetically.

Chapter 2. Joachim Gudmundsson, Sampson Wong. *Cubic upper and lower bounds for subtrajectory clustering under the continuous Fréchet distance.* In Proceedings of the 33rd Symposium on Discrete Algorithms, SODA 2022.

Chapter 3. Joachim Gudmundsson, Martin P. Seybold, Sampson Wong. *Map matching queries on realistic input graphs under the Fréchet distance.* In Proceedings of the 34th Symposium on Discrete Algorithms, SODA 2023.

Chapter 4. Kevin Buchin, André Nusser, Sampson Wong. *Computing continuous dynamic time warping of time series in polynomial time.* In Proceedings of the 38th Symposium on Computational Geometry, SoCG 2022.

Chapter 5. Joachim Gudmundsson, Sampson Wong. *Improving the dilation of metric graphs by adding edges.* In Proceedings of the 32nd Symposium on Discrete Algorithms, SODA 2021. Journal version in Transactions on Algorithms, TALG 2022.

Chapter 6. Kevin Buchin, Maike Buchin, Joachim Gudmundsson, Sampson Wong. *Bicriteria approximation for minimum dilation graph augmentation.* Under submission.

As supervisor for the candidature upon which this thesis is based, I can confirm that the authorship attribution statements above are correct.

Joachim Gudmundsson

Statement of Originality

This is to certify that to the best of my knowledge, the content of this thesis is my own work. This thesis has not been submitted for any degree or other purposes.

I certify that the intellectual content of this thesis is the product of my own work and that all the assistance received in preparing this thesis and sources have been acknowledged.

Sampson Wong

Acknowledgements

This thesis would not have been possible without the help of many people.

First, I would like to thank my amazing supervisor — Joachim. I feel incredibly lucky to be your student.

Second, I would like to thank my colleagues and friends at Sydney University — André, John, Julian, Martin, Milutin, Patrick, Vikrant, William, Yuan; Adam, Alan, Baptiste, Chris, Clément, Lindsey, Niku, Sarah, Sasha, Zijin; Anushka, Helen, Kai, Li, Omid, Yuan. Thank you for creating such a positive atmosphere at the university.

Third, I would like to thank my collaborators — Mark de Berg, Kevin Buchin, Mees van de Kerkhof, Koen Klaren, Aleksandr Popov, Zeinab Saedi, Frank Staals, Lionov Wiratma; Mikkel Abrahamsen, Maike Buchin, Ivor van der Hoog, Antonia Kalb, André Nusser, Lukas Plaetz, Eva Rotenberg, Yucheng Sun, Lea Thiel, Hanwen Zhang. Thank you for the enriching research discussions, and I hope to work together soon.

Fourth, I would like to thank my family — 爸爸, 媽咪, 家姐. Thank you for always being there for me, and for always believing in me.

Finally, I would like to thank my wonderful wife — Rachel. I love you.

Contents

Chapter 1

Introduction

1.1 Computational geometry

Computer science is the study of computers and computation. It encompassed a wide range of fields, including:

- Computer systems. This field focuses on the design, implementation, and management of hardware, software, and networks. Subfields include computer architecture, operating systems, computer engineering, parallel computing, concurrency, distributed computing, computer networking, computer security, cryptography, blockchains, databases and data mining.
- Applied computer science, which focuses on practical applications of computer science to solve real-world problems. Subfields include computer graphics, information processing, visualisation, computational science, human computer interaction, software engineering, programming languages, compilers, artificial intelligence and machine learning.
- Theoretical computer science focuses on the mathematical foundations of computer science. Subfields include automata theory, computability theory, complexity theory, quantum computing, information theory, coding theory, algorithms, data structures, programming languages and formal methods.

Computational geometry is a subfield of theoretical computer science that focuses on problems that can be stated geometrically. Geometric problems are commonly set in the Euclidean plane (alternatively, the Cartesian plane or the xy-plane). Geometric problems may also be set in higher dimensions.

Computational geometry overlaps with other subfields of theoretical computer science. In particular, there is significant overlap with:

- Algorithms. Given an input, perform a sequence of steps to produce the desired output. An algorithm is more efficient if it requires fewer steps. In theoretical computer science, an algorithm's efficiency is described mathematically. In particular, an algorithms running time is often stated in Big-O notation. Examples of algorithms in computational geometry include convex hull algorithms and sweep line algorithms.

- Data structures. Given input data, perform processing steps to construct a data structure. Given a query and the data structure, perform query steps to produce the desired output. The quality of a data structure depends on: the efficiency of the preprocessing algorithm, the efficiency of the query algorithm, and the size of the data structure. Examples of data structures in computational geometry include range searching data structures and point location data structures.

- Complexity theory. Given an input, how many steps are required to produce the desired output? The complexity of a problem is the efficiency of the best algorithm for solving it. A problem's complexity is often stated in terms of its complexity class. Examples of complexity classes in computational geometry include NP-hard problems and 3SUM-hard problems.

In this thesis, we focus on two topics in computational geometry. The first topic is analysing trajectory similarity, which we introduce in Section 1.2. The second topic is improving graph dilation, which we introduce in Section 1.3.

1.2 Trajectory similarity

The widespread use of the Global Positioning System (GPS) in location aware devices has led to an abundance of trajectory data. Although collecting and storing this data is cheaper and easier than ever, this rapid increase of data is making the problem of analysing this data more demanding.

A GPS trajectory tracks the movement of an object over time. We model a trajectory as a polygonal curve in the Euclidean plane. A common way to analyse trajectories is by finding similarities. Computing similarities is a fundamental building block for other analyses, such as clustering, classification, or simplification. There are numerous similarity measures considered in literature [16, 58, 80, 116, 151, 155], many of which are application dependent.

The Fréchet distance is a similarity measure that has gained popularity, especially in the theory community [12, 37, 69, 148]. Under the Fréchet distance, a minimum cost continuous alignment is computed between the pair of trajectories. A continuous alignment is a simultaneous traversal of the pair of trajectories that satisfies four conditions: *(i)* the first pair is the first point from both trajectories, *(ii)* the last pair is the last point from both trajectories, *(iii)* each point must appear in some pair in the alignment, and *(iv)* the alignment must be a monotonically increasing sequence for both trajectories. The cost of a continuous alignment, under the Fréchet distance, is the maximum distance between a pair of points in the alignment.

Alt and Godau [12] were the first to study algorithms for computing the Fréchet distance. For a pair of trajectories of complexity n, they provide an $O(n^2 \log n)$ time algorithm. Later, Buchin et al. [37] provide an improved $O(n^2\sqrt{\log n}(\log\log n)^{3/2})$ time algorithm.

Bringmann [28] was the first to apply fine-grained lower bounds, conditioned on the Strong Exponential Time Hypothesis (SETH), to Fréchet distance problems. Bringmann [28] showed a quadratic lower bound for computing the Fréchet distance between trajectories of dimension two or higher. Buchin et al. [42] extended the results to hold for the Fréchet distance of 1-dimensional trajectories.

One way to analyse trajectories using the Fréchet distance is to cluster trajectories into groups of similar trajectories. The (k, ℓ)-center and (k, ℓ)-median clustering problems

are two such problems that have received attention. Given a set $\mathcal{G}$ of trajectories, and parameters k and ℓ, the problem is to find a set $\mathcal{C}$ of k trajectories (not necessarily in $\mathcal{G}$), each of complexity at most ℓ, so that the maximum Fréchet distance (center) or the sum of the Fréchet distances (median) over all trajectories in $\mathcal{G}$ to its closest trajectory in $\mathcal{C}$ is minimised. The set $\mathcal{C}$ is also known as the set of center curves, and the intuition behind restricting the complexity of the center curves is to avoid overfitting.

Driemel et al. [69] were the first to consider (k, ℓ)-center and (k, ℓ)-medians clustering of trajectories. They showed that both problems are NP-hard when k is part of the input, and provided $(1 + \varepsilon)$-approximation algorithms if the trajectories are one-dimensional. Buchin et al. [40] showed that (k, ℓ)-center clustering is NP-hard if ℓ is part of the input, and provided a 3-approximation algorithm for trajectories of any dimension. Buchin et al. [44] provided a randomised bicriteria-approximation algorithm with approximation factor $(1+\varepsilon)$ for trajectories of any dimension.

For vehicle trajectories, another way to analyse trajectories is to compute the path on the underlying road network that best represents the trajectory. Alt et al. [11] study this problem on geometric planar graphs. They provide an $O(pq \log p)$ time algorithm, where p is the complexity of the graph and q is the complexity of the trajectory. Alt et al. [11]'s algorithm forms the basis of several existing implementations [25, 53, 145, 161, 163]. Brakatsoulas et al. [25] implement Alt et al. [11]'s algorithm and experimentally compare it to a linear-time heuristic and an algorithm minimising the weak Fréchet distance. In their experiments, forty-five vehicle trajectories, each with approximately one hundred edges, are mapped onto an underlying road network with approximately ten thousand edges. Their experiments conclude that out of the three algorithms, Alt et al. [11]'s provides the best results but is the slowest. Subsequent papers focus on improving the practical running time of the algorithm [145, 163].

1.3 Graph dilation

Let $G = (V, E)$ be a graph embedded in a metric space M. For every pair of points $u, v \in V$, the weight of the edge (u, v) is equal to the distance $d_M(u, v)$ between points u and v in the metric space M. Let $d_G(u, v)$ be the weight of the shortest path between u and v in the graph G. For any real number $t > 1$, we call G a *t-spanner* if $d_G(u, v) \leq t \cdot d_M(u, v)$ for every pair of points $u, v \in V$. The *stretch*, or *dilation*, of G is the smallest t for which G is a t-spanner.

Spanners have been studied extensively in the literature, especially in the geometric setting. Given a fixed $t > 1$, a fixed dimension $d \geq 1$, and a set of n points V in d-dimensional Euclidean space, there is a t-spanner on the point set V with $O(n)$ edges. For a summary of the considerable research on geometric spanners, see the surveys [77, 98, 147] and the book by Narasimhan and Smid [133]. Spanners in doubling metrics [50, 95, 110] and in general graphs [19, 135, 154] have also received considerable attention.

Given a set of points V, a spanning tree of V with minimum dilation is known as a minimum dilation spanning tree [15, 26, 56], a tree spanner [47, 82, 87] or a minimum maximum-stretch spanning tree [76, 121, 136]. Computing a minimum dilation spanning tree is NP-hard even if M is an unweighted graph metric [47] or the Euclidean plane [56]. The minimum dilation spanning tree problem is closely related to tree embeddings of general metrics [17], and has applications to communication networks and distributed systems [136].

The approximability of the minimum dilation spanning tree problem is an open problem

stated in surveys and papers [56, 77, 136], and is a major obstacle towards constructing low dilation graphs with few edges [15, 108]. The minimum spanning tree is an $O(n)$-approximation [77], but no better result is known. Only in the special case where M is an unweighted graph is there an $O(\log n)$-approximation [76].

1.4 Contributions

In this section, we summarise the main contributions in this thesis. Chapters 2, 3 and 4 contribute to analysing trajectory similarity. Chapters 5 and 6 contribute to improving graph dilation.

Chapter 2: Subtrajectory clustering

Detecting commuting patterns or migration patterns in movement data is an important problem in computational movement analysis. Given a trajectory, or set of trajectories, this corresponds to clustering similar subtrajectories.

We study subtrajectory clustering under the continuous and discrete Fréchet distances. The most relevant theoretical result is by Buchin et al. (2011). They provide, in the continuous case, an $O(n^5)$ time algorithm and a 3SUM-hardness lower bound, and in the discrete case, an $O(n^3)$ time algorithm. We show, in the continuous case, an $O(n^3 \log^2 n)$ time algorithm and a 3OV-hardness lower bound, and in the discrete case, an $O(n^2 \log n)$ time algorithm and a quadratic lower bound. Our bounds are almost tight unless SETH fails.

Chapter 3: Map matching queries

Map matching is a common preprocessing step for analysing vehicle trajectories. In the theory community, the most popular approach for map matching is to compute a path on the road network that is the most spatially similar to the trajectory, where spatial similarity is measured using the Fréchet distance. A shortcoming of existing map matching algorithms under the Fréchet distance is that every time a trajectory is matched, the entire road network needs to be reprocessed from scratch. An open problem is whether one can preprocess the road network into a data structure, so that map matching queries can be answered in sublinear time. We investigate map matching queries under the Fréchet distance. We provide a negative result for geometric planar graphs, and a positive result for realistic input graphs. We show that for c-packed graphs, one can construct a data structure of $\tilde{O}(cp)$ size that can answer $(1+\varepsilon)$-approximate map matching queries in $\tilde{O}(c^4 q \log^4 p)$ time, where $\tilde{O}(\cdot)$ hides lower-order factors and dependence on ε.

Chapter 4: Continuous dynamic time warping

Dynamic Time Warping is a popular similarity measure for time series, where we define a time series to be a one-dimensional polygonal curve. The drawback of Dynamic Time Warping is that it is sensitive to the sampling rate of the time series. The Fréchet distance is an alternative that has gained popularity, however, its drawback is that it is sensitive to outliers. Continuous Dynamic Time Warping (CDTW) is a recently proposed alternative that does not exhibit the aforementioned drawbacks. CDTW combines the continuous nature of the Fréchet distance with the summation of Dynamic Time Warping, resulting in a similarity measure that is robust to sampling rate and to outliers. Despite its advantages,

the major shortcoming of CDTW is that there is no exact algorithm for computing CDTW in polynomial time. We present the first exact algorithm for computing CDTW of one-dimensional curves. Our algorithm runs in time $O(n^5)$ for a pair of one-dimensional curves, each with complexity at most n.

Chapters 5 and 6: Improving graph dilation by adding edges

Most of the literature on spanners focuses on building the graph from scratch. We instead focus on adding edges to improve an existing graph. A major open problem in this field is: given a graph embedded in a metric space, and a budget of k edges, which k edges do we add to produce a minimum-dilation graph? The special case where $k = 1$ has been studied in the past, but no major breakthroughs have been made for $k > 1$. We provide the first positive result, an $O(k)$-approximation algorithm that runs in $O(n^3 \log n)$ time. We also provide a $(2\sqrt[r]{2}\ k^{1/r}, 2r)$-bicriteria approximation that runs in $O(n^3 \log n)$ time, for all $r \geq 1$. In other words, if t^* is the minimum dilation after adding any k edges to a graph, then our algorithm adds $O(k^{1+1/r})$ edges to the graph to obtain a dilation of $2rt^*$.

1.5 Other projects during PhD

In this section, we summarise three other projects that the author was involved in. These projects will not be discussed further in the rest of this thesis.

Covering a set of line segments with a few squares

We study three covering problems in the plane. The first is to decide whether a given set of line segments can be covered by up to $k = 4$ unit-sized, axis-parallel squares. We give linear time algorithms for $k \leq 3$ and an $O(n \log n)$ time algorithm for $k = 4$. The second is to build a data structure on a trajectory to efficiently answer whether any query subtrajectory is coverable by up to three unit-sized axis-parallel squares. For $k = 2$ and $k = 3$ we construct data structures of size $O(n\alpha(n) \log n)$, in $O(n\alpha(n) \log n)$ time, so that we can test if an arbitrary subtrajectory can be k-covered in $O(\log n)$ time. The third problem is to compute a longest subtrajectory of a given trajectory that can be covered by up to two unit-sized axis-parallel squares. We give $O(n2^{\alpha(n)} \log^2 n)$ time algorithms for $k \leq 2$.

This work is joint with Joachim Gudmundsson, Mees van de Kerkhof, André van Renssen, Frank Staals, Lionov Wiratma. The conference version appeared in Proceedings of the 12th International Conference on Algorithms and Complexity, CIAC 2021. The journal version appeared in Theoretical Computer Science, TCS 2022.

Approximating the packedness of polygonal curves

Driemel, Har-Peled and Wenk [68] introduced the concept of c-packed curves as a realistic input model. In the case when c is a constant they gave a near linear time $(1+\varepsilon)$-approximation algorithm for computing the Fréchet distance between two c-packed polygonal curves. Since then a number of papers have used the model. We consider the problem of computing the smallest c for which a given polygonal curve in $\mathbb{R}^d$ is c-packed. We present two approximation algorithms. The first algorithm is a 2-approximation algorithm and runs in $O(dn^2 \log n)$ time. In the case $d = 2$ we develop a faster algorithm that returns a $(6 + \varepsilon)$-approximation and runs in $O((n/\varepsilon^3)^{4/3} \operatorname{poly} \log(n/\varepsilon))$ time.

This work is joint with Joachim Gudmundsson, Yuan Sha. The conference version appeared in Proceedings of the 31st International Symposium on Algorithms and Computation, ISAAC 2020. The journal version appeared in Computational Geometry, CGTA 2023.

(k, ℓ)-medians clustering of trajectories

We consider the problem of center-based clustering of trajectories. In this setting, the representative of a cluster is again a trajectory. To obtain a compact representation of the clusters and to avoid overfitting, we restrict the complexity of the representative trajectories by a parameter ℓ. This restriction, however, makes discrete distance measures like dynamic time warping (DTW) less suited. While the Fréchet distance allows for restriction of the center complexity, it can also be sensitive to outliers. To obtain a trajectory clustering algorithm that allows restricting center complexity and is more robust to outliers, we propose the usage of a continuous version of DTW as distance measure, which we call continuous dynamic time warping (CDTW). Our contribution is twofold. First, to combat the lack of practical algorithms for CDTW, we develop an approximation algorithm that computes it. Second, we develop the first clustering algorithm under this distance measure and show a practical way to compute a center from a set of trajectories and iteratively improve it.

This work is joint with Milutin Brankovic, Kevin Buchin, Koen Klaren, André Nusser, Aleksandr Popov. The conference version appeared in Proceedings of the 28th International Conference on Advances in Geographic Information Systems, SIGSPATIAL 2020.

Chapter 2

Cubic upper and lower bounds for subtrajectory clustering under the continuous Fréchet distance

2.1 Introduction

The widespread use of the Global Positioning System (GPS) in location aware devices has led to an abundance of trajectory data. Although collecting and storing this data is cheaper and easier than ever, this rapid increase of data is making the problem of analysing this data more demanding. One way of extracting useful information from a large trajectory data set is to cluster the trajectories into groups of similar trajectories. However, focusing on clustering entire trajectories can overlook significant patterns that exist only for a small portion of their lifespan. Consequently, subtrajectory clustering is more appropriate if we are interested in similar portions of trajectories, rather than entire trajectories.

Subtrajectory clustering has been used to detect similar movement patterns in various applications. Gudmundsson and Wolle [107] applied it to football analysis. They reported common movements of the ball, common movements of football players, and correlations between players in the same team moving together. Buchin et al. [35] applied subtrajectory clustering to map reconstruction. They reconstructed the location of roads, turns and crossings from urban vehicle trajectories, and the location of hiking trails from hiking trajectories. Many other applications have been considered in the Data Mining and the Geographic Information Systems communities, including behavioural ecology, computational biology and traffic analysis [5, 34, 51, 104, 105, 120, 123, 149, 150].

Despite considerable attention across multiple communities, the theoretical aspects of subtrajectory clustering are not well understood. The most closely related result is by Buchin et al. [36]. Their algorithm forms the basis of several implementations [34, 35, 104, 105, 107]. Other models of subtrajectory clustering have also been considered [5, 9], which we will briefly discuss in our related work section.

To measure the similarity between subtrajectories, numerous distance measures have

been proposed in the literature [138, 151, 155]. In this chapter, we will use the (discrete and continuous) Fréchet distance, which is the most common and successful distance measure used for trajectories, and also the preferred distance measure in the theory community.

Given a trajectory T, the subtrajectory clustering (SC) problem considered by Buchin et al. [36] is to compute a subtrajectory cluster consisting of m non-overlapping subtrajectories of T, one of which is called the reference subtrajectory. The reference subtrajectory must have length at least ℓ, and the Fréchet distance between the reference subtrajectory and any of the other $m-1$ subtrajectories is at most d. We formally define the SC problem in Section 2.2. For SC under the continuous Fréchet distance, Buchin et al. [36] provide an $O(n^5)$ time algorithm and a 3SUM-hardness lower bound. For SC under the discrete Fréchet distance, they provide an $O(n^3)$ time algorithm. Closing the gaps between the two upper and lower bounds have remained important open problems.

In this chapter, we provide an $O(n^3 \log^2 n)$ time algorithm for SC under the continuous Fréchet distance, which is a significant improvement over the previous algorithm [36]. Along the way, we also show an $O(n^2 \log n)$ time algorithm for SC under the discrete Fréchet distance.

We argue that our algorithms are essentially optimal. Our lower bounds are conditional on the Strong Exponential Time Hypothesis (SETH). Our main technical contribution is an intricate 3OV-hardness lower bound for SC under the continuous Fréchet distance. This implies that there is no $O(n^{3-\varepsilon})$ time algorithm for any $\varepsilon > 0$, unless SETH fails. We also show, via a simple reduction, that Bringmann's [28] SETH-based quadratic lower bound applies to SC under the discrete Fréchet distance. These lower bounds show that our two algorithms are almost optimal, unless SETH fails.

Interestingly, our results show that there is a provable separation between the discrete and continuous Fréchet distance for SC. A similar separation between the discrete and continuous variants has been shown for computing the weak Fréchet distance between one-dimensional curves [42], for computing the Fréchet distance between two point sets [45], and for computing the Fréchet distance between uncertain curves with imprecise inputs [41].

Next, we outline the structure of the chapter. We discuss related work in Section 2.1.1 and preliminaries in Section 2.2. In Section 2.3.1, we provide an overview of the key insights that lead to our improved algorithms. In Section 2.3.2, we give a quadratic lower bound in the discrete case, and an overview of the key components of our cubic lower bound in the continuous case. Detailed descriptions and the full proofs are provided in Section 2.4 for our algorithm under the discrete Fréchet distance, in Section 2.5 for our algorithm under the continuous Fréchet distance, and in Section 2.6 for our 3OV reduction.

2.1.1 Related work

Recently, the closely related problem of clustering trajectories has received considerable attention, especially the (k, ℓ)-center and (k, ℓ)-median clustering problems. In these problems, entire trajectories are clustered. Given a set $\mathcal{G}$ of trajectories, and parameters k and ℓ, the problem is to find a set $\mathcal{C}$ of k trajectories (not necessarily in $\mathcal{G}$), each of complexity at most ℓ, so that the maximum Fréchet distance (center) or the sum of the Fréchet distances (median) over all trajectories in $\mathcal{G}$ to its closest trajectory in $\mathcal{C}$ is minimised. The set $\mathcal{C}$ is also known as the set of center curves, and the intuition behind restricting the complexity of the center curves is to avoid overfitting.

Driemel et al. [69] were the first to consider (k, ℓ)-center and (k, ℓ)-medians clustering of trajectories. They showed that both problems are NP-hard when k is part of the in-

put, and provided $(1+\varepsilon)$-approximation algorithms if the trajectories are one-dimensional. Buchin et al. [40] showed that (k,ℓ)-center clustering is NP-hard if ℓ is part of the input, and provided a 3-approximation algorithm for trajectories of any dimension. Buchin et al. [44] provided a randomised bicriteria-approximation algorithm with approximation factor $(1+\varepsilon)$ for trajectories of any dimension.

The idea of computing a set of center curves for trajectory clustering has been extended to subtrajectory clustering. Agarwal et al. [5] compute center curves (which they call pathlets) from a set of input trajectories. The key difference is that pathlets are similar to portions of the input trajectory, rather than the entire trajectories. Each trajectory is then modelled as a concatenation of pathlets, with possible gaps in between. Agarwal et al. [5] propose an objective function for finding the optimal set of pathlets that best describe the set of input trajectories, and they show that finding the optimum value of the objective function is NP-hard. They provide an $O(\log n)$-approximation algorithm that runs in polynomial time. Akitaya et al. [9] consider a similar model, except that they compute a set of center curves with complexity at most ℓ, and the concatenation of center curves covers the input trajectory without gaps. The objective is to compute the set of center curves of minimum size. Akitaya et al. [9] show that computing the optimal set of center curves is NP-hard, and provide a polynomial-time $O(\log n)$-approximation.

Bringmann [28] was the first to apply fine-grained lower bounds, conditioned on the Strong Exponential Time Hypothesis (SETH), to Fréchet distance problems. Bringmann [28] showed a quadratic lower bound for computing the (discrete or continuous) Fréchet distance between trajectories of dimension two or higher. Bringmann and Mulzer [33] extended the results to also hold for the discrete Fréchet distance of 1-dimensional trajectories. Buchin et al. [42] further extended the results to hold for the continuous Fréchet distance of 1-dimensional trajectories. Bringman et al. [32] showed a 4OV-hardness lower bound for computing the translation invariant discrete Fréchet distance between two trajectories of dimension two or higher.

2.2 Preliminaries

In this chapter, we study the problem of detecting a movement pattern that occurs frequently in a trajectory, or in a set of trajectories. The problem was first proposed by Buchin et al. [36]. We retain the existing convention by referring to this problem as the subtrajectory clustering (SC) problem. In the SC problem, a trajectory T of complexity n is defined to be a sequence of points $v_1, v_2, \ldots, v_n$ in the c-dimensional Euclidean space $\mathbb{R}^c$, connected by segments.

Problem 1 (SC problem)**.** *Given a trajectory T of complexity n, a positive integer m, and positive real values ℓ and d, decide if there exists a subtrajectory cluster of T such that:*

- *the cluster consists of one reference subtrajectory and $m-1$ other subtrajectories of T,*
- *the reference subtrajectory has Euclidean length at least ℓ,*
- *the Fréchet distance between the reference subtrajectory and any other subtrajectory is $\leq d$,*
- *any pair of subtrajectories in the cluster overlap in at most one point.*

Buchin et al. [36] show how the case where the input is a set of trajectories can be reduced to the case when a single trajectory is given as input. See Figure 2.1. Since our two

algorithms build upon the algorithm by Buchin et al. [36] we briefly describe their algorithm. First, we discuss their algorithm for the discrete Fréchet distance, then for the continuous.

Figure 2.1: A subtrajectory cluster, for a single trajectory (left), or for a set of trajectories (right).

The first step is to transform SC into a problem in the discrete free space diagram. Let $F_d(T,T)$ be the discrete Fréchet free space diagram between two copies of T and with distance value d. Suppose the conditions of SC hold for some reference subtrajectory starting at vertex s and ending at vertex t. Let l_s and l_t be the vertical lines in $F_d(T,T)$ representing s and t. The conditions of SC state that there are $m-1$ other subtrajectories so that the Fréchet distance between the reference subtrajectory and each of the other subtrajectories is at most d. Therefore, the SC problem reduces to deciding if there are vertical lines l_s and l_t so that the reference subtrajectory from s to t has length at least ℓ, and there are $m-1$ monotone paths in $F_d(T,T)$ starting at l_s and ending at l_t. Moreover, since these $m-1$ subtrajectories overlap with each other in at most one point, the y-coordinates of our monotone paths overlap in at most one point. Similarly, the y-coordinates of our monotone paths overlap with the y-interval from s to t in at most one point. See Figure 2.2, left.

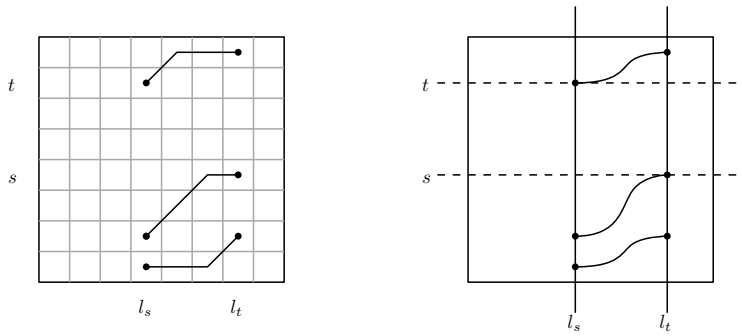


Figure 2.2: An example of three monotone paths from l_s to l_t in the discrete (left) and continuous (right) free space diagrams. The monotone paths overlap with each other in at most one point, and overlap with the y-interval from s to t in at most one point.

The second step is to iterate over all reference subtrajectories with length at least ℓ. Buchin et al. [36] show that, for SC under the discrete Fréchet distance, there are only $O(n)$ candidate reference subtrajectories to consider, and of these, no reference subtrajectory is a subtrajectory of any other reference subtrajectory.

The third step is, given a candidate subtrajectory starting at s and ending at t, deciding whether there is a subtrajectory cluster satisfying the conditions of SC with this candidate subtrajectory as its reference subtrajectory. The third step is an important subproblem in both the algorithm of Buchin et al. [36] and our algorithm. We state the subproblem

formally.

Subproblem 2. *Given a trajectory T of complexity n, a positive integer m, a positive real value d, and a reference subtrajectory of T starting at vertex s and ending at vertex t, let l_s and l_t be two vertical lines in $F_d(T,T)$ representing the vertices s and t. Decide if there exist:*

- *$m-1$ distinct monotone paths starting at l_s and ending at l_t, such that*
- *the y-coordinate of any two monotone paths overlap in at most one point, and*
- *the y-coordinate of any monotone path overlaps the y-interval from s to t in at most one point.*

Buchin et al. [36] solve each instance of Subproblem 2 individually in $O(n+ml)$ time, where l is the maximum complexity of the reference subtrajectory. As there are $O(n)$ reference subtrajectories, there are $O(n)$ subproblems to solve. Therefore, the total running time of the algorithm is $O(n^2+nml)$ which in the worst case is $O(n^3)$.

Next, we describe the algorithm of Buchin et al. [36] under the continuous Fréchet distance. The first step is exactly the same as the discrete case, except that we substitute the discrete free space diagram with the continuous free space diagram. See Figure 2.2, right.

The second step is significantly different between the discrete and continuous case. In the continuous case, the starting point s and ending point t may be any arbitrary points on the trajectory T, not just vertices of T. Buchin et al. [36] claim that there are $O(n^2)$ critical points in the free space diagram, and the vertical line l_s representing the starting point of the reference subtrajectory must pass through one of these critical points. We show in Section 2.3.2 that there are $\Omega(n^3)$ critical points, and we show in Lemma 34 that there are $O(n^3)$ critical points. Hence, in the general case, there are $\Theta(n^3)$ possible reference subtrajectories to consider.

The third step is to solve Subproblem 2. Instead of l_s and l_t representing vertices that are the starting and ending points of the reference subtrajectories, we consider l_s and l_t representing arbitrary points that are the starting and ending points. Buchin et al. [36] solve each instance of Subproblem 2 in $O(nm)$ time. As there are $O(n^3)$ critical points and therefore $O(n^3)$ reference subtrajectories to consider, the overall running time of Buchin et al.'s [36] algorithm is $O(n^4m)$, which in the worst case is $O(n^5)$.

2.3 Technical Overview

Our main technical contributions in this chapter are cubic upper and lower bounds for Subtrajectory Clustering (SC) under the continuous Fréchet distance.

As a stepping stone towards our cubic upper bound, we study two special cases. The first special case is the SC problem under the discrete Fréchet distance. The second special case is the SC problem under the continuous Fréchet distance, but under the restriction that the reference subtrajectory must be a vertex-to-vertex subtrajectory. The final case is the general case. In Section 2.3.1, we provide an overview of our key insights in each case. Detailed descriptions of the algorithms and their analyses can be found in Sections 2.4 and 2.5.

For our lower bounds, assuming SETH, we show in Section 2.3.2 a simple reduction that proves that there is no $O(n^{2-\varepsilon})$ algorithm for SC for any $\varepsilon > 0$ under the discrete Fréchet distance. Then, we show that there is no $O(n^{3-\varepsilon})$ algorithm for SC for $\varepsilon > 0$ under the

continuous Fréchet distance. We provide an overview of our reduction from 3OV to SC in Section 2.3.2, and a detailed analysis of our construction can be found in Section 2.6.

2.3.1 Algorithm Overview

Our algorithms, both in the special and general cases, build on the work of Buchin et al. [36]. We make several key insights that lead to improvements over previous work. In this section, we present our main technical contributions, and defer relevant proofs to Sections 2.4 and 2.5.

Key Insight 1: Reusing monotone paths between different subproblems

As previously stated, the algorithm of Buchin et al. [36] divides SC into one subproblem per candidate reference subtrajectory. For the discrete Fréchet distance, there are $O(n)$ candidate reference subtrajectories, so SC is divided into $O(n)$ instances of Subproblem 2. Buchin et al. [36] showed that each instance of Subproblem 2 can be solved in $O(n + ml) \approx O(n^2)$ time individually, so all $O(n)$ subproblems can be solved in $O(n^2 + nml) \approx O(n^3)$ time.

Our first key insight is that we need not handle each subproblem individually. If we can reuse monotone paths between the $O(n)$ subproblems, this could significantly speed up the algorithm. For example, suppose that a subproblem starts at vertex s and ends at vertex t, whereas an adjacent subproblem starts at vertex $s + 1$ and ends at vertex $t + 1$. Suppose that we solve the subproblem (s, t) first, and we solve the subproblem $(s+1, t+1)$ next. Our key insight is to reuse the monotone paths that we computed in subproblem (s, t) to guide our search in subproblem $(s + 1, t + 1)$. In fact, since almost all grid cells in subproblem $(s + 1, t + 1)$ have already appeared in the subproblem (s, t), the speedup from using previously computed paths could be quite large. See Figure 2.3, left.

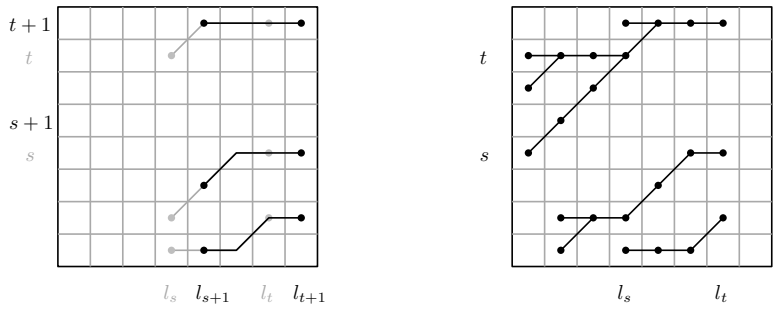


Figure 2.3: Monotone paths from l_s to l_t that are similar to monotone paths from l_{s+1} to l_{t+1} (left). The link-cut tree data structure retaining monotone paths from previous values of s and t (right).

In short, our approach is as follows. We sort the $O(n)$ subproblems by the x-coordinates of their vertical lines l_s and l_t. We consider those with the smallest x-coordinates first. For each subproblem, we perform a greedy depth first search to find $m - 1$ non-overlapping monotone paths. Our search algorithm is very similar to and has the same running time as the original search algorithm by Buchin et al. [36].

While performing our greedy depth first search, we maintain a dynamic tree data structure to store our monotone paths as we compute them. In particular, we use a link-cut tree [146], to store a set of rooted trees. The invariant maintained by our data structure is that every node has a monotone path to the root of its link-cut tree, as shown in Figure 2.3, right. The link-cut data structure is maintained via edge insertions and deletions, which require $O(\log n)$ time per update. Using this data structure, we can significantly reuse the monotone paths between subproblems. Whenever we visit a node that has been considered by a previous subproblem, we can simply query for its root in the link-cut data structure, without needing to recompute the monotone path.

In Section 2.4.1, we provide details of our greedy depth first search. In Section 2.4.2 we provide details of how we apply the link-cut tree data structure to our greedy depth first search. We then use amortised analysis to bound the running time. Putting this together yields:

Theorem 3. *There is an $O(n^2 \log n)$ time algorithm for SC under the discrete Fréchet distance.*

Key Insight 2: Transforming continuous free space reachability into graph reachability

As we transition from the discrete case of SC to the continuous case, the free space diagram becomes significantly more complex. In particular, reachability in continuous free space is calculated in a fundamentally different way to reachability in the discrete free space. The most common approach for computing the continuous Fréchet distance is to compute the reachable space. Reachable space is defined to be the subset of free space that is reachable via monotone paths from the bottom left corner, as shown in Figure 2.4, left. For example, this is the approach used by Alt and Godau [12] in their original algorithm, and also by Buchin et al. [37] in their improved algorithm.

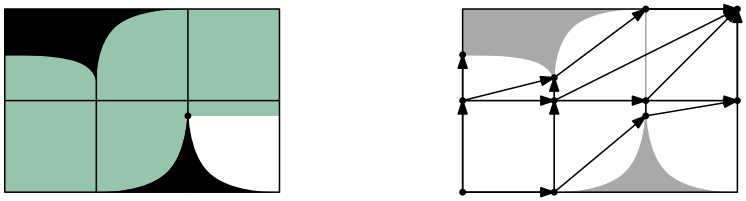


Figure 2.4: The reachable free space from the bottom left corner, shaded in green (left). The directed graph $G = (V, E)$ where there is a path in the graph if and only if there is a monotone path in the continuous free space diagram (right).

Maintaining the reachable space is fundamentally incompatible with our algorithm. The reason is that the reachable space only considers monotone paths that start at the bottom left corner of the free space diagram. In our problem, the starting point of our monotone path constantly changes, and the reachable space for one starting point may not be used for computing monotone paths from any other starting point.

We propose an alternative to computing the reachable space in the continuous free space diagram. A critical point in the continuous free space diagram is the intersection of the boundary of a cell with the boundary of the free space for that cell (ellipse), or is a cell corner. We build a directed graph $G = (V, E)$ of size $O(n^2 \log n)$, where V is the set of

$O(n^2)$ critical points in the free space diagram, so that there is a monotone path between two points in the free space diagram if and only if there is a path between the same two points in the directed graph G. See Figure 2.4, right. Note that reachability in this directed graph works for any pair of critical points, not just those where the starting point is the bottom left corner.

In Section 2.5.1, we establish the equivalence between continuous free space reachability and graph reachability in G. In Section 2.5.2, we combine this with our first key insight to obtain an $O(n^2 \log^2 n)$ time algorithm for SC under the continuous Fréchet distance, in the special case where the reference subtrajectory is vertex-to-vertex.

Key Insight 3: Handling additional critical points and reference subtrajectories

In all special cases considered so far, there are $O(n)$ candidate reference subtrajectories, and hence $O(n)$ instances of Subproblem 2. However, in the continuous case where the reference subtrajectory may be any arbitrary subtrajectory, there are significantly more candidate reference subtrajectories to consider. We call a potential starting point for the reference subtrajectory either an internal or external critical point. The difference is that an internal critical point lies in the interior of a free space cell, whereas an external critical point lies on the boundary of a free space cell.

We show that there are $O(n^3)$ internal critical points, and therefore $O(n^3)$ instances of Subproblem 2 in the general case. The number of internal critical points is dominated by critical points of the following type: a point that is on the boundary between free and non-free space, and shares a y-coordinate with an external critical point. See Figure 2.5, left. Surprisingly, these $O(n^3)$ internal critical points are necessary, and form one of the key components of our lower bound.

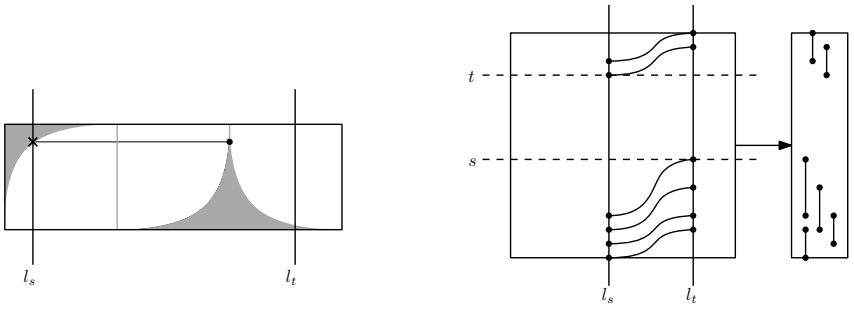


Figure 2.5: An example of an internal critical point, marked with a cross (left). The overlapping y-intervals maintained by the dynamic monotone interval data structure (right).

Given this increase in critical points and reference subtrajectories in the worst case, if we were to naïvely apply our first two key insights, we would obtain an $O(n^3 m)$ time algorithm for the general case. The difference is that, previously, our algorithm's running times were dominated by maintaining the link-cut tree data structure over our set of critical points. However, now that there are significantly more reference subtrajectories, the relatively simple process of enumerating up to $m-1$ non-overlapping monotone paths per reference subtrajectory is the new bottleneck. To handle this bottleneck, we require another data structure. The dynamic monotone interval data structure [90] reports whether there exist $m-1$ non-overlapping y-intervals, which represent our $m-1$ non-overlapping mono-

tone paths, without explicitly computing them. See Figure 2.5, right. Putting this together with our first two key insights we obtain the following theorem.

Theorem 4. *There is an $O(n^3 \log^2 n)$ time algorithm for SC under the continuous Fréchet distance.*

2.3.2 Lower Bound Overview

Our aim is to show that the algorithms in Theorems 3 and 4 are essentially optimal, assuming SETH. We start by showing that there is no strongly subquadratic time algorithm for SC under the discrete Fréchet distance. We achieve this by reducing from Bringmann's [28] lower bound for computing the discrete Fréchet distance.

Theorem 5. *There is no $O(n^{2-\varepsilon})$ time algorithm for SC under the discrete Fréchet distance, for any $\varepsilon > 0$, unless SETH fails.*

Proof. Assuming SETH, Bringmann [28] constructs a pair of trajectories, T_1 and T_2, so that deciding whether the discrete Fréchet distance between T_1 and T_2 is at most one has no $O(n^{2-\varepsilon})$ time algorithm for any $\varepsilon > 0$. Let the trajectories T_1 and T_2 lie within a ball of radius r centered at the origin, and let ℓ be the sum of the lengths of the trajectories T_1 and T_2. Let $\lambda = \ell + 2r$, and place point A at $(-4\lambda, 0)$ and point B at $(4\lambda, 0)$. Construct the trajectory $T = A \circ T_1 \circ B \circ A \circ T_2 \circ B$. We will show that there is a subtrajectory cluster of T with parameters $m = 2$, $\ell = 8\lambda$ and $d = 1$ if and only if the discrete Fréchet distance between T_1 and T_2 is at most one.

Our key observation is that the subtrajectories $A \circ T_1 \circ B$ and $A \circ T_2 \circ B$ have discrete Fréchet distance of at most one if and only if T_1 and T_2 have a discrete Fréchet distance of at most one.

For the if direction, the pair of subtrajectories $A \circ T_1 \circ B$ and $A \circ T_2 \circ B$ each have length at least 8λ, and have discrete Fréchet distance at most one from one another.

For the only if direction, suppose there is a subtrajectory cluster of size two. Note that subtrajectory $A \circ T_1$, or any other subtrajectory that contains at most one copy of A or B, will have length strictly less than 8λ. Therefore, A and B must occur in both subtrajectories in the cluster. The corresponding A's and B's must match to one another for the discrete Fréchet distance to be at most one, so they must be in the same order in their respective subtrajectory. Without loss of generality, the first subtrajectory contains $A \circ T_1 \circ B$ and the second subtrajectory contains $A \circ T_2 \circ B$. So the discrete Fréchet distance between T_1 and T_2 must be at most one, as required.

As there is no $O(n^{2-\varepsilon})$ time algorithm for deciding if the discrete Fréchet distance is at most one for any $\varepsilon > 0$, there is no $O(n^{2-\varepsilon})$ time algorithm for SC under the discrete Fréchet distance. □

Next, we show that there is no strongly subcubic time algorithm for SC under the continuous Fréchet distance. We achieve this by reducing the three orthogonal vectors problem (3OV) to SC.

Problem 6 (3OV)**.** *We are given three sets of vectors $\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$, $\mathcal{Y} = \{Y_1, Y_2, \ldots, Y_n\}$ and $\mathcal{Z} = \{Z_1, Z_2, \ldots, Z_n\}$. For $1 \leq i, j, k \leq n$, each of the vectors X_i, Y_j and Z_k are binary vectors of length W. Our problem is to decide whether there exists a triple of integers $1 \leq i, j, k \leq n$ such that X_i, Y_j and Z_k are orthogonal. The three vectors are orthogonal if $X_i[h] \cdot Y_j[h] \cdot Z_k[h] = 0$ for all $1 \leq h \leq W$.*

We employ a three step process in our reduction in Section 2.6. First, given a 3OV instance $(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$, we construct in $O(nW)$ time an SC instance (T, m, ℓ, d) of complexity $O(nW)$. Second, for this instance, we consider the free space diagram $F_d(T, T)$ and prove various properties of it. Third, we use the properties of $F_d(T, T)$ to prove that $(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$ is a YES instance if and only if (T, m, ℓ, d) is a YES instance.

Our reduction implies that there is no $O(n^{3-\varepsilon})$ time algorithm for SC for any $\varepsilon > 0$, unless SETH fails. If such an algorithm for SC were to exist, then by our reduction we would obtain an $O(n^{3-\varepsilon}W^{3-\varepsilon})$ time algorithm for 3OV. But under the Strong Exponential Time Hypothesis (SETH), there is no $O(n^{3-\varepsilon}W^{O(1)})$ time algorithm for 3OV, for any $\varepsilon > 0$ [165].

Next, we present the three key components of our reduction. For the full reduction see Section 2.6.

Key Component 1: Diamonds in continuous free space

As a stepping stone towards the full reduction, we construct a trajectory T so that $F_d(T, T)$ has $\Theta(n^3)$ internal critical points. This weaker result shows that the analysis of our algorithm in Section 2.5.3 is essentially tight, up to polylogarithmic factors.

To obtain these $\Theta(n^3)$ internal critical points, we introduce the first key component of our reduction. It is a method to generate two curves T_1 and T_2, so that $F_d(T_2, T_1)$ consists predominantly of free space, with small regions of diamond-shaped non-free space. By varying the positions of the vertices on T_1 and T_2, we can change both the position and sizes of these small diamonds in $F_d(T_2, T_1)$.

To construct T_1 and T_2, we use the polar coordinates in the complex plane. Recall that $r \operatorname{cis} \theta$ has distance r from the origin and is at an anticlockwise angle of θ from the positive real axis. The vertices of T_1 will be on the ball of radius r centered at the origin, as illustrated in Figure 2.6, left. The vertices of T_2 will be on the ball of radius r' centered at the origin, where $r' > 10r$. For now, define $\phi = \frac{\pi}{2}$, although a much smaller value of ϕ will be used in the full reduction.

Let O be the origin. Place the vertices A, B, C, D, respectively, at $r \operatorname{cis} 0$, $r \operatorname{cis} \phi$, $r \operatorname{cis}(2\phi)$, and $r \operatorname{cis}(3\phi)$. Place the vertices A^+, B^+, C^+, D^+, respectively, at $r' \operatorname{cis} \pi$, $r' \operatorname{cis}(\pi + \phi)$, $r' \operatorname{cis}(\pi + 2\phi)$, and $r' \operatorname{cis}(\pi + 3\phi)$. Note that A and A^+ are diametrically opposite from one another, but on differently sized circles. Let $||\cdot||$ denote the Euclidean norm. Define $d = ||A^+B||$. Then A^+ is within distance d of B, C and D, but not within distance d of A. Similarly, A is within distance d of B^+, C^+ and D^+, but not within distance d of A^+. We call the pairs (A^+, A), (B^+, B), (C^+, C) and (D^+, D) antipodes.

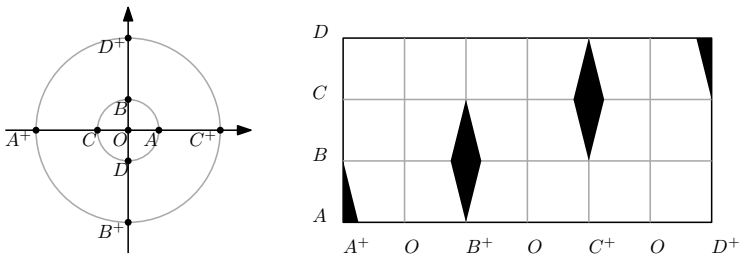


Figure 2.6: The vertices of T_1 and T_2 (left), and the free space diagram $F_d(T_2, T_1)$ (right).

Our key insight is that, if the only vertices that appear in T_1 are A, B, C and D, and the only vertices that appear in T_2 are A^+, B^+, C^+ and D^+, then the free space diagram

$F_d(T_2, T_1)$ will consist predominantly of free space, with small regions of non-free space. The centers of the regions of non-free space will have x-coordinate v^+ and y-coordinate v, where (v^+, v) are antipodes. Section 2.6.3 is dedicated to properties of antipodal pairs, and Section 2.6.5 is dedicated to how these small regions associated with the antipodes fit in with the rest of the free space diagram.

Let $A \circ B \circ C \circ D$ denote the trajectory formed by joining, with straight segments, the vertices A, B, C, and D in order. Define:

$$\begin{aligned} T_1 &= A \circ B \circ C \circ D \\ T_2 &= A^+ \circ O \circ B^+ \circ O \circ C^+ \circ O \circ D^+. \end{aligned}$$

The vertices of T_1 and T_2 and the free space diagram $F_d(T_2, T_1)$ are shown in Figure 2.6, left. It is not too difficult to see that by swapping the order of the vertices in T_2, or by inserting additional vertices into T_2, we can change the placements of our diamonds, or partial diamonds, in the free space in Figure 2.6, right.

We would also like to vary the sizes of our diamonds. To do this, we introduce points that are approximately distance r' from the origin. Let B^- be on BB^+ so that $||BB^-|| = d$, that is, $B^- = (d-r)\operatorname{cis}\phi$. Similarly, define $C^- = (d-r)\operatorname{cis}(2\phi)$. Next, let $B_1^+, B_2^+, \ldots, B_n^+$ be points on segment B^-B^+ so that $B^-, B_1^+, \ldots, B_n^+, B^+$ are evenly spaced. Then the antipodal pair (B_i^+, B) would generate a different sized diamond to the antipodal pair (B^+, B). This is because $||B_i^+ B|| < d$, so the non-free space it generates will be smaller.

We are ready to construct the trajectory T with n^3 internal critical points. Define

$$\begin{aligned} T_1 &= A \circ B \circ C \circ D, \\ T_2 &= \bigcirc_{1 \le i \le n}(B_i^+ \circ O) \circ \bigcirc_{1 \le i \le n}(C^+ \circ O), \end{aligned}$$

and set $T = \bigcirc_{1 \le i \le n}(T_1) \circ T_2$. Note that T has linear complexity.

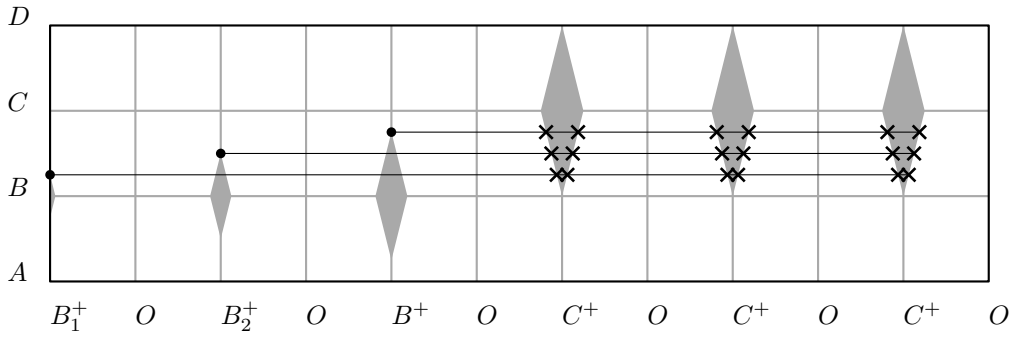


Figure 2.7: A free space diagram $F_d(T_2, T_1)$ with $2n^2$ critical points.

In Figure 2.7, on the left, we have the diamonds associated with the antipodal pairs (B_i^+, B). On the right, we have the diamonds associated with the antipodal pairs (C^+, C). Each diamond on the left generates an external critical point for its topmost point (marked with dots). Moreover, all these external critical points have different y-coordinates. Each of these distinct y-coordinates generates two internal critical points on each diamond on the right (marked with crosses). Therefore, $F_d(T_2, T_1)$ has $2n^2$ internal critical points. Since $F_d(T, T)$ contains n copies of $F_d(T_2, T_1)$ in it, we have that $F_d(T, T)$ contains $\Theta(n^3)$ critical points, as required.

Key Component 2: Combining diamonds to form gadgets

Recall that in our first component, we generate pairs of curves T_1 and T_2, so that the only regions of non-free space in $F_d(T_2, T_1)$ are small diamonds. We can change T_1 and T_2 to vary the number, positions and sizes of the diamonds in $F_d(T_2, T_1)$. Our second key component is to position the diamonds in a way that encodes boolean formulas. To help simplify the description of these boolean formulas, we only consider the two sets $(\mathcal{X}, \mathcal{Z})$, making the input a 2OV instance for now.

Our first gadget is an OR gadget and checks if one of two booleans is zero. Our second gadget is an AND gadget and checks if a pair of vectors are orthogonal.

Our OR gadget receives as input two booleans, X and Z, and constructs a pair of curves T_1 and T_2. Let s and t be the starting and ending points of T_2, and l_s and l_t be the vertical lines corresponding to s and t. The trajectories T_1 and T_2 are constructed in such a way that, if $X \cdot Z = 0$ then there is a monotone path from l_s to l_t, otherwise, there is no such monotone path.

We use the same definitions of vertices as in the first key component. The pairs (A^+, A), (B^+, B), (C^+, C) and (D^+, D) are antipodes. Define $B_X^+ = O$ if $X = 0$, and $B_X^+ = B^+$ otherwise. Similarly, define $D_Z^+ = O$ if $Y = 0$, and $D_Y^+ = D^+$ otherwise.

Now we are ready to construct the curves T_1 and T_2.

$$\begin{aligned} T_1 &= A \circ B \circ C \circ D \\ T_2 &= B^+ \circ O \circ C^+ \circ O \circ D^+ \circ O \\ &\quad \circ A^+ \circ O \circ B_X^+ \circ O \circ C^+ \circ O \circ D_Z^+. \end{aligned}$$

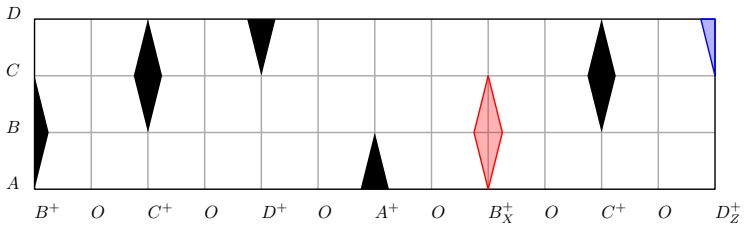


Figure 2.8: The free space diagram $F_d(T_2, T_1)$ for the OR gadget. The red diamond in column B_X^+ disappears if $X = 0$, and the blue diamond in column D_Z^+ disappears if $Z = 0$.

The free space diagram $F_d(T_2, T_1)$ is shown in Figure 2.8. The red diamond in column B_X^+ disappears if $X = 0$, whereas the blue diamond in column D_Z^+ disappears if $Z = 0$. If either one is zero, there is a gap for there to be a monotone path from l_s to l_t. Otherwise, there is no gap, and no monotone path. This completes the description of the OR gadget.

Our AND gadget receives as input a pair of binary vectors $X = (X[1], X[2], \ldots, X[W])$ and $Z = (Z[1], Z[2], \ldots, Z[W])$, and constructs a pair of trajectories T_1 and T_2. Let s and t be the starting and ending points of T_2, and let l_s and l_t be the vertical lines corresponding to s and t. If X and Z are orthogonal, in other words, if $X[h] \cdot Z[h] = 0$ for all $1 \leq h \leq W$, then the maximum number of monotone paths from l_s to l_t is W. If X and Z are not orthogonal, in other words, if $X[h] \cdot Z[h] = 1$ for some $1 \leq h \leq W$, then the maximum number of monotone paths from l_s to l_t is $W - 1$. Let r be a positive real and $r' > 10r$. Now, let $\phi = \frac{2\pi}{2W+4}$, let $d = ||r \operatorname{cis} \phi - r' \operatorname{cis} \phi||$ and, let $\varepsilon = r' + r - d$.

Next, we define the vertices of T_1 and T_2.

$$
\begin{array}{rcl}
A & = & r\operatorname{cis}0 \\
B_{u,h} & = & r\operatorname{cis}(u\cdot\phi),\ \text{if } u\neq 2h \\
B_{u,h} & = & (r-\frac{2}{3}\varepsilon)\operatorname{cis}(u\cdot\phi),\ \text{if } u=2h \\
C_h & = & r\operatorname{cis}((2W+2)\cdot\phi),\ \text{if } Z[h]=1 \\
C_h & = & (r-\frac{2}{3}\varepsilon)\operatorname{cis}((2W+2)\cdot\phi),\ \text{if } Z[h]=0 \\
D & = & r\operatorname{cis}((2W+3)\cdot\phi) \\
O & = & 0\operatorname{cis}0 \\
A^+ & = & r'\operatorname{cis}\pi \\
B^+_{u,0} & = & r'\operatorname{cis}(\pi+u\cdot\phi) \\
B^+_{u,1} & = & r'\operatorname{cis}(\pi+u\cdot\phi),\ \text{if } u \text{ is odd} \\
B^+_{u,1} & = & r'\operatorname{cis}(\pi+u\cdot\phi),\ \text{if } u \text{ is even and } X[\tfrac{u}{2}]=1 \\
B^+_{u,1} & = & (r'-\frac{2}{3}\varepsilon)\operatorname{cis}(\pi+u\cdot\phi) \\
 & & \quad \text{if } u \text{ is even and } X[\tfrac{u}{2}]=0 \\
C^+_0 & = & r'\operatorname{cis}(\pi+(2W+2)\cdot\phi) \\
C^+_1 & = & (r'-\frac{2}{3}\varepsilon)\operatorname{cis}(\pi+(2W+2)\cdot\phi) \\
D^+ & = & r'\operatorname{cis}(\pi+(2W+3)\cdot\phi)
\end{array}
$$

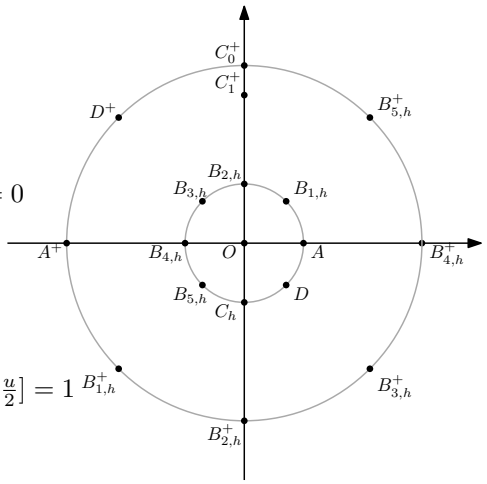


Figure 2.9: Vertices of T_1 and T_2, for $W=2$.

Now we can construct T_1 and T_2.

$$
\begin{array}{rcl}
T_1 & = & \bigcirc_{1\le h\le W}\big(A\circ\bigcirc_{1\le u\le 2W+1}(B_{u,h})\circ C_h\circ D\big)\circ A \\
T_2 & = & \bigcirc_{1\le u\le 2W+1}(B^+_{u,0}\circ O)\circ C^+_0\circ O \\
 & & \circ A\circ O\circ\bigcirc_{1\le u\le 2W+1}(B^+_{u,1}\circ O)\circ C^+_1\circ O \\
 & & \circ A\circ O\circ\bigcirc_{1\le u\le 2W+1}(B^+_{u,0}\circ O)\circ D^+\circ O\circ A\circ O\circ C^+_0\circ O\circ D\circ O
\end{array}
$$

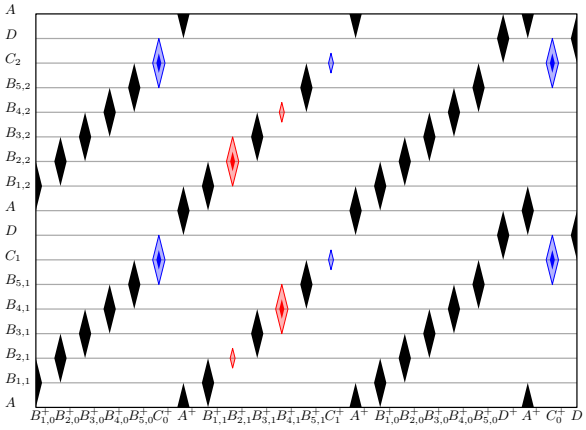


Figure 2.10: The free space diagram $F_d(T_2,T_1)$ for the AND gadget, for $W=2$.

The free space diagram $F_d(T_2,T_1)$ is shown in Figure 2.10, for $W=2$. The labels for the repeated O's are omitted from the x-axis of $F_d(T_2,T_1)$. For $W=2$, as we can see, there are four red diamonds, in columns $B^+_{2,1}$ and $B^+_{4,1}$, and six blue diamonds, in columns C^+_0, C^+_1 and C^+_0. The red diamond in column $B^+_{2,1}$ and row $B_{2,1}$ disappears if and only if $X[1]=0$. The red diamond in column $B^+_{4,2}$ and row $B_{4,2}$ disappears if and only if $X[2]=0$. The other red diamonds may shrink, but do not completely disappear. The blue diamond in column

C_1^+ and row C_1 disappears if and only if $Z[1] = 0$. The blue diamond in column C_1^+ and row C_2 disappears if and only if $Z[2] = 0$. The other blue diamonds may shrink, but do not completely disappear.

The bottom half of the free space diagram is essentially an OR gadget for $X[0]$ and $Z[0]$, and the top half is essentially an OR gadget for $X[1]$ and $Z[1]$. In Figure 2.11 left, the vectors X and Z are orthogonal. In this case, there are two monotone paths, one for the OR gadget $X[0] \cdot Z[0] = 0$ and one for the OR gadget $X[1] \cdot Z[1] = 0$. In Figure 2.11 right, the vectors X and Z are not orthogonal. In this case, there is a maximum of one monotone path, and this monotone path passes through the "gap" between the two OR gadgets.

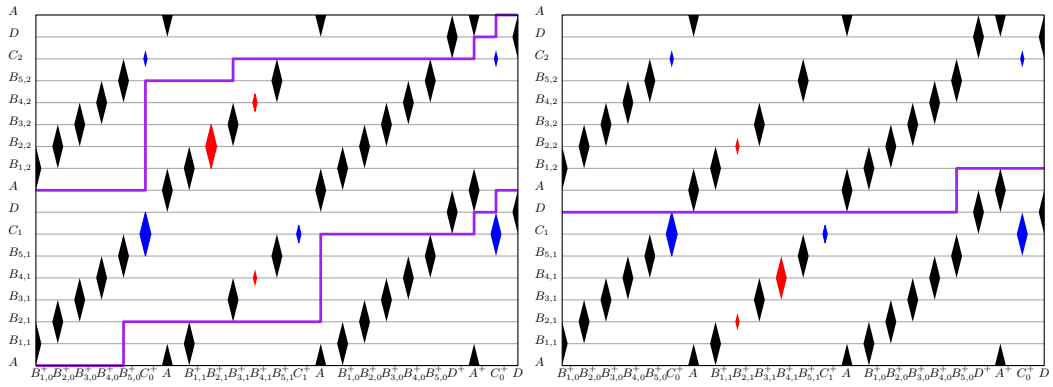


Figure 2.11: For $W = 2$, if X and Z are orthogonal, there are W paths (left), whereas if X and Z are not orthogonal, there are $W - 1$ paths (right).

For general values of W, the AND gadget is a stack of OR gadgets for $X[h]$ and $Z[h]$ for $1 \leq h \leq W$. If $X[h] \cdot Z[h] = 0$ for all $1 \leq h \leq W$, we obtain W monotone paths, one per OR gadget. Otherwise, we obtain a maximum of $W - 1$ monotone paths, one for each gap between two consecutive OR gadgets. This completes the description of the AND gadget.

Key Component 3: Combining gadgets to form the full reduction

The gadgets in our full construction are inspired by the gadgets in our second key component. However, the gadgets in our full construction are more sophisticated in two important ways.

First, in the gadgets constructed so far, we only consider a single pair of vertical lines, that is, the pair of vertical lines that correspond to the start and end points of T_2. In our full construction, we consider all vertical lines that start and end at internal critical points. In particular, we consider n^2 pairs of vertical lines that correspond to a pair of integers (i, j) where $1 \leq i, j \leq n$. Each of these vertical lines passes through nW internal critical points. The nW internal critical points correspond to pairs of integers (k, h) where $1 \leq k \leq n$ and $1 \leq h \leq W$.

Second, in the gadgets we have constructed so far, we only consider two sets of binary vectors. In our full construction, we consider three sets of binary vectors, $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$.

With these two key differences in mind, we can describe the gadgets in our full construction. We receive as input a 3OV instance, in other words, we are given three sets $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{Z}$, each containing n binary vectors of length W.

First, we describe the OR gadget that appears in our full construction. There will be nW copies of the OR gadget. Given $1 \leq k \leq n$ and $1 \leq h \leq W$, the OR gadget for the

pair (k, h) satisfies the following property for all $1 \leq i, j \leq n$: there are two monotone paths from l_s to l_t if $X_i[h] \cdot Y_j[h] \cdot Z_k[h] = 0$, whereas there is a maximum of one monotone path from l_s to l_t if $X_i[h] \cdot Y_j[h] \cdot Z_k[h] = 1$, where l_s and l_t are the vertical lines corresponding to the pair (i, j).

In Figure 2.12, there is one monotone path from l_s to l_t, and this is the only monotone path in the OR gadget in the case that $X_i[h] \cdot Y_j[h] \cdot Z_k[h] = 1$. On the other hand, if $X_i[h] \cdot Y_j[h] \cdot Z_k[h] = 0$, then there are two monotone paths in the OR gadget.

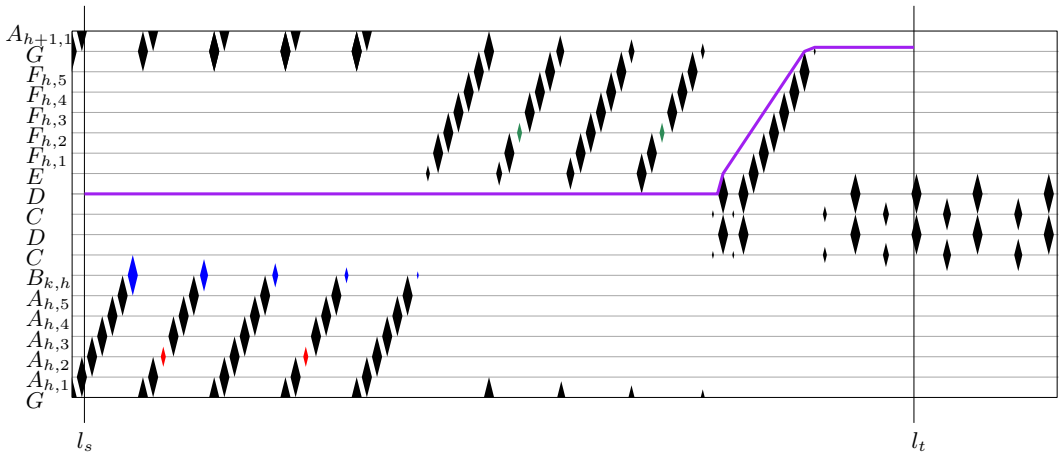


Figure 2.12: The OR gadget for booleans $X_i[h]$, $Y_j[h]$ and $Z_k[h]$.

In Figure 2.13, we show the two monotone paths from l_s to l_t in three separate cases: for $X_i[h] = 0$, $Y_j[h] = 0$ and $Z_k[h] = 0$. We briefly describe the behaviour in these three cases. If $X_i[h] = 0$, a red diamond in the bottom right disappears and one of the two monotone paths passes through this gap. If $Y_j[h] = 0$, a green diamond disappears and one of the two monotone paths passes through this gap. Finally, if $Z_k[h] = 0$, all blue diamonds shrink in size, allowing the lower monotone path to have a much smaller maximum y-coordinate.

It is worth noting the connection between these OR gadgets and internal critical points. In all three cases, the starting point of the first monotone path is an internal critical point. If $Y_j[h] = 0$, all starting and ending points of the two monotone paths are either internal critical points, or share a y-coordinate with an internal critical point.

Similarly to the AND gadget in the second key component, we stack W copies of the OR gadget on top of each other. Each stack of OR gadgets checks if a triple of vectors (X_i, Y_j, Z_k) are orthogonal. There are $2W + 1$ monotone paths from l_s to l_t if $X_i[h] \cdot Y_j[h] \cdot Z_k[h] = 0$ for all $1 \leq h \leq W$, whereas there are $2W$ paths from l_s to l_t if $X_i[h] \cdot Y_j[h] \cdot Z_k[h] = 1$ for some $1 \leq h \leq W$.

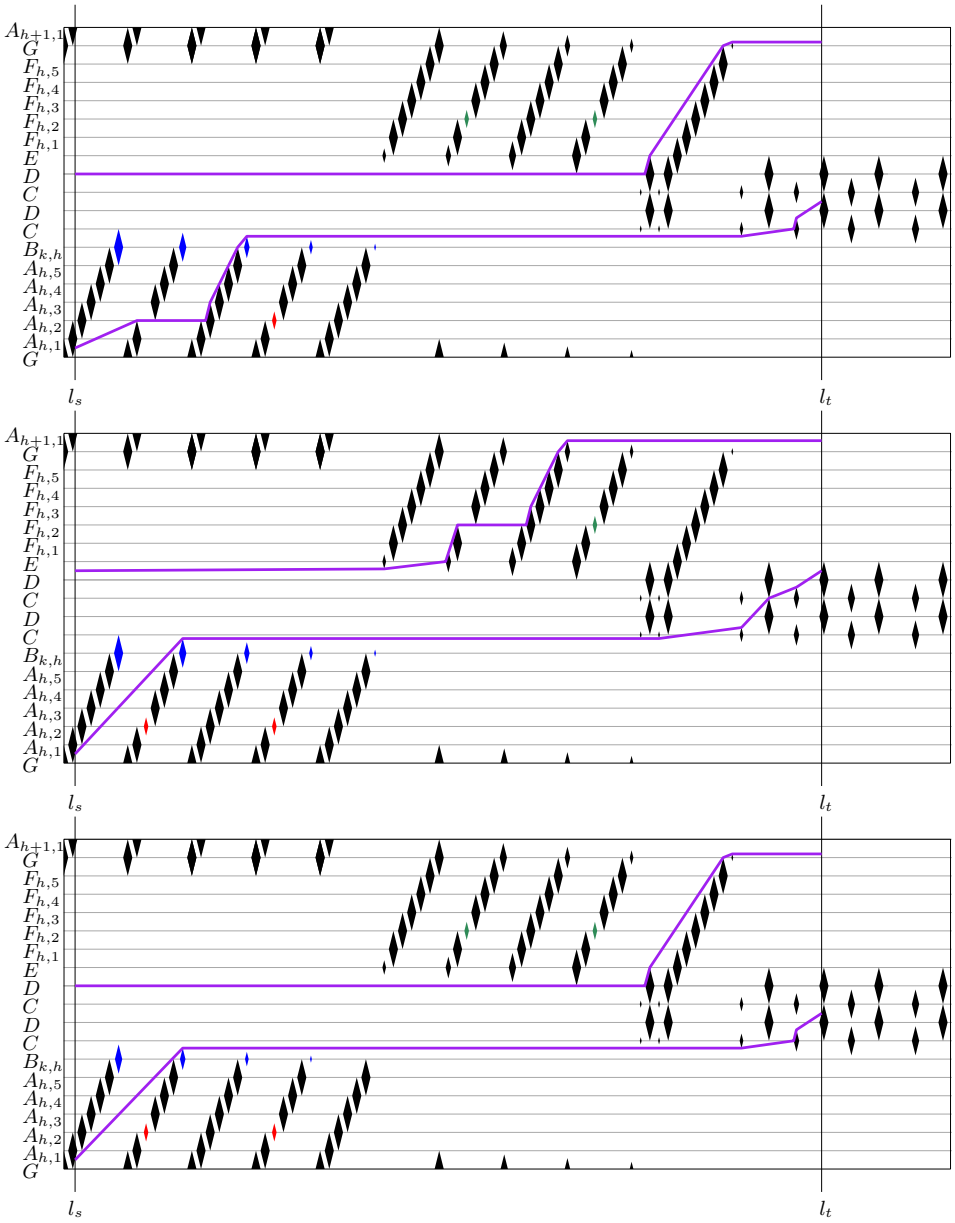


Figure 2.13: The two monotone paths from l_s to l_t in the cases where $X_i[h] = 0$, $Y_j[h] = 0$ and $Z_k[h] = 0$, respectively

Finally, we combine these AND gadgets to form the full construction. We give a high level overview. We stack, one on top of another, n copies of the AND gadgets. We add non-free space between consecutive AND gadgets, so that monotone paths in one AND gadget cannot interact with monotone paths in another AND gadget. We let the two curves that generate this free space be T_1 and T_2. We join T_1 and T_2 end to end to form the final curve and set $m = 2nW + 2$. The intuition behind setting $m = 2nW + 2$ is that, as there are n copies of the AND gadget, if there are $m - 1$ monotone paths, then by the pigeonhole principle, we must have one AND gadget with $2W + 1$ monotone paths, which implies that X_i, Y_j and Z_k are orthogonal for some triple (i, j, k). Otherwise, all AND gadgets have $2W$ monotone paths, so for all triples (i, j, k), there is some h so that $X_i[h] \cdot Y_j[h] \cdot Z_k[h] = 1$. Putting this all together yields the following theorem.

Theorem 7. *There is no $O(n^{3-\varepsilon})$ time algorithm for SC under the continuous Fréchet distance, for any $\varepsilon > 0$, unless SETH fails.*

2.4 Discrete Fréchet Distance

The main theorem that we will prove in this section is:

Theorem 3. *There is an $O(n^2 \log n)$ time algorithm for SC under the discrete Fréchet distance.*

We will be using the discrete free space diagram extensively in our algorithm. Recall that for the discrete Fréchet distance, the free space diagram $F_d(T, T)$ consists of n^2 grid points. A grid point (x, y) is free if vertices x and y of the trajectory T are within distance d of one another. A monotone path is a sequence of free grid points where a grid point (x, y) is followed by $(x + 1, y)$, $(x, y + 1)$, or $(x + 1, y + 1)$.

In Section 2.4.1, we show how to solve Subproblem 2 in $O(nl)$ time, where $l = t - s$, under the discrete Fréchet distance. Then, in Section 2.4.2, we show how to extend this to an algorithm that solves SC in $O(n^2 \log n)$ time, under the discrete Fréchet distance.

2.4.1 Subproblem 2 under the discrete Fréchet distance

We inductively define our algorithm for Subproblem 2 under the discrete Fréchet distance. In the base case, we compute the monotone path P_1 from l_s to l_t that minimises its maximum y-coordinate. In the inductive case, we compute the monotone path P_i from l_s to l_t that minimises its maximum y-coordinate, under the condition that P_i does not overlap in y-coordinate with $P_1, P_2, \ldots, P_{i-1}$ or the y-interval corresponding to the reference subtrajectory.

To compute each of the paths P_i, we begin by picking its starting point of l_s. Initially, we mark all free grid points as valid, and all non-free grid points as invalid. For $i = 1$, we pick the lowest valid grid point on l_s as the initial starting point. For $i > 1$, we pick the lowest valid grid point on l_s that has y-coordinate at least the maximum y-coordinate on P_{i-1}. From this initial starting point we begin a greedy depth first search to compute P_i.

Any grid point has up to three neighbouring grid points to explore: $(x+1, y)$, $(x+1, y+1)$ and $(x, y + 1)$, as long as these grid points are valid. Similar to the standard depth first search, we explore each branch of the search tree as far as possible first before backtracking. The greedy aspect of our depth first search is to first explore the neighbour $(x + 1, y)$, then $(x + 1, y + 1)$, and finally $(x, y + 1)$. The intuition is that we would like to minimise the

y-coordinate of our search. If we are forced to backtrack, i.e. if all neighbours lead to dead ends, we mark the grid point as invalid and backtrack to its original parent. This is the only way a free grid point becomes invalid. Once a grid point is marked as invalid, it is never reverted back to valid, and can never be used in any monotone paths. In Figure 2.14, backtracking occurred on the red paths, so these cells will be marked as invalid.

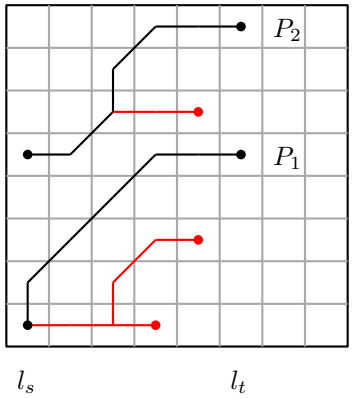


Figure 2.14: Our greedy depth first search algorithm searches for the monotone paths P_i. The red paths are dead ends, so we backtrack and then continue searching.

The greedy depth first search halts if one of the following three conditions are met. First, if our algorithm reaches l_t, we have computed P_i, and therefore we halt the search. We continue by computing the next monotone path P_{i+1}. Second, if our algorithm backtracks to our initial starting point, then our algorithm cannot find a monotone path starting at this point, and therefore we halt the search. We continue by trying to compute P_i, but starting from a higher valid grid point on l_s. Third, if our algorithm moves to a grid point with y-coordinate strictly between s and t, then the monotone path intersects the reference trajectory at more than one point, which contradicts the conditions of Subproblem 2, and therefore we halt the search. We continue by trying to compute P_i again, but we start at the lowest valid grid point on l_s that has y-coordinate at least t.

If our algorithm computes $P_1, P_2, \ldots, P_{m-1}$, then our set of $m-1$ monotone paths are returned. Otherwise, if all valid grid points on l_s are exhausted, then our algorithm returns that there is no set of $m-1$ monotone paths. This completes the statement of our algorithm.

Next we argue the correctness of our algorithm. We make three observations. Each observation follows immediately from the greedy depth first search and the following ordering on the outgoing edges: first $(x+1, y)$, then $(x+1, y+1)$, and finally $(x, y+1)$. For our third observation, we define $l_{\geq g}$ be the set of grid points with the same x-coordinate as g and have y-coordinate greater than or equal to the y-coordinate of g.

Observation 8. *Let P be a monotone path from l_s to l_t. Throughout the execution of the greedy depth first search, all grid points on P will remain valid.*

Observation 9. *Suppose our algorithm starts at a grid point on l_s and does not find a monotone path to l_t. Then there is no monotone path from l_s to l_t starting at that grid point.*

Observation 10. *Let P be a monotone path from l_s to l_t computed by our algorithm. Let the initial starting point of P be g and the final grid point of P be r. Then any monotone path that starts on $l_{\geq g}$ and ends on l_t must end on $l_{\geq r}$.*

The third observation formalises the intuition that if a monotone path is found, then a lower monotone path cannot exist. With these three observations in mind, we are now ready to prove the correctness of our algorithm.

Lemma 11. *There exist $m-1$ monotone paths satisfying the conditions of Subproblem 2 under the discrete Fréchet distance if and only if our algorithm returns a set of $m-1$ monotone paths.*

Proof. Suppose our algorithm returns a set of $m-1$ monotone paths. It is straightforward to check from the definition of our algorithm that our set of monotone paths all start on l_s, all end on l_t, are distinct, and overlap in at most one y-coordinate. Therefore, our $m-1$ monotone paths satisfy the conditions of Subproblem 2.

Next, we prove the converse. We assume that there exist monotone paths $Q_1, Q_2, \ldots, Q_{m-1}$ that satisfy the conditions of Subproblem 2. We will prove that our algorithm computes a set of monotone paths $P_1, P_2, \ldots, P_{m-1}$ that also satisfy the conditions of Subproblem 2.

We prove by induction that, for all $1 \leq i \leq m-1$, our algorithm computes a monotone path P_i so that the maximum y-coordinate of P_i is at most the maximum y-coordinate of Q_i. We will focus on the inductive case, since the base case follows similarly. Our inductive hypothesis implies that the maximum y-coordinate of P_{i-1} is at most the maximum y-coordinate of Q_{i-1}. So the maximum y-coordinate of P_{i-1} is at most the minimum y-coordinate of Q_i.

Next, our algorithm attempts to compute P_i. It starts at a y-coordinate that is at most the minimum y-coordinate of Q_i. By the contrapositive of Lemma 9, our algorithm computes a monotone path when considering the starting point on Q_i, if not earlier. Hence, the minimum y-coordinate of P_i is at most the minimum y-coordinate of Q_i. By Lemma 10, the maximum y-coordinate of P_i is at most the maximum y-coordinate of Q_i, completing the induction and the proof of the lemma. So our algorithm computes a set of $m-1$ monotone paths that satisfy the conditions of Subproblem 2. □

Finally, we analyse the running time of our algorithm. Whenever our algorithm visits a grid point, a constant number of operations are performed. The operations involve checking if its three neighbours are valid, moving to a neighbour, or backtracking. Each operation takes constant time.

It suffices to count the number of grid points visited by our algorithm. There are $O(nl)$ grid points in total between the vertical lines l_s and l_t where $l = t - s$. When computing a monotone path, for example when computing P_i we use a depth first search. We never visit the same grid point more than once when we compute P_i. Hence, a grid point can only be revisited when computing different monotone paths, for example, when computing P_i and P_j. However, this cannot happen very often, as we show next.

Lemma 12. *There are at most $2ml$ instances where P_j revisits a grid point that has previously been visited by P_i for some $i < j$.*

Proof. First we prove that $j \leq i + 2$. Then we use this bound to show that there are at most $2ml$ revisiting instances, as claimed.

Suppose for the sake of contradiction that a grid point is visited by P_i and P_j, and $j > i+2$. So P_j and P_i must share a y-coordinate at the shared grid point. But the monotone

paths P_{i+1} and P_{j-1} have y-coordinates that are at least the maximum y-coordinate of P_i and at most the minimum y-coordinate of P_j. Therefore, P_{i+1} and P_{j-1} must be horizontal paths. But these paths are no longer unique, which is a contradiction. This proves that $j \leq i + 2$.

There are at most $2m$ pairs (i, j) where $j - i \leq 2$, and for each such pair (i, j) there is at most one y-coordinate, i.e. l cells, where P_i and P_j can visit the same cell. In total, there are at most $2ml$ cells where both P_i and P_j can visit, for some pair (i, j). □

There are $O(nl)$ initial visits to grid points and $O(ml)$ revisits, where $l = t - s$. This yields:

Theorem 13. *There is an $O(nl)$ time algorithm that solves Subproblem 2 under the discrete Fréchet distance.*

2.4.2 SC under the discrete Fréchet distance

Similar to the previous algorithm of Buchin et al. [36], our algorithm for SC under the discrete Fréchet distance involves solving $O(n)$ instances of Subproblem 2 with a sweepline approach. We maintain a link-cut data structure [146] that allows us to reuse monotone paths during our sweep. Our data structure maintains a set of rooted trees, where the nodes of the trees are grid points in the free space diagram.

Fact 14 ([146])**.** *A link-cut tree maintains a set of rooted trees, and offers the following four operations. Each operation can be performed in $O(\log n)$ amortised time.*

- *Add a tree consisting of a single node,*
- *Attach a node (and its subtree) to another node as its child,*
- *Disconnect a node (and its subtree) from its current tree,*
- *Given a node, find the root of its current tree.*

The invariant maintained by our data structure is that there is a monotone path from any node to the root of its current tree. We will first describe our algorithm and how we update our data structure. Then we will prove our invariant in Lemma 15.

We use a sweepline approach, starting with $s = 0$ and incrementing s. For each s, we let t be the final vertex of the shortest subtrajectory that starts at s and has length $\geq \ell$. For each pair (s, t), we decide whether there exists a set of $m-1$ monotone paths between l_s and l_t that do not overlap in y-coordinate. We perform a modified version of the greedy depth first search in Section 2.4.1. Our modifications include updating the link-cut data structure whenever we explore a node, and querying the link-cut data structure to reuse paths.

Whenever our greedy depth first search moves from a current grid point, which we call g_c, to an outgoing neighbour, which we call g_n, we update our link-cut data structure. We link g_c as a child of g_n, thus attaching the subtree rooted at g_c to the tree containing g_n. Under the algorithm described in Section 2.4.1, we would continue the depth first search from the neighbour g_n. However, we already know that there is a monotone path from g_n to the root of its current link-cut tree. Hence, we set the new current node, $g_{c'}$, to be the root of the tree containing g_n and continue our greedy depth first search from there. See Figure 2.15, left.

Whenever our greedy depth first search backtracks from a current grid point, which we call g_c, we also update our link-cut data structure. An invariant maintained by our

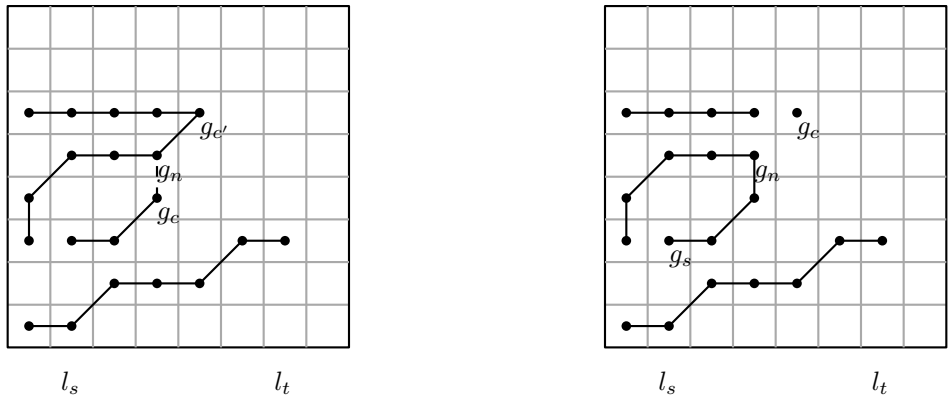


Figure 2.15: Adding a link from g_c to g_n when exploring a neighbour (left), and removing links from g_c to its children when backtracking (right).

algorithm is that the current grid point is always the root of its link-cut tree. Hence, its valid incoming neighbours are its children in the link-cut data structure. We disconnect g_c from each of its children. We backtrack to the incoming neighbour g_n with the following property. If we are currently searching for the monotone path P_i, and the initial node of P_i on l_s is g_s, then we choose g_n to be the root of the tree containing g_s. See Figure 2.15, right.

The greedy depth first search halts if any of the same three conditions as described in Section 2.4.1 are met. This completes the description of our algorithm. Next, we argue its correctness.

Lemma 15. *Let g be a grid point in the free space diagram and r be a grid point corresponding to the root of the link-cut tree containing g. Then there is a monotone path from g to r.*

Proof. Each link in the link-cut data structure is from a grid point (x, y) and to a grid point $(x, y + 1)$, $(x + 1, y + 1)$ or $(x + 1, y)$. Since r is an ancestor of g in the link-cut tree data structure, there must be a monotone path from g to r. □

Next, we prove an analogous lemma to Lemma 10. Recall that $l_{\geq g}$ is the set of grid points with the same x-coordinate as g and have y-coordinate greater than or equal to the y-coordinate of g.

Lemma 16. *For a fixed pair (s, t), suppose g is a grid point on l_s and suppose that its root r lies on l_t. Then any monotone path starting on $l_{\geq g}$ that ends on l_t must end on $l_{\geq r}$.*

Proof. By Lemma 15, there exists a monotone path P from g to r. The remainder of the proof is identical to the proof of Lemma 10, except we replace the claim that our algorithm prefers to search its lower neighbours with our algorithm prefers to link to its lower neighbours. □

Now we show that our algorithm solves SC.

Lemma 17. *There exist $m - 1$ monotone paths satisfying the conditions of SC under the discrete Fréchet distance if and only if our algorithm returns a set of $m-1$ monotone paths.*

Proof. First we prove the if direction. If our algorithm returns a set of $m-1$ monotone paths from l_s to l_t, then the conditions of Subproblem 2 are satisfied for this fixed pair of vertices (s,t). With the subtrajectory from s to t acting as the reference trajectory, and the $m-1$ monotone paths acting as the other $m-1$ subtrajectories, this cluster of subtrajectories satisfies the conditions of SC.

Next we prove the only if direction. Suppose there exists a set of $m-1$ monotone paths that satisfy SC. Let the reference subtrajectory start at s and end at u. Then the $m-1$ monotone paths between l_s and l_u corresponding to the $m-1$ subtrajectories that satisfy the conditions of Subproblem 2. Let the shortest subtrajectory starting at s with length $\geq \ell$ end at the vertex t. Shorten the $m-1$ monotone paths to be between l_s and l_t. There exists a set of $m-1$ monotone paths from l_s to l_t that satisfy the conditions of Subproblem 2. The remainder of the proof is identical to the proof of Lemma 11, except we replace our reference to Lemma 10 with a reference to Lemma 16. Therefore, our algorithm returns a set of $m-1$ monotone paths from l_s to l_t, as required. □

Finally, we analyse the overall running time of our algorithm to obtain the main theorem of Section 2.4. The running time is dominated by the greedy depth first search and updating the link-cut data structure.

Theorem 3. *There is an $O(n^2 \log n)$ time algorithm for SC under the discrete Fréchet distance.*

Proof. We will bound the number of steps in the greedy depth first search, and hence the number of updates made on the data structure. Since each grid point has degree at most three, there are at most $3n^2$ pairs of vertices between which there could be a link. At each search step, our algorithm either moves to a neighbour, in which case a link is added, or our algorithm backtracks, in which case a link is removed. Once a link is removed it cannot be added again. Hence, there are at most $O(n^2)$ updates made on the data structure, being either links or cuts, and each update takes $O(\log n)$ amortized time. Hence, the overall running time is $O(n^2 \log n)$. □

By Theorem 5, there is no $O(n^{2-\varepsilon})$ time algorithm for SC under the discrete Fréchet distance, for any $\varepsilon > 0$, assuming SETH. Therefore, our algorithm is almost tight, unless SETH fails.

2.5 Continuous Fréchet Distance

The main theorem that we will prove in this section is the following:

Theorem 4. *There is an $O(n^3 \log^2 n)$ time algorithm for SC under the continuous Fréchet distance.*

We will be using the continuous free space diagram extensively in this section. Recall that for the continuous Fréchet distance, the free space diagram $F_d(T,T)$ consists of n^2 cells. The free space within a single cell is the intersection of an ellipse with the cell. A monotone path is a continuous monotone path in the free space. For each cell, we define its critical points to be the intersection of the boundary of the free space with the boundary of the cell. A cell corner in free space is considered a critical point. There are at most eight critical points per cell.

In Section 2.5.1 we provide a modified version of the algorithm by Alt and Godau [12] that decides if the continuous Fréchet distance between two trajectories is at most d. In Section 2.5.2 we extend this algorithm to solve SC under the continuous Fréchet distance in $O(n^2 \log^2 n)$ time.

2.5.1 The continuous Fréchet distance decision problem

The problem we focus on in this section is to decide if the continuous Fréchet distance between a pair of trajectories is at most d. We let the complexities of our two trajectories be n_1 and n_2. Note that the complexities are usually denoted with m and n, however, we use n_1 and n_2 to avoid confusion with the size of the subtrajectory cluster.

Similar to the original algorithm by Alt and Godau [12], we decide whether there is a monotone path from the bottom left to the top right corner of the free space diagram. The running time of original algorithm requires $O(n_1 n_2)$ time [12], whereas ours requires $O(n_1 n_2 \log(n_1 + n_2))$ time. The original algorithm computes reachability intervals, which are horizontal or vertical propagations of critical points. We avoid computing reachability intervals. Instead, our algorithm decomposes long monotone paths into shorter ones, which we call basic monotone paths.

Recall that a critical point in the continuous free space diagram is either a cell corner, or the intersection point of a cell boundary with an elliptical boundary between free space and non-free space.

Definition 18. *A basic monotone path is a monotone path that is contained entirely in a single row (resp. column) of the free space diagram, starts at a critical point on a vertical (resp. horizontal) cell boundary, and ends on a point on a horizontal (resp. vertical) cell boundary. See Figure 2.16.*

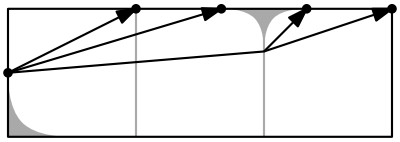


Figure 2.16: Example of basic monotone paths in a row.

The next lemma decomposes a monotone path from the bottom left to the top right corner into basic monotone paths.

Lemma 19. *Given a pair of critical points a and b in the free space diagram, there is a monotone path from a to b if and only if there is a sequence of critical points $p_0, p_1, p_2, \ldots, p_k$ so that $p_0 = a$, $p_k = b$, and there is a basic monotone path from p_i to p_{i+1} for every $i = 0, 1, \ldots, k-1$.*

Proof. For the "if" direction, there is a monotone path from p_i to p_{i+1} for all $i = 0, 1, \ldots, k-1$. Concatenating these paths yields a monotone path from a to b.

For the "only if" direction, suppose there is a monotone path Q starts at a and ends at b. Define $q_0 = a$. We define q_i inductively for $i \geq 1$. If q_i is on a vertical (resp. horizontal) boundary, then we define q_{i+1} as the first intersection of Q with a horizontal (resp. vertical) boundary occurring after q_i. Between q_i and q_{i+1} there is a monotone path, and the sequence

alternates between being on horizontal and vertical boundaries. Eventually we have $q_k = b$ for some k.

The monotone path from q_i to q_{i+1} may not be basic if q_i and q_{i+1} are not critical points. For $1 \leq i \leq k-1$, if q_i is on a vertical (resp. horizontal) boundary, we define p_i to be the critical point below (resp. left of) q_i and on the same cell boundary segment as q_i. Then p_i is either the lowest free point on a vertical boundary, or the leftmost free point on a horizontal boundary. Finally, we set $p_0 = q_0$ and $p_k = q_k$. This completes the construction of the sequence of critical points $p_0, p_1, p_2, \ldots, p_k$. It suffices to show that there is a basic monotone path from p_i to p_{i+1}. See Figure 2.17.

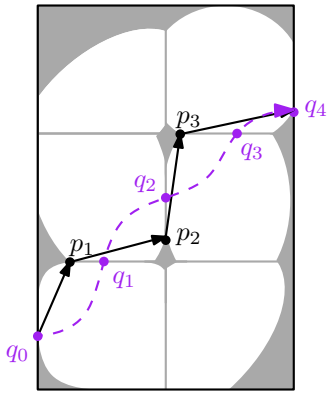


Figure 2.17: Constructing a sequence of basic monotone paths $\{p_i\}$, from any monotone path $\{q_i\}$.

There is a monotone path from p_i to q_i, since p_i is in the same cell and either directly below or directly to the left of q_i. Next, we show that there is a monotone path from q_i to p_{i+1}. Consider the monotone path from q_i to q_{i+1}, which is a subpath of Q. Recall that if q_i is on a vertical (resp. horizontal) boundary, then q_{i+1} is the first intersection of Q with a horizontal (resp. vertical) boundary. Therefore, the monotone path Q must intersect the left (resp. bottom) boundary of the cell that has q_{i+1} on its top (resp. right) boundary. Let the intersection of Q with this left (resp. bottom) boundary be r_i. Now, we have a monotone path from q_i to r_i, and there is a monotone path from r_i to p_{i+1}. Thus, there is a monotone path from p_i to q_i, to r_i, to p_{i+1}. Moreover, p_i is on a vertical (resp. horizontal) boundary, and p_{i+1} is on a horizontal (resp. vertical) boundary, and their cells share a y-coordinate (resp. x-coordinate). We have shown that there is a basic monotone path from p_i to p_{i+1}, completing our proof. □

Lemma 19 motivates us to build the following directed graph. Let $G = (V, E)$ be a graph where V is the set of critical points in the free space diagram, and E is the set of all pairs (p, q) such that there is a basic monotone path from p to q. Unfortunately, there are cases where a critical point has $O(n)$ outgoing edges, so that $|E| = O(n^3)$. Our goal will be to reduce the size of this graph, or rather, build essentially the same graph, but implicitly. To do this, we observe the following property of outgoing neighbours of a critical point.

Lemma 20. *Let p be the lowest free point on a vertical cell boundary. Let q be the rightmost critical point in the same row as p such that there is a basic monotone path from p to q.*

Then for any critical point r, there is a basic monotone path from p to r if and only if r is to the right of p, to the left of q, and has the same y-coordinate as q. See Figure 2.18.

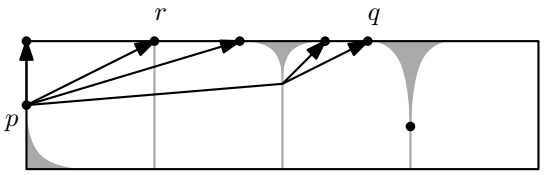


Figure 2.18: There is a basic monotone path pr if and only if r is to the right of p, to the left of q, and r has the same y-coordinate as q.

Proof. We first prove the "only if" direction. Suppose there is a basic monotone path from p to r. Then clearly r is to the right of p. Moreover, q is to the right of r since q is the rightmost critical point so that there is a basic monotone path from p to r. Finally, p, q and r are all on boundaries of cells that share a y-coordinate. Both q and r must be on the top boundary of their respective cells as there are basic monotone paths from p to q and r. Hence, q and r share the same y-coordinate. This completes the "only if" direction.

Next we prove the "if" direction. Suppose r shares the same y-coordinate as q, is to the right of p and to the left of q. Let L be the left boundary of the cell with r on its top boundary. Since p and q are in the same row of the free space diagram, and L is between p and q, the basic monotone path from p to q must intersect L at some point, which we will call s. So there is a monotone path from p to s and s to r, since s is on the left boundary and r is on the top boundary of the same cell. Moreover, this monotone path is basic since p is on a vertical boundary, r is on a horizontal boundary, and their cells share a y-coordinate. This completes the "if" direction. □

We also have a corollary for this lemma where the x and y-coordinates are switched.

Corollary 21. *Let p be the leftmost free point on a horizontal cell boundary. Let q be the topmost critical point in the same column as p such that there is a basic monotone path from p to q. Then for any critical point r, there is a basic monotone path from p to r if and only if r is above p, is below q, and has the same x-coordinate as q.*

We leverage Lemma 20 and Corollary 21 to build an improved graph that has fewer edges. We use binary trees as an intermediary between the start and end point of the edges in E. We construct a (directed) binary tree B_i for each horizontal line $h_0, h_1, \ldots h_{n_2}$ in the free space diagram. Every parent in B_i has a (directed) edge to its two children. The leaves of the binary tree are a sorted list of the critical points on h_i. Analogous binary trees are constructed for the vertical lines in the free space diagram. Now we describe the improved graph $G' = (V', E')$. Let V' be the union of the set of critical points in the free space diagram plus the set of internal vertices in the binary trees B_i for the horizontal and vertical lines in the free space diagram. For each critical point p on a vertical boundary, compute the rightmost critical point q so that there is a basic monotone path from p to q. In $G = (V, E)$, there is a directed edge from p to every critical point that is between p and q and on the horizontal line through q. Let the binary tree that corresponds to this horizontal line be B_i. We construct a directed edge from p to nodes of B_i so that the union of their descendants in B_i matches this set of contiguous critical points on h_i. We do so similarly

for the critical points p on horizontal boundaries. This completes the construction of the graph $G' = (V', E')$.

To decide whether the Fréchet distance between our two trajectories is at most d, we perform a depth first search in G' to decide whether there is a directed path from the bottom left corner to the top right corner. This completes the statement of the algorithm. Now we prove its correctness.

Lemma 22. *Given a pair of critical points a and b, there is a monotone path from a to b if and only if there is a directed path from a to b in the graph $G' = (V', E')$.*

Proof. By Lemma 19 and the definition of the graph G, there is a monotone path from a to b if and only if there is a directed path from a to b in the graph $G = (V, E)$. By Lemma 20 and Corollary 21, the outgoing neighbours of a critical point form a set of contiguous critical points on either a horizontal or vertical line in the free space diagram. Without loss of generality, suppose the set of contiguous critical points lie on h_i. By the definition of the binary trees B_i, we can convert a set of edges to this contiguous set of critical points into a set of paths down the binary tree B_i. Hence, there is a directed edge (p, q) in $G = (V, E)$ if and only if there is a directed path $(p_0, p_1, \ldots, p_k)$ in $G' = (V', E')$ where $p_0 = p$, $p_k = q$ and p_j is in a binary tree B_i for all $1 \leq j \leq k - 1$. Hence, there is a monotone path from a to b in the free space diagram if and only if there is a directed path from a to b in the directed graph $G' = (V', E')$. □

Finally, we analyse the running time of our algorithm. First we consider the running time of constructing the graph $G' = (V', E')$. Constructing the set of vertices V' takes $O(n_1 n_2)$ time. It remains to construct the set of edges E'. We first compute, for each critical point p, the rightmost (resp. topmost) critical point q where there is a basic monotone path from p to q.

Lemma 23. *Given a row of n_1 cells in a free space diagram, let $p_1, p_2, \ldots, p_{n_1+1}$ be the lowest free points on its vertical boundaries. Then we can compute, in $O(n_1 \log n_1)$ time, the set of critical points $q_1, q_2, \ldots, q_{n_1+1}$, so that for all $1 \leq i \leq k$, q_i is the rightmost critical point on a horizontal cell boundary such that there is a basic monotone path from p_i to q_i.*

Proof. Let the lowest free points on the vertical boundaries from left to right be $p_1, p_2, \ldots, p_{n_1+1}$. Let the corresponding highest free points on the same vertical boundaries be $r_1, r_2, \ldots, r_{n_1+1}$. Our algorithm is a dynamic program that considers the critical points p_i and r_i for decreasing values of i.

While performing the dynamic program on decreasing values of i, we maintain two lists, one for p_i and one for r_i. The list for p_i is of all p_j that have y-coordinate greater than the y-coordinates of $p_{i+1}, \ldots, p_{j-1}$. The list for r_i is of all r_j that have y-coordinate less than the y-coordinates of $r_{i+1}, \ldots, r_{j-1}$.

Both lists can be maintained in amortized constant time per update by storing the list as a stack. When a new vertical boundary is considered, we will add p_i and r_i to the top of the stack. To maintain the invariant that all p_j in the list must have greater y-coordinates than p_i, we pop off all elements on the top of the stack that have y-coordinate less than or equal to the y-coordinate of p_i before adding p_i. We maintain r_i analogously, but we check if the y-coordinate is greater than or equal.

Next, we use this pair of stacks to compute our dynamic program. The idea is that we construct a horizontal path starting at the critical point p_i and find the first non-free point it intersects. There are three cases. Either it reaches the rightmost vertical boundary of

the row, it intersects a point that is below p_j for some $j > i$, or it intersects a point that is above r_j for some $j > i$.

In the first case, the horizontal path intersects the rightmost vertical boundary. If $i < n_1 + 1$, then q_i is on the top boundary of the rightmost cell in the row. If $i = n_1 + 1$, then q_i does not exist.

In the second case, the horizontal line intersects a point that is below p_j. By our invariant, the critical point p_j must be in our stack. We locate p_j by computing the smallest index j such that the y-coordinate of p_j is at least the y-coordinate of p_i. For points to the right of p_j, our monotone path starting at p_i can only reach points where the monotone path starting at p_j can reach. Hence, we set q_i to q_j, which has previously been computing.

The third case is that the horizontal line intersects a point that is above r_j. By our invariant, the critical point r_j must be in our stack. Again, we locate r_j by computing the smallest index j such that the y-coordinate of r_j is at most the y-coordinate of p_i. Our monotone path starting at p_i can reach r_j, but cannot reach any points to the right of r_j. Hence, we can set q_j to be the leftmost free point on the top boundary of the cell that has r_j on its right boundary.

Maintaining the stacks takes $O(n_1)$ time. Performing the binary searches to find p_j in the first case and r_j in the second case takes $O(n_1 \log n_1)$ time in total. Hence, the overall running time for computing $q_1, q_2, \ldots, q_{n_1+1}$ is $O(n_1 \log n_1)$ time, as required. □

Lemma 23 allows us to compute all the edges E in $O(n_1 n_2 \log(n_1 + n_2))$ time. Next, we convert the edges E into the edges E'. We show that in the graph $G' = (V', E')$, there are at most $O(\log n_1)$ outgoing neighbours of p, and that these neighbours can be computed in $O(\log n_1)$ time.

Fact 24 (Chapter 5.1 of [61])**.** *Let B be a binary search tree with size $O(n)$. Suppose the leaves of B, from left to right, are a sorted list of real numbers. Then we can preprocess B in $O(n)$ time, so that given a pair of real numbers s and t, we can select $O(\log n)$ nodes of B in $O(\log n)$ time so that their descendants are those that lie in the interval $[s, t]$.*

Applying Lemma 23 and Fact 24 to each of the $O(n_1 n_2)$ critical points in the free space diagram leads to an $O(n_1 n_2 \log(n_1 + n_2))$ time algorithm for constructing all edges in the graph $G' = (V', E')$. Moreover, the size of E' is $O(n_1 n_2 \log(n_1 + n_2))$. Finally, running the depth first search takes $O(|V'| + |E'|) = O(n_1 n_2 \log(n_1 + n_2))$. This yields the following theorem.

Theorem 25. *Given a pair of trajectories of complexities n_1 and n_2, there is an $O(n_1 n_2 \log(n_1 + n_2))$ time algorithm that solves the Fréchet distance decision problem by running a depth first search algorithm on the set of critical points in the free space diagram.*

2.5.2 Reference subtrajectory is vertex-to-vertex

Our approach is to run the sweepline algorithm in Section 2.4.2, but we replace the discrete free space diagram (i.e. the n^2 grid points) with the graph $G' = (V', E')$ defined in Section 2.5.1. For this we require three modifications to the sweepline algorithm.

The first modification is to generalise the greedy aspect of the depth first search to the new graph. In the discrete free space diagram, we first explore $(x+1, y)$, then $(x+1, y+1)$, and finally $(x, y+1)$. In the graph $G' = (V', E')$, we explore the neighbours with minimum y-coordinate first, and of those with the same y-coordinate, we explore those with maximum x-coordinate first.

The second modification is to create additional nodes in $G' = (V', E')$ for the ending points of the monotone paths P_i. The ending point is the lowest point on l_t such that there is a monotone path to that point. However, this lowest point on l_t may not be a node in G'. We can detect this case by checking if the last critical point before reaching l_t, say p, has a basic monotone path through l_t. In this case the ending point is simply the intersection of l_t with a horizontal line through p. We add this intersection to the graph G', and calculate its outgoing neighbours.

The third modification is to create additional nodes in $G' = (V', E')$ for the starting points of the monotone paths P_i. As usual, the starting point of P_{i+1} is the lowest free point on l_s that has a y-coordinate greater than or equal to the maximum y-coordinate of P_i. However, this lowest free point may not be a node of G'. If it is not, we add it to G' and calculate its outgoing neighbours.

All additional nodes created in the second and third modifications are not initially part of the graph $G' = (V', E')$, and are only added to G' when necessary. Hence, the graph G' increases in size as the sweep line algorithm is performed.

A special case that is closely related to the second and third modifications is to detect if there are infinitely many horizontal monotone paths between l_s and l_t. We use the same method as Buchin et al. [36] to detect if adding any of these additional nodes to $G' = (V', E')$ creates infinitely many horizontal monotone paths, in which case we return a set of $m - 1$ monotone paths.

This completes the statement of our algorithm. Next, we argue its correctness.

Lemma 26. *There exist $m - 1$ monotone paths satisfying the conditions of SC under the continuous Fréchet distance in the case that the reference subtrajectory is vertex-to-vertex if and only if our algorithm returns a set of $m - 1$ monotone paths.*

Proof. Observe that Lemma 15 generalises from the discrete free space diagram to the graph $G' = (V', E')$. In particular, we add a link between a pair of critical points in the continuous free space diagram only if there is a directed path between them in G'. So there is always a monotone path from any critical point to the root of its link-cut tree.

Observe that Lemma 16 generalises to G'. In particular, by our first modification, our algorithm prefers to link to its lower neighbours first, and the remainder of the proof is identical to the proof of Lemma 16.

Finally, we observe that Lemma 17 generalises to G'. The proof of both the if and only if directions are identical, so long as we take into account the additional nodes from the second and third modifications. These additional nodes can be treated exactly the same as any other critical point in G', other than that they require additional time to compute. By generalising Lemma 17 to the continuous free space diagram, we yield the Lemma. □

It remains only to analyse the running time.

Theorem 27. *There is an $O(n^2 \log^2 n)$ time algorithm that solves SC under the continuous Fréchet distance in the case that the reference subtrajectory is vertex-to-vertex.*

Proof. First, we analyse the running time of constructing G'. By Lemmas 23 and 24, we can construct G' in $O(n^2 \log n)$ time. Next, we analyse the running time of the sweepline algorithm. This running time is dominated by two processes, maintaining the link-cut tree and inserting the additional nodes for the second and third modifications.

First, we bound the number of additional nodes, and the time required to insert them. There are at most $m - 1$ paths per reference subtrajectory we consider, and we consider

$O(n)$ reference subtrajectories. Therefore, we add at most $O(mn)$ additional points to the graph G'. Inserting these additional points requires $O(mn)$ time. Inserting the outgoing edges for each of these points requires amortised $O(\log n)$ time per edge by Lemmas 23 and 24, which is $O(mn \log n)$ time in total

Next, we analyse the running time of maintaining the link-cut tree. For this, the running time is dominated by updating the data structure. There are $O(n^2 \log n)$ edges in G', and $O(mn \log n)$ edges for the additional points. Each update, which is either a link or cut operation, requires $O(\log n)$ amortised time. We observe that, similarly to in Section 2.4.2, once a link is removed it cannot be added again. So all link and cut updates can be performed in $O(n^2 \log^2 n)$ time. This dominates the running time of our algorithm, yielding the theorem. □

2.5.3 Reference subtrajectory is arbitrary

Next, we handle the case where the reference subtrajectory may start and end at arbitrary points on the trajectory. Although there are infinitely many possible starting and ending points, we only consider starting and ending points associated to either vertices of the trajectory, or to additional critical points that we call internal critical points.

We define an external critical point to be critical points that lie on the boundary of a free space cell. In contrast, an internal critical point is in the interior of a cell, and is defined as follows:

Definition 28. *We define an internal critical point to be a point that is in the interior of a cell, on the boundary between free and non-free space, and satisfies one of the following three conditions:*

- *it is the leftmost or rightmost free point in its cell, or*
- *it shares a y-coordinate with an external critical point, or*
- *it is ℓ units horizontally to the right of a point that is on the boundary between free and non-free space.*

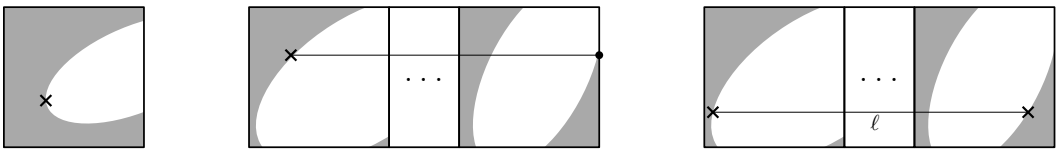


Figure 2.19: The three types of internal critical points.

In Figure 2.19 we show the three types of interior critical points. Both the vertices of the trajectory and the internal critical points are candidate starting points for the reference subtrajectory. To decide whether any such subtrajectory satisfies the properties in SC, we would like to perform the same sweepline algorithm as the one in Section 2.5.2. Recall that this algorithm iterates through all reference subtrajectories, and reuses monotone paths between these subproblems using a link-cut data structure. We provided the details for maintaining the link-cut data structure in Section 2.4.2. Recall that three modifications

were applied to our sweepline algorithm so that it may be applied to the graph $G' = (V', E')$. We provided the details of these three modifications in Section 2.5.2.

However, if we were to perform the same sweepline algorithm as in Section 2.5.2, our running time would increase to accommodate the additional internal critical points and their reference subtrajectories. In Lemma 34, we show that there are $O(n^3)$ internal critical points. Therefore, the running time for maintaining the link-cut tree would increase to $O(n^3 \log^2 n)$. The running time for inserting the additional nodes for the second and third modifications would increase to $O(n^3 m)$.

To perform the sweepline algorithm in $O(n^3 \log^2 n)$ time, we avoid computing the $O(n^3 m)$ additional nodes for the second and third modifications entirely. We use a completely different approach. Our intuition is that if a monotone path exists for a reference subtrajectory, then either the same or a very similar monotone path is likely to exist for the next reference subtrajectory. We divide the process of computing $m - 1$ non-overlapping monotone paths into two steps. The first step is to maintain a large set of overlapping monotone paths, so that any monotone path is represented within this set. The second step is to query this set of overlapping monotone paths to decide whether there are $m - 1$ elements that do not overlap. Surprisingly, building and maintaining this set of overlapping monotone paths is more efficient than recomputing the non-overlapping monotone paths for each reference subtrajectory.

Our set of overlapping monotone paths is maintained by the following dynamic data structure.

Fact 29 (Theorem 16 in [90])**.** *A dynamic monotonic interval data structure maintains a set of monotonic intervals. A set of intervals is monotonic if no interval contains another. The data structure offers the following three operations. Each operation can be performed in $O(\log n)$ amortised time.*

- *Insert an interval, so long as the monotonic property is maintained,*
- *Remove an interval,*
- *Report the maximum number of non-overlapping intervals in the data structure.*

With this data structure in mind, we are now ready to state our algorithm in full. Compute all external critical points and build the graph $G' = (V', E')$ defined in Section 2.5.1. Our sweepline algorithm starts with $s = 0$. For each external critical point g on l_s, we perform a greedy depth first search to find the lowest point r on l_t so that there is a monotone path from g to r. We maintain the link-cut data structure throughout the greedy depth first search, so that r is the root of the tree containing g. For $s = 0$, we only compute a set of additional external critical points, that is, all points on l_s that share a y-coordinate with another external critical point. For these external critical points, we compute its lowest monotone path to l_t, so that the root of the tree containing the external critical point is on l_t.

For $s = 0$, we now have a set of link-cut trees, each of which has their root on l_t. For each root r on l_t, compute its highest descendant g on l_s. This highest descendant can be maintained by each link-cut tree individually, so that when two link-cut trees are merged, we simply take the higher descendant for the merged tree. Finally, for each root r on l_t, and its highest descendant g on l_t, we insert the interval $(y(g), y(r))$ into the dynamic monotonic interval data structure, where $y(g)$ and $y(r)$ denote the y-coordinates of g and r respectively. See Figure 2.20. This completes the base case of $s = 0$ in the sweepline algorithm.

Next, compute all internal critical points defined in Definition 28, and sort them by x-coordinate. We sweep the vertical lines l_s and l_t from left to right, and whenever l_s or l_t

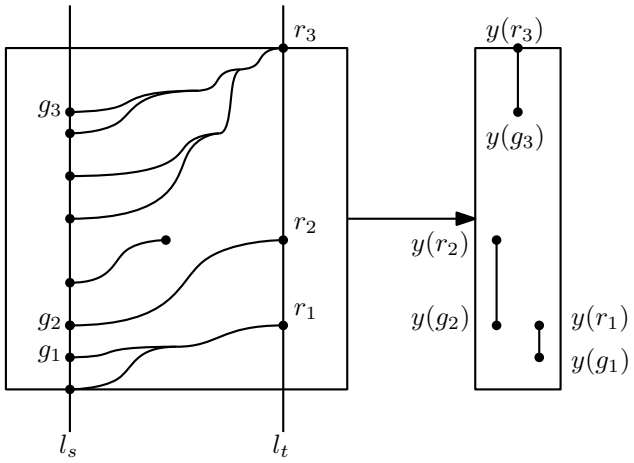


Figure 2.20: The link cut tree rooted at r_i, with its highest descendant g_i on l_s, shown on the left. The y-intervals $(y(g_i), y(r_i))$ stored by the dynamic monotonic interval data structure, shown on the right.

pass through an internal critical point, we process it as an event. There are five types of events, depending on the type of the internal critical point and whether it passes through l_s or l_t. On all five of these events, we maintain the invariant that the intervals inserted into our data structure are monotonic. We also maintain the invariant that any monotone path is represented by an interval in the data structure. So that we do not need to re-compute the exact y-coordinates of these monotone paths at every event point, we only store the relative positions of the intervals with respect to one another (i.e. whether they are overlapping or non-overlapping). By only storing the relative positions of the y-coordinates, we can avoid computing the $O(n^3m)$ starting and ending points of monotone paths in the second and third modifications that are required for the algorithm in Section 2.5.2. We split our analysis of our five types of events into five cases:

- The first type of event is if l_s passes through an internal critical point g that is the leftmost free point in its cell. For this event, insert g into the graph G', and compute the lowest monotone path from g to the l_t. If a jump operation was performed using the link-cut data structure, then the root r of the tree containing g already exists in the dynamic monotonic interval data structure. We only update this interval to $(y(g), y(r))$ if g is the highest descendant of r on l_s. If no jump operation was performed, then r is a new root on l_t, so we simply insert the interval $(y(g), y(r))$ into our data structure.
- The second type of event is if l_s passes through an internal critical point g that shares a y-coordinate with an external critical point. Similarly to the first event, we insert the g into the graph G', and compute its root r on the line l_t. We insert a new interval $(y(g), y(r))$ into the dynamic monotonic interval data structure if r is new, or replace an existing interval if r is not new, but g is the highest descendant on l_s. After this, we consider whether l_s passing through g causes a pair of monotone paths that were previously overlapping to now be non-overlapping. In particular, suppose that g shares a y-coordinate with an exterior critical point, which in turn shares a y-coordinate with a root r' on l_t. In other words, $y(g) = y(r')$. Then the pair of intervals $(y(g), y(r))$ and $(y(g'), y(r'))$ may switch from overlapping to non-overlapping, or vice versa. If

this is the case, we remove the interval $(y(g), y(r))$ and replace it with a new interval so that their relative positions are g', r', g, r instead of g', g, r', r, or vice versa.

- The third type of event is if l_t passes through an internal critical point r that is the rightmost free point in its cell. For this event, simply remove the interval (g, r) from the dynamic monotone interval data structure, where g is the highest ancestor of r that is on l_s.
- The fourth type of event is if l_t passes through an internal critical point r that shares a y-coordinate with an external critical point. Let the highest descendant of r on l_s be g. We consider whether l_t passing through r makes the monotone path from g to r invalid. This could be the case if the last segment in this monotone path is horizontal, and if r lies on a boundary between free and non-free space that has a negative gradient. If this is the case, then we remove the invalid path from r to its child, and recompute its new highest descendant on g, if one exists. Next, similar to the second event, we consider whether l_t passing through r causes a pair of monotone paths that were previously non-overlapping to now be overlapping. In particular, suppose that r shares a y-coordinate with an exterior critical point, which in turn shares a y-coordinate with a root g' on l_s. In other words, $y(r) = y(g')$. Then the pair of intervals $(y(g), y(r))$ and $(y(g'), y(r'))$ may switch from overlapping to non-overlapping, or vice versa. If this the case, we remove the interval $(y(g), y(r))$ and $(y(g'), y(r;))$ and replace it with a new interval so that their relative positions are g, r, g', r' instead of $g, g', r'r'$, or vice versa.
- The fifth type of event is if l_s passes through a point g' while l_t simultaneously passes through a point r, so that both g' and r are on the boundaries between free and non-free space, and $y(g') = y(r)$. We consider whether l_s and l_t passing through g' and r causes a pair of monotone paths that were previously non-overlapping to now be overlapping. In particular, the pair of intervals $(y(g), y(r))$ and $(y(g'), y(r'))$ may switch from overlapping to non-overlapping, or vice versa. If this is the case, we remove the interval $(y(g), y(r))$, and replace it with a new interval so that their relative positions of g', r', g, r instead of g', g, r', r, or vice versa.

After processing an event, we report whether there are $m-1$ monotone paths for this event and its associated reference subtrajectory. To do this, we query the dynamic monotone interval data structure to report the maximum number of non-overlapping intervals in the data structure. If there are $m-1$ or more non-overlapping intervals, we report these $m-1$ intervals. We can retrieve the original monotone paths by storing the monotone paths with the intervals when we insert them. We can modify the monotone paths to start and end at l_s and l_t in constant time per monotone path.

This completes the statement of our algorithm. Next, we prove its correctness.

Definition 30. *Given s and t, a monotone path from g on l_s to r on l_t is called minimal if there does not exist a monotone path from g' on l_s to r' on l_t such that the interval $[y(g'), y(r')]$ is a strict subset of the interval $[y(g), y(r)]$.*

Lemma 31. *Given s and t, and a monotone path from g on l_s to r on l_t, there exists a minimal monotone path from g' on l_s to r' on l_t so that $[(y(g'), y(r')] \subseteq [(y(g), y(r))]$.*

Proof. Let r' be the lowest point on l_t so that there is a monotone path from g to r'. We know r' exists since we can compute it with a greedy depth first search. Let g' be the highest point on l_s so that there is a monotone path from g' to r'. We have by definition

that $y(g) \leq y(g') \leq y(r') \leq y(r)$, so $[(y(g'), y(r')] \subseteq [(y(g), y(r))]$. It suffices to show that the monotone path from g' to r' is minimal.

Suppose the monotone path from g' to r' is not minimal. Then there exists a monotone path from g'' to r'' so that $[(y(g''), y(r'')] \subset [(y(g'), y(r'))]$. But now, the monotone paths from g'' to r'' and from g' to r' must cross at some point u. Construct the monotone path from g'' to u to r'. But g' is the highest point on l_s so that there is a monotone path from g' to r', so $y(g'') \leq y(g')$. But $y(g') \leq y(g'')$ since $[(y(g''), y(r'')] \subset [(y(g'), y(r'))]$. So $y(g'') = y(g')$.

Therefore, there exists a monotone path from g' to r'' such that $y(r'') < y(r')$. But now, the monotone path from g' to r'' and the monotone path from g to r must cross at some point u. Construct the monotone path g to u to r''. Since $y(r'') < y(r')$, this contradicts the construction that r' is the lowest point on l_t such that there is a monotone path from g to r'. Hence, our initial assumption that g' to r' is not minimal cannot hold, and we are done. □

Lemma 32. *Given s and t, suppose we run our sweepline algorithm until our pair of sweeplines reach l_s and l_t respectively. Then there is an order preserving bijection from the y-intervals of the minimal monotone paths to the set of intervals in our dynamic monotonic interval data structure.*

Proof. Our proof is divided into three parts. First we prove an order preserving bijection in the base case, where $s = 0$. Next, we prove that in the inductive case, if the sweepline does not pass through any internal critical points, then the order preserving bijection is preserved. Finally, we prove that as the sweepline passes through an internal critical point, the order preserving bijection is preserved.

For the base case, consider when $s = 0$. Let g be on l_s and r be on l_t so that the path from g to r is a minimal monotone path. Suppose g is not an external critical point. Then there is a segment of free space directly above g. We show that the monotone path from g to r consists of a horizontal path from g to an external critical point. Suppose the contrary, that the initial path from g is not horizontal. Let the initial path be from g to h, so that $y(h) > y(g)$. Then if we selected the point on l_s with y-coordinate $y(g) + \varepsilon$ for a sufficiently small ε, then there would still have $y(h) > y(g) + \varepsilon$, and we would maintain our monotone path. But now, the new path would have a y-interval from $y(g) + \varepsilon$ to $y(r)$, which is a strict subset of the y-interval from $y(g)$ to $y(r)$, contradicting the fact that the monotone path from g to r is minimal. Therefore, if g is not an external critical point, then there is a horizontal path from g to its next point, which must be an external critical point. Therefore, all minimal monotone paths start at either external critical points or these additional points that share a y-coordinate with an external critical point. By the definition of our base case in our algorithm, we insert all y-intervals for these potential starting points, so we have all minimal monotone paths in our initial data structure. Hence, we have shown the base case of our induction.

For the inductive case where the sweepline does not pass through an internal critical point, suppose the inductive hypothesis that l_s and l_t are sweeplines for which there is an order preserving bijection. Let $l_{s'}$ and $l_{t'}$ be another pair of sweeplines to the right of l_s and l_t, so that there are no internal critical points between l_s and $l_{s'}$, and similarly between l_t and $l_{t'}$. We will show that there is a bijection between the y-intervals of the minimal monotone paths between l_s and l_t, and $l_{s'}$ and $l_{t'}$. By composing this bijection with the bijection between the minimal monotone paths between l_s and l_t and the data structure,

we yield a bijection between the minimal monotone paths between $l_{s'}$ and $l_{t'}$ and the data structure.

We construct our bijection between the two sets of minimal monotone paths as follows. For each minimal monotone path starting at g on l_s and ending at r on l_t, we perform an operation to yield a minimal monotone path between $l_{s'}$ and $l_{t'}$. First, we let g' be the highest point on $l_{s'}$ such that there is a monotone path from g to g' and a monotone path from g' to r. Second, we extend the monotone path horizontally from r to its right, until it either hits $l_{t'}$, or the boundary between free and non-free space. We extend the monotone path along the boundary between free and non-free space until we reach $l_{t'}$. When this path reaches $l_{t'}$, we define this point to be r'. This completes the description of the mapping from minimal monotone paths of l_s and l_t to paths between $l_{s'}$ and $l_{t'}$.

Our proof of this bijection is divided into five parts. First, we show that the mapping is well defined. Second, we show that the mapping is from minimal monotone paths to minimal monotone paths. Third, we show that the mapping is injective. Fourth, we show that the mapping is surjective. Fifth, we show that the mapping is order preserving.

First, we show that the mapping is well defined. The point g' is well defined. Extending r horizontally is well defined, however, it may be that r cannot be extended along the boundary between free and non-free space until we reach $l_{t'}$. There are two ways this can occur. First, the free space may stop at some x-coordinate before $l_{t'}$. However, in this case, we have an internal critical point that is the rightmost free point in its cell. But we assumed there were no such critical points between l_t and $l_{t'}$. Second, the boundary between free space and non-free space may have a negative gradient, so that the monotone path cannot travel along it. In this case, we must have a horizontal path starting at r that is extended towards the boundary with a negative gradient. If the point r does not share a y-coordinate with an external critical point, then we can reduce the y-coordinate of the horizontal path containing r, contradicting the fact that g to r is minimal. Hence, r shares a y-coordinate with an external critical point, so does our horizontal path that intersects the boundary between free and non-free space. Therefore, this point is an internal critical point, which again contradicts our assumption that there are no such critical points between l_t and $l_{t'}$.

Second, we show that the mapping is from minimal monotone paths to minimal monotone paths. By definition, r' is the lowest point on $l_{t'}$ such that there is a monotone path from g' to r'. We will show that g' is the highest point on l_s so that there is a monotone path from g' to r'. Putting these two facts together yields that the monotone path from g' to r' is minimal. Suppose for the sake of contradiction that there is another point, g'', that has a larger y-coordinate than g', and there is a monotone path from g'' to r'. Extend the monotone path starting at g'' horizontally to the left, and then along the boundary between free and non-free space, until it reaches l_s. For the same reason as extending r horizontally to the right to reach $l_{t'}$, we must be able to extend g'' horizontally to the left to reach l_s, as there are no internal critical points between l_s and $l_{s'}$. We show that the extension of g'' meets l_s at g. If $y(g'') > y(g)$, this would contradict the minimality of the monotone path from g to r. If $y(g'') < y(g)$, then the extension of g'' crosses the path from g' to g, which is impossible. Hence, we have monotone paths from g to g'' to r where g'' has a larger y-coordinate of g'. This contradicts the definition of g', so the point g'' cannot exist, as required.

Third, we show the mapping is injective. Suppose for the sake of contradiction that there are two monotone paths, g to r and g' to r', from l_s to l_t, that both map to a monotone path g'' to r'', from $l_{s'}$ to $l_{t'}$. Then there are monotone paths from g to g'' to r and from g' to g'' to r'. Without loss of generality, suppose $y(g') \geq y(g)$. Then by the minimality of

the path from g to r, we get that $y(r') \geq y(r)$. But now, consider the monotone path from g to g'' to r'. By the minimality of g to r, we get that $y(r') \leq y(r)$, so $y(r') = y(r)$. By the minimality of g' to r', we get that $y(g') \leq y(g)$, so $y(g') = y(g)$. Hence, $g = g'$ and $r = r'$, so our mapping is injective.

Fourth, we show the mapping is surjective. Suppose for the sake of contradiction that there is a monotone path between $l_{s'}$ and $l_{t'}$ that is not mapped to. Let this minimal monotone path be from g' to r'. Let r be the highest point on l_t such that there is a monotone path from g' to r to r'. Let g be the highest point on l_s such that there is a monotone path from g to g' to r. We claim that our construction maps the path g to r to the path g' to r'. First, we show that g' is the highest point on $l_{s'}$ such that there is a monotone path from g to g' to r. Suppose there is a higher point g'' so that there is a monotone path from g'' to r. Then we have a monotone path from g'' to r', contradicting the minimality of the path from g' to r'. Next, we show that if we extend the monotone path from r horizontally to the right, then we reach the point r' on $l_{t'}$. Suppose that instead we reach another point r''. By the definition of our extension, r'' is the lowest point on $l_{t'}$ such that there is a monotone path from r to r''. So $y(r'') \leq y(r')$. But by the minimality of the path g' to r', we get that $y(r'') \geq y(r')$. Hence, $r' = r''$ as required, and the mapping is surjective.

Fifth, we show that the mapping is order-preserving. Suppose there are paths g to r and g' to r', both between l_s and l_t. Without loss of generality, let $y(g) < y(g')$. Suppose these monotone paths map to g'' to r'' and g''' to r''' respectively. We show that $y(g'') < y(g''')$. Suppose for the sake of contradiction that $y(g'') \geq y(g''')$. If $g'' = g'''$, by minimality we would have $r'' = r'''$, contradicting the injectivity of our mapping. So $y(g'') > y(g''')$. Therefore, the path from g to g'' and the path from g' to g''' must cross at some point u. But now we have a monotone path from g to u to g''', which contradicts the fact that g'' is the highest point on $l_{s'}$ such that there is a path from g to g'' to r. Hence, $y(g'') < y(g''')$ as required. Finally, we show that $y(g') < y(r)$ if and only if $y(g''') < y(r'')$. Suppose for the sake of contradiction that $y(g') < y(r)$ and $y(g''') > y(r'')$. Take the path from r to r'' that is a horizontal line to the right, until it reaches the boundary between free and non-free space, and take the path along this boundary. Similarly, take the path from g''' to g' that is a horizontal line to the left, until it reaches the boundary between free and non-free space, and take the path along this boundary. For any pair of vertical lines l_a between l_s and $l_{s'}$ and l_b between l_t and $l_{t'}$, such that $l_b - l_a = \ell$, define a to be the intersection of the path between g''' and g' with l_a, and define b to the intersection of the path between r and r'' with l_b. When $l_a = l_s$, we have $y(a) < y(b)$. When $l_s = l_{s'}$, we have $y(a) > y(b)$. So by the intermediate value theorem, there must be a point where $y(a) = y(b)$. This cannot occur when both a and b are on the horizontal portions of their respective paths. If they are both on the boundary between free and non-free space, then a and b are interior critical points, where a is horizontally ℓ units to the right b. If one of a or b is on the boundary between free and non-free space, and the other is on the horizontal portion, then we also have an internal critical point, as there is a point on the boundary between free and non-free space sharing a y-coordinate with an external critical point. Putting this all together, we yield a contradiction in the case where $y(g') < y(r)$ and $y(g''') > y(r'')$. We yield a similar contradiction in that case where $y(g') > y(r)$ and $y(g''') < y(r'')$. This shows that our bijection is indeed order preserving. This completes the proof of the inductive case where the sweepline does not pass through an internal critical point.

For the inductive case where the sweepline passes through an internal critical point, we show that the bijection between the set of y-intervals and the set of minimal monotone

paths is preserved. If the internal critical point is the leftmost free point in the cell, there is one extra starting point for a minimal monotone path to be considered. If an additional minimal monotone path exists due to this point, our algorithm adds it to our data structure. Similarly, if the internal critical point is the rightmost free point in the cell, there is one fewer ending point for minimal monotone paths, and a y-interval is deleted from our data structure if necessary. If our internal critical point is a point on the boundary between free and non-free space that shares a y-coordinate with an external critical point, we consider the two potential modifications to the set of minimal monotone paths. First, if the internal critical point is on l_s, we consider whether this point is the starting point of a new minimal monotone path. If the internal critical point is on l_t, we consider whether this point is the last valid point for a minimal monotone path that needs to be removed. Second, we consider whether there may be a swap in relative positions of the y-coordinates in minimal monotone paths. If this is the case, we replace the previous y-coordinates with new y-coordinates in the data structure. Finally, if the internal critical point is a pair of points on the boundary between free and non-free space that are ℓ units horizontally away from one another, we consider whether this internal critical point changes the relative positions of the y-coordinates of minimal monotone paths, and update the data structure accordingly. Note that if the internal critical point is a leftmost free point or rightmost free point, this only adds or deletes minimal monotone paths, since the critical point is local to either l_s or l_t. In contrast, if the internal critical point is a pair of points on the boundaries of free and non-free space that are ℓ units horizontally from one another, this only affects the relative positions of y-coordinates of minimal monotone paths, so it suffices to consider swapping these relative positions in the data structure. To summarise, in all cases we update the data structure and preserve the relative positions of the y-intervals of the minimal monotone paths in our data structure. This completes the proof of our lemma. □

Lemma 33. *There exist $m-1$ monotone paths satisfying the conditions of SC under the continuous Fréchet distance if and only if our algorithm returns a set of $m-1$ monotone paths.*

Proof. Suppose our algorithm returns a set of $m-1$ monotone paths. We have already shown in Section 2.5.2 that the monotone paths computed by the greedy depth first search and link-cut data structure on $G' = (V', E')$ result in valid paths. These monotone paths are non-overlapping due to the order preserving bijection in Lemma 32. Hence, our reference subtrajectory plus our $m-1$ monotone paths forms a subtrajectory cluster that satisfies the conditions of SC.

Suppose there exist $m-1$ monotone paths satisfying the conditions of SC. By Lemma 31 there exist $m-1$ minimal monotone paths satisfying the conditions of SC. By Lemma 32 there is an order preserving bijection from the y-intervals of these $m-1$ minimal monotone paths to the set of intervals in our dynamic monotonic interval data structure. Hence, when our algorithm reaches the reference subtrajectory that satisfies the conditions of SC, our algorithm will report that there are $m-1$ non-overlapping intervals in the data structure, as required. □

Finally, we perform a running time analysis on our algorithm. We begin by bounding the number of internal critical points.

Lemma 34. *There are $O(n^3)$ internal critical points.*

Proof. If the internal critical point is the leftmost free point in its cell, there is only one copy of it in its cell, so there are at most $O(n^2)$ internal critical points of this type.

If the internal critical point shares a y-coordinate with an external critical point, and is on the boundary between free and non-free space, there are only two copies of it per external critical point and per cell in the same row as it. In total, there are $O(n^2)$ external critical points, and $O(n)$ cells in the same row as it, so there are $O(n^3)$ internal critical points of this type.

If the internal critical point is ℓ unites horizontally to the right of a point that is on the boundary between free and non-free space, there is at most a constant number of copies per pair of cells in the same row. This is because, for any pair of cells in the same row as each other, a ℓ-unit horizontal translation of its free space boundary would only intersect the free space boundary of the other cell a constant number of times. As there are $O(n^2)$ pairs of cells that share a row, there are $O(n^2)$ internal critical points of this type. □

Now we can analyse the running time of our algorithm to obtain the main result of Section 2.5.

Theorem 4. *There is an $O(n^3 \log^2 n)$ time algorithm for SC under the continuous Fréchet distance.*

Proof. Computing the graph $G' = (V', E')$ requires $O(n^2 \log n)$ time. Computing all internal critical points takes $O(n^3)$ time, by computing each critical point individually in constant time. Sorting them by x-coordinate requires $O(n^3 \log n)$ time. At each of the $O(n^3)$ events, we may insert the critical point into the graph G'. Inserting a single critical point inserts up to $O(\log n)$ edges into the graph G'. In total, there are $O(n^3 \log n)$ edges in the graph G'. Each edge can be linked or cut at most once, so by the same amortised analysis as in Section 2.4.2, we spend at most $O(n^3 \log^2 n)$ time updating the link-cut data structure.

We maintain the dynamic monotonic interval data structure by inserting, deleting, or replacing (both a delete and an insert) an interval, which requires $O(\log n)$ time per event. In total, maintaining the dynamic monotonic interval data structure requires $O(n^3 \log n)$ time.

In total, the running time is dominated by updating the link-cut data structure, and the running time of our algorithm is $O(n^3 \log^2 n)$. □

2.6 Lower bound

The main theorem we will prove in this section is the following:

Theorem 7. *There is no $O(n^{3-\varepsilon})$ time algorithm for SC under the continuous Fréchet distance, for any $\varepsilon > 0$, unless SETH fails.*

We reduce from the 3OV problem to SC. The formal definition of 3OV is as follows:

Problem 34 (3OV)**.** *We are given three sets of vectors $\mathcal{X} = \{X_1, X_2, \ldots, X_n\}$, $\mathcal{Y} = \{Y_1, Y_2, \ldots, Y_n\}$ and $\mathcal{Z} = \{Z_1, Z_2, \ldots, Z_n\}$. For $1 \leq i, j, k \leq n$, each of the vectors X_i, Y_j and Z_k are binary vector of length W. Our problem is to decide whether there exists a triple of integers $1 \leq i, j, k \leq n$ such that X_i, Y_j and Z_k are orthogonal. The three vectors are orthogonal if $X_i[h] \cdot Y_j[h] \cdot Z_k[h] = 0$ for all $1 \leq h \leq W$.*

We employ a three step process in our reduction. We first provide an overview of each step, then we outline the structure of this section in relation to these three steps.

- Step 1: Given a 3OV instance $(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$, we construct in $O(nW)$ time an SC instance (T, m, ℓ, d) of complexity $O(nW)$. We construct T by constructing two subtrajectories T_1 and T_2 and connecting them by a point. We encode the booleans from the vector set $\mathcal{Z}$ into the subtrajectory T_1, and we encode the booleans from the vectors sets $\mathcal{X}$ and $\mathcal{Y}$ into the subtrajectory T_2. The subtrajectories T_1 and T_2 are analogous to the subtrajectories in the overview section under the same name, however, the subtrajectories in the full construction is significantly more complex.
- Step 2: We consider the free space diagram $F_d(T, T)$ and prove various properties of it. One of the key properties is the existence of antipodes. In the first key component of our overview in Section 2.3.2, we briefly described these antipodes, and how these antipodes determine the positions of diamond-shaped non-free regions in the free space diagram. Another key property is the structure of $F_d(T_2, T_1)$, which we show is mostly of free-space, with all non-free space associated with antipodes. The final key property are the positions of the vertical lines l_s and l_t that we will consider in our reduction. In the third key component in Section 2.3.2, we mentioned that our reduction considers n^2 reference subtrajectories. We describe the positions of n^2 pairs of vertical lines l_s and l_t, so that l_s passes through multiple critical points. Moreover, we show that all of our n^2 reference subtrajectories have the same length.
- Step 3: We use the properties of the free space diagram $F_d(T, T)$ to prove that $(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$ is a YES instance if and only if (T, m, ℓ, d) is a YES instance. If $(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$ is a YES instance, we construct a set of monotone paths in $F_d(T, T)$ to show that (T, m, ℓ, d) is a YES instance. If $(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$ is a NO instance, we define sequences which we call cutting sequences. We show that if there is a cutting sequence between a pair of points in $F_d(T, T)$, there is no monotone path between the same pair of points. We then use these cutting sequences to show that if $(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$ is a NO instance, then (T, m, ℓ, d) is a NO instance.

This rest of this section is structured as follows. In Section 2.6.1, given an instance $(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$ of 3OV, we construct an SC instance (T, m, ℓ, d) of complexity $O(nW)$. In Section 2.6.2 we fill in some missing details in our construction, in particular, we show that certain points in our construction are well defined. This completes Step 1. In Section 2.6.3 we identify the antipodal pairs in our construction. In Section 2.6.4 we identify the starting and ending points of our n^2 reference subtrajectories, and prove that each reference subtrajectory has length ℓ. In Section 2.6.5 we describe the free space diagram of our construction, and show that all non-free regions are associated with an antipodal pair. This completes Step 2. In Section 2.6.6, we construct a set of monotone paths in the free space diagram. In Sections 2.6.7, we use this set of monotone paths to prove that if our input $(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$ is a YES instance for 3OV, then our construction (T, m, ℓ, d) is a YES instance for SC. In Section 2.6.8, we define and construct a set of sequences that show that there are is no monotone path between certain points. In Section 2.6.9, we use this set of sequences to show that if our input $(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$ is a NO instance for 3OV then our construction (T, m, ℓ, d) is a NO instance for SC. Finally, in Section 2.6.10, we put it all together and prove Theorem 7. This completes Step 3.

2.6.1 Construction

Recall that as input we are given a 3OV instance, that is, three sets $\mathcal{X}$, $\mathcal{Y}$, $\mathcal{Z}$ of n binary vectors of length W. Given this instance, we construct an instance (T, m, ℓ, d) for SC. We

construct the trajectory T by constructing two subtrajectories T_1 and T_2 and connecting them via a point.

Let r and r' be positive real numbers so that $10r < r'$. We use polar coordinates in the complex plane. Recall that $r \operatorname{cis} \theta$ is a point that is r units away from the origin, at an angle of θ anticlockwise from the positive real axis. Define $\phi = \frac{\pi}{4W+6}$ and $d = ||r \operatorname{cis} \pi - r' \operatorname{cis} \phi||$, where $||\cdot||$ denotes the Euclidean norm. Define $\delta = (r + r' - d)$. Let $\varepsilon > 0$ be arbitrarily small relative to δ and r.

Define the following points which will be used to define T_1. See Figure 2.21.

$$
\begin{array}{lcl}
A_{h,u} & = & r \operatorname{cis}((2W+3+u)\cdot\phi), \text{ if } u \neq 2h \\
A_{h,u} & = & (r - \frac{2}{3}\delta) \operatorname{cis}((2W+3+u)\cdot\phi), \text{ if } u = 2h, \\
A_{W+1,u} & = & r \operatorname{cis}((2W+3+u)\cdot\phi) \\
B_{k,h} & = & r \operatorname{cis}((4W+5)\cdot\phi), \text{ if } Z_k[h] = 1 \\
B_{k,h} & = & (r-\varepsilon) \operatorname{cis}((4W+5)\cdot\phi), \text{ if } Z_k[h] = 0 \\
B_{k,W+1} & = & r \operatorname{cis}((4W+5)\cdot\phi) \\
C & = & r \operatorname{cis}((4W+6)\cdot\phi) \\
D & = & r \operatorname{cis} 0 \\
E & = & r \operatorname{cis} \phi \\
F_{h,u} & = & r \operatorname{cis}((u+1)\cdot\phi), \text{ if } u \neq 2h \\
F_{h,u} & = & (r - \frac{2}{3}\delta) \operatorname{cis}((u+1)\cdot\phi), \text{ if } u = 2h \\
G & = & r \operatorname{cis}((2W+3)\cdot\phi) \\
H_1 & = & 4r' \operatorname{cis}((3W+4)\cdot\phi) \\
H_2 & = & 8r' \operatorname{cis}((2W+3)\cdot\phi).
\end{array}
$$

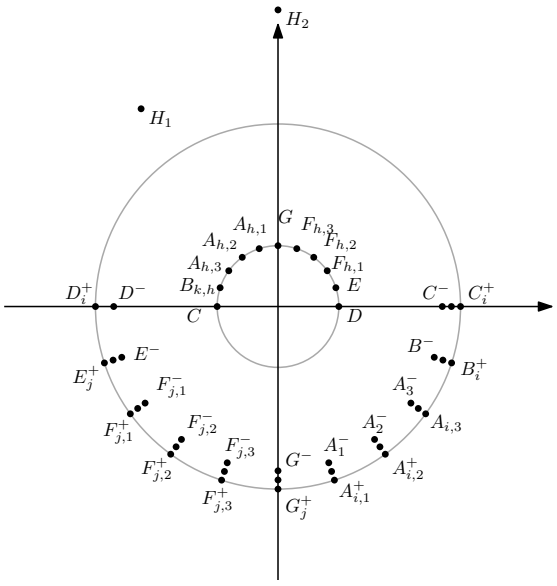


Figure 2.21: The vertices of T_1 and T_2, for $W = 1$.

Now we are ready to define the first subtrajectory T_1.

$$\begin{aligned} T_1 = \bigcirc_{1\le k\le n} \Big(\bigcirc_{1\le h\le W} \big(G \circ \bigcirc_{1\le u\le 2W+1}(A_{h,u}) \circ B_{k,h} \circ C \circ D \circ C \circ D \\ \circ E \circ \bigcirc_{1\le u\le 2W+1}(F_{h,u})\big) \\ \circ G \circ \bigcirc_{1\le u\le 2W+1}(A_{W+1,u}) \circ B_{k,W+1} \circ C \circ D \circ C \circ D \\ \circ C \circ H_1 \circ H_2\Big) \end{aligned}$$

Define the following points which will be used to define T_2.

$$\begin{array}{lcl} A_u^- & = & (r'-\delta)\operatorname{cis}(\pi+(2W+3+u)\cdot\phi) \text{ for all } 1\le u\le 2W+1 \\ B^- & = & (r'-\delta)\operatorname{cis}(\pi+(4W+5)\cdot\phi) \\ C^- & = & (r'-\delta)\operatorname{cis}(\pi+(4W+6)\cdot\phi) \\ D^- & = & (r'-\delta)\operatorname{cis}(\pi) \\ E^- & = & (r'-\delta)\operatorname{cis}(\pi+\phi) \\ F_u^- & = & (r'-\delta)\operatorname{cis}(\pi+(u+1)\cdot\phi) \text{ for all } 1\le u\le 2W+1 \\ G^- & = & (r'-\delta)\operatorname{cis}(\pi+(2W+3)\cdot\phi) \\ H^- & = & (r'-\delta)\operatorname{cis}((2W+3)\cdot\phi) \\ A_{i,u}^+ & = & r'\operatorname{cis}(\pi+(2W+3+u)\cdot\phi), \text{ if } i \text{ or } u \text{ are odd} \\ A_{i,u}^+ & = & r'\operatorname{cis}(\pi+(2W+3+u)\cdot\phi), \text{ if } i \text{ and } u \text{ are even and } X_{\frac{i}{2}}[\frac{u}{2}]=1 \\ A_{i,u}^+ & = & (r'-\frac{2}{3}\delta)\operatorname{cis}(\pi+(2W+3+u)\cdot\phi), \text{ if } i \text{ and } u \text{ are even and } X_{\frac{i}{2}}[\frac{u}{2}]=0 \\ B_i^+ & = & b_i\operatorname{cis}(\pi+(4W+5)\cdot\phi), \text{ where } b_i \text{ is defined below} \\ C_i^+ & = & c_i\operatorname{cis}((4W+6)\cdot\phi), \text{ where } c_i \text{ is defined below} \\ D_i^+ & = & r'\operatorname{cis}(\pi) \\ E_j^+ & = & e_j\operatorname{cis}(\pi+\phi) \text{ where } e_j \text{ is defined below} \\ F_{j,u}^+ & = & r'\operatorname{cis}(\pi+(u+1)\cdot\phi), \text{ if } j \text{ or } u \text{ are odd} \\ F_{j,u}^+ & = & r'\operatorname{cis}(\pi+(u+1)\cdot\phi), \text{ if } j \text{ and } u \text{ are even and } Y_{\frac{j}{2}}[\frac{u}{2}]=1 \\ F_{j,u}^+ & = & (r'-\frac{2}{3}\delta)\operatorname{cis}(\pi+(u+1)\cdot\phi), \text{ if } j \text{ and } u \text{ are even and } Y_{\frac{j}{2}}[\frac{u}{2}]=0 \\ G_j^+ & = & g_j\operatorname{cis}(\pi+(2W+3)\cdot\phi), \text{ where } g_j \text{ is defined below} \end{array}$$

Now we define b_i, c_i, g_j, and e_j.

Let $B_\varepsilon = (r-\varepsilon)\operatorname{cis}((4W+5)\cdot\phi)$. For $1\le i\le 2n+3$, define I_i^+ so that $C, I_1^+, I_2^+, \ldots, I_{2n+3}^+, B_\varepsilon$ are evenly spaced along the segment CB_ε. For $1\le i\le 2n+3$ define b_i so that $||B_iI_i|| = d$. For $1\le i\le 2n+1$ define c_i so that $||C_{i+2}I_i|| = d$. Define $c_1 = c_2 = c_3$.

For all $1\le j\le 2n+1$, define $J_j^+ = (r'-\frac{j}{2n+2}\delta)\operatorname{cis}(\pi+(2W+4)\cdot\phi)$ to be a point on $A_1^-A_{1,1}^+$. Define K_j to be the point on $GA_{1,1}$ such that $||K_jJ_j^+|| = d$. Define g_j to be the positive real so that $||G_j^+K_{2n+2-j}|| = d$.

For all $1\le j\le 2n+1$, define $L_j^+ = (r'-\frac{2n+4-j}{2n+2}\delta)\operatorname{cis}(\pi)$ to be a point on $D^-D_1^+$. Define M_j to be the point on DE such that $||M_jL_j^+|| = d$. Define e_j to be the positive real so that $||E_j^+M_{2n+2-j}|| = d$.

Now we are ready to define the second subtrajectory T_2.

$$\begin{aligned}T_2 = &\bigcirc_{1\le i\le 2n+1}\big(G^-\circ G_i^+\circ G^-\circ\bigcirc_{1\le u\le 2W+1}(A_u^-\circ A_{i,u}^+\circ A_u^-)\circ B^-\circ B_i^+\circ B^-\circ\bigcirc_{1\le u\le 3}(P_i\circ Q_i)\big)\\ &\circ\bigcirc_{1\le j\le 2n}\big(E^-\circ E_j^+\circ E^-\circ\bigcirc_{1\le u\le 2W+1}(F_u^-\circ F_{j,u}^+\circ F_u^-)\circ G^-\circ G_j^+\circ G^-\big)\\ &\circ C^-\circ C_1^+\circ C^-\circ D^-\circ D_1^+\circ D^-\circ C^-\circ C_2^+\circ C^-\circ D^-\circ D_2^+\circ D^-\circ H^-\\ &\circ E^-\circ E_{2n+1}^+\circ E^-\circ\bigcirc_{1\le u\le 2W+1}(F_u^-\circ F_{2n+1,u}^+\circ F_u^-)\circ G^-\circ G_{2n+1}^+\circ G^-\\ &\circ\bigcirc_{3\le i\le 2n+2}\big(C^-\circ C_i^+\circ C^-\circ G^-\circ D^-\circ D_i^+\circ D^-\big)\\ &\circ C^-\circ C_{2n+3}^+\circ C^-\circ G^-\circ D^-\circ D_{2n+3}^+\end{aligned}$$

where P_i and Q_i are points on B^-G^- so that, for all $1\le i\le n$, the subtrajectory from $A_{2i-1,1}^+$ to D_{2i+3}^+ has total length λ, for some constant λ. Define $T = T_1\circ D^-\circ T_2$. Define $m = 2nW+2$. Define $\ell = \lambda-\delta$. Recall that $d = ||r\operatorname{cis}\pi - r'\operatorname{cis}\phi||$. Then our constructed instance is (T, m, ℓ, d).

We show that B_i, E_i, G_j and K_j are well defined in Section 2.6.2, and show that P_i and Q_i are well defined in Section 2.6.4. Once we show that these points are indeed well defined, we would complete the Step 1 our reduction.

2.6.2 Points B_i, E_i, G_j and K_j are well defined

In order to show that points B_i, E_i, G_j and K_j are well defined, we start with a useful lemma. Given a point P and a segment, the lemma constructs a point X on the segment so that the distance from P to X is exactly d.

Lemma 35. *Suppose P, A, B are arbitrary points in the plane and let d be a given constant. Suppose further that $|PA| < d < |PB|$. Then there exists a point X on the segment AB so that $|PX| = d$.*

Proof. As X varies along the segment AB, the distance of P to X is a continuous function. This function starts at a value less than d (when $X = A$) and ends at a value greater than d (when $X = B$). By the intermediate value theorem, there is a point so that $|PX| = d$. □

Next, we apply this lemma repeatedly to show that B_i, E_i, G_j and K_j are well defined.

Lemma 36. *The points B_i, C_i, E_j and G_j are well defined.*

Proof. Let O be the origin. The points B_i and C_i can be shown to be well defined in a similar way, so we focus only on B_i. The definition of B_i is $b_i\operatorname{cis}(\pi+4W+6\cdot\phi)$ so that $||I_iB_i|| = d$. Let $B^+ = r'\operatorname{cis}(\pi+4W+6\cdot\phi)$. To show that B_i is well defined, we apply Lemma 35 on segment OB^+. We know $||I_iO|| < r < d$, and $||I_iB^+|| > ||OB^+|| > r' > d$ since $\angle I_iOB^+ > \frac{\pi}{2}$. Therefore, there exists a point B_i on OB^+ so that $||I_iB_i|| = d$.

The points E_j and G_j can be shown to be well defined in a similar way, so we focus only on E_j. For E_j, we first show that M_j is well defined, then we show E_j is well defined. To show that M_j is well defined, we apply Lemma 35 on segment DE. We know $||DL_j^+|| > ||DD^-|| = d$, and $||EL_j^+|| < ||ED^+|| = d$. Hence, there exists M_j on DE so that $||M_jL_j^+|| = d$. Let $E^+ = r'\operatorname{cis}(\pi+\phi)$. To show that E_j is well defined, we apply Lemma 35 on segment OE^+. We know $||M_jO|| < r < d$, and $||M_jE^+|| > ||OE^+|| > r' > d$ since $\angle M_jOE^+ > \frac{\pi}{2}$. Therefore, there exists a point E_j so that $||E_jM_j|| = d$. □

2.6.3 Antipodal property

Similar to our first key component in Section 2.3.2, the subtrajectories T_1 and T_2 are constructed in such a way that most points are within distance d of one another, whereas a few pairs of points are of distance greater than d from one another. The pairs of points with distance greater than d are called antipodes, which we list in our definition below.

Definition 37. *Define the following pairs of vertices of T to be antipodes: $(A^+_{i,u}, A_{h,u})$, $(B^+_i, B_{k,h})$, (C^+_i, C), (D^+, D), (E^+_j, E), $(F^+_{j,u}, F_{h,u})$, (G^+_j, G), and (v^+, H_u), where $u \in \{1, 2\}$ and v^+ is any vertex of T_2.*

Next, we prove that these antipodes have distance greater than d from one another, and are the only such pairs with this property.

Lemma 38. *Suppose v is a vertex of T_1 and v^+ is a vertex of T_2, so that the distance between v^+ and v is greater than d. Then v^+ and v are antipodes.*

Proof. Note that if $v = H_u$ for some $u \in \{1, 2\}$, then we immediately get that v^+ and v are antipodes. So for the remainder of this proof we assume $v \neq H_u$.

Let O be the origin. We will show that v^+, O and v are collinear, in that order. Suppose the contrary. Then $\angle v^+Ov \leq \pi - \phi$, since all vertices in our construction lie on rays emanating from the origin that are ϕ radians apart. Now, $||Ov|| \leq r$, $||Ov^+|| \leq r'$ and $\angle v^+Ov \leq \pi - \phi$. Therefore, $||v^+v|| \leq ||r \operatorname{cis} \pi - r' \operatorname{cis} \phi|| = d$, which is a contradiction. Hence, v^+, O, v are collinear, in that order. Moreover, $||v^+v|| > d$. The only pairs of vertices (v^+, v) satisfying these two properties are the antipodes listed in Definition 37. □

In the next lemma, we prove a useful property of antipodal pairs. In Section 2.6.5 we will use this useful property to prove that $F_d(T_2, T_1)$ consists of mostly free-space, with small regions of non-free space, and that each region of non-free space is associated with an antipodal pair.

Lemma 39. *Suppose a is a point on T_1 and b is a point on T_2, not necessarily vertices. Suppose that $|ab| > d$. Then there exists an endpoint E_a of the segment containing a, and an endpoint E_b of the segment containing b, so that (E_b, E_a) are antipodes.*

Proof. Suppose for the sake of contradiction that there does not exist an endpoint of the segment containing a and an endpoint of the segment containing b that are antipodes. Let a be on a_1a_2 and b be on b_1b_2. Since none of (a_u, b_w) are antipodes for $1 \leq u, w \leq 2$, we have by Lemma 38 that $||a_ub_w|| \leq d$. The distance of point a_1 to b_1b_2 is maximised at the endpoints b_1 or b_2. So $||a_1b|| \leq d$, $||a_2b|| \leq d$. But now, the distance of point b to a_1a_2 is maximised at the endpoints. So $||ab|| \leq d$. This contradiction means that our initial assumption cannot hold, and there is a pair of endpoints (E_b, E_a) which are antipodes. □

2.6.4 Subtrajectory length property

There are two aims of this subsection. First, we complete the Step 1 our reduction by showing that the points P_i and Q_i are indeed well defined. Second, we identify the critical points $J^+_{2i-1,j}$ and $L^+_{2i+3,j}$ for $1 \leq i, j \leq 2n + 1$, and define a reference subtrajectory from s to t so that l_s passes through $J^+_{2i-1,j}$ and l_t passes through $L^+_{2i+3,j}$. We show that there are $O(n^2)$ reference subtrajectories of this form, and that each of these reference subtrajectories have Euclidean length equal to ℓ.

We start with the first aim of this subsection, that is to show that P_i and Q_i are well defined. The main lemma for this is Lemma 41. Recall that P_i and Q_i are points on B^-G^- so that, for all $1 \leq i \leq n$, the subtrajectory from $A^+_{2i-1,1}$ to D^+_{2i+3} has total length λ, for some constant λ. We first show a useful lemma that we will use to prove the main lemma.

Lemma 40. *Suppose A and B are arbitrary points in the plane and λ is a positive real satisfying $|AB| < \lambda < 7 \cdot |AB|$. Then there exist points P and Q on AB so that the curve $A \circ \bigcirc_{1 \leq u \leq 3}(P \circ Q) \circ B$ has total length λ.*

Proof. Start with $P = Q = A$. At first, the total length of $A \circ \bigcirc_{1 \leq u \leq 3}(P \circ Q) \circ B$ is $|AB|$. Now, move P continuously from A to B. When P reaches B, the total length is $7|AB|$. By the intermediate value theorem, there is a position of P so that the total length of $A \circ \bigcirc_{1 \leq u \leq 3}(P \circ Q) \circ B$ is λ. □

Now we are ready to prove the main lemma.

Lemma 41. *The points P_i and Q_i are well defined for $1 \leq i, j \leq 2n + 1$.*

Proof. Define $P_{2n+1} = Q_{2n+1} = B^-$. Similarly, define $P_{2n} = Q_{2n} = P_{2n-1} = Q_{2n-1} = B^-$. Define λ to be the length of the subtrajectory from $A^+_{2n-1,1}$ to D^+_{2n+3}. We define P_i and Q_i inductively, starting at $i = 2n - 1$ and working down to $i = 1$, so that the subtrajectory from $A^+_{2i-1,1}$ to D^+_{2i+3} has length λ. In the base case of $i = 2n - 1$, the subtrajectory has length λ by definition.

In the inductive case, assume that the subtrajectory from $A^+_{2i+1,1}$ to D^+_{2i+5} has length λ. We would like to define P_i and Q_i so that the subtrajectory from $A^+_{2i-1,1}$ to D^+_{2i+3} also has length λ. This is equivalent to the subtrajectory from $A^+_{2i-1,1}$ to $A^+_{2i+1,1}$ and the subtrajectory from D^+_{2i+3} to D^+_{2i+5} having the same length. We will approximate the lengths of these subtrajectories, and show that P_i and Q_i can be defined using Lemma 40 to make the subtrajectories the same length.

First, we approximate the length of the subtrajectory from $A^+_{2i-1,1}$ to $A^+_{2i+1,1}$. We ignore P_i and Q_i initially, and add its contribution later. The length is dominated by the distances between the vertices $A^+_{2i-1,u}$ for $1 \leq u \leq 2W - 1$, B^+_{2i-1}, G^+_{2i-1}, $A^+_{i,u}$ for $1 \leq u \leq 2W - 1$, B^+_{2i}, G^+_{2i} and finally $A^+_{2i+1,u}$. Formally, we can set $r' >> r$ so that δ approaches zero, so B^- is arbitrarily close to B^+. With this simplification in mind, the subtrajectory is approximately a closed loop that visits B^+ twice and G^+ twice. So a lower bound for the length of the subtrajectory is $4||B^+G^+|| \approx 4\sqrt{2}r'$. An upper bound for the length of the subtrajectory is to replace the segments of the subtrajectory with an arc of the circle centered at the origin with radius r'. This upper bound is a closed loop on the circumference that visits B^+ twice and G^+ twice. So an upper bound for the length of the subtrajectory is four times the arc from B^+ to G^+, which is approximately $2\pi r'$. Let the length of $A^+_{2i-1,1}$ to $A^+_{2i+1,1}$, ignoring the contributions of P_i and Q_i, be λ_i, so that $4\sqrt{2}r' \leq \lambda_i \leq 2\pi r'$.

Next, we approximate the length of the subtrajectory from D^+_{2i+3} to D^+_{2i+5}. The length is dominated by the distances between the vertices D^+, G^-, C^+, G^-, D^-, G^-, C^+, G^- and back to D^+. The length of this closed loop is approximately $8||D^+G^-|| = 8\sqrt{2}r'$.

Now we use the same idea as Lemma 40. Start with $P_i = Q_i = B^-$. Then the length of the subtrajectory from $A^+_{2i-1,1}$ to $A^+_{2i+1,1}$ is simply λ_i. Now, move P_i continuously from B^- to G^-. When P_i reaches G^-, then the length of the subtrajectory from $A^+_{2i-1,1}$ to $A^+_{2i+1,1}$ is $\lambda_i + 6|B^-G^-| \approx \lambda_i + 6\sqrt{2}r'$. Therefore, as P_i moves continuously, the initial length of $A^+_{2i-1,1}$ to $A^+_{2i+1,1}$ is at most $2\pi r'$, and its final length is at least $10\sqrt{2}r'$. By the intermediate

value theorem, there exists a position of P_i so that the lengths of the subtrajectories from $A^+_{2i-1,1}$ to $A^+_{2i+1,1}$ and from D^+_{2i+3} to D^+_{2i+5} are the same. □

Now that all points in the construction are well defined, we have completed the first aim of this subsection. Next, we focus on the second aim of this subsection, which is to identify critical points $J^+_{2i-1,j}$ and $L^+_{2i+3,j}$.

To help us define the critical points $J^+_{2i-1,j}$ and $L^+_{2i+3,j}$, we add subscripts to the vertices of T, without modifying the trajectory T. For example, for the vertex G, we let $G_{k,h}$ denote the $(kW+k+h)^{th}$ time G appears in T. This subscripted version of the trajectory T is given as follows:

$$
\begin{aligned}
T = \bigcirc_{1\le k\le n} \Big(\bigcirc_{1\le h\le W} \big(G_{k,h} \circ \bigcirc_{1\le u\le 2W+1}(A_{k,h,u}) \circ B_{k,h} \circ C_{k,h,1} \circ D_{k,h,1} \circ C_{k,h,2} \circ D_{k,h,2} \\
\circ E_{k,h} \circ \bigcirc_{1\le u\le 2W+1}(F_{k,h,u})\big) \\
\circ G_{k,W+1} \bigcirc_{1\le u\le 2W+1} (A_{k,W+1,u}) \circ B_{k,W+1} \circ C_{k,W+1,1} \circ D_{k,W+1,1} \circ C_{k,W+1,2} \\
\circ D_{k,W+1,2} \circ C_{k,W+1,3} \circ H_{k,1} \circ H_{k,2}\Big) \circ D^-_0 \\
\circ \bigcirc_{1\le i\le 2n+1}\big(G^-_{i,1} \circ G^+_{i,1} \circ G^-_{i,1} \circ \bigcirc_{1\le u\le 2W+1}(A^-_{i,u} \circ A^+_{i,u} \circ A^-_{i,u}) \circ B^-_i \circ B^+_i \circ B^-_i \\
\circ \bigcirc_{1\le u\le 3}(P_{i,u} \circ Q_{i,u})\big) \\
\circ \bigcirc_{1\le j\le 2n}\big(E^-_j \circ E^+_j \circ E^-_j \circ \bigcirc_{1\le u\le 2W+1}(F^-_{j,u} \circ F^+_{j,u} \circ F^-_{j,u}) \circ G^-_{j,2} \circ G^+_{j,2} \circ G^-_{j,2}\big) \\
\circ C^-_1 \circ C^+_1 \circ C^-_1 \circ D^-_1 \circ D^+_1 \circ D^-_1 \circ C^-_2 \circ C^+_2 \circ C^-_2 \circ D^-_2 \circ D^+_2 \circ D^-_2 \circ H^-_1 \\
\circ E^-_{2n+1} \circ E^+_{2n+1} \circ E^-_{2n+1} \circ \bigcirc_{1\le u\le 2W+1}(F^-_{2n+1,u} \circ F^+_{2n+1,u} \circ F^-_{2n+1,u}) \circ G^-_{2n+1,2} \circ G^+_{2n+1,2} \\
\circ G^-_{2n+1,2} \circ \bigcirc_{3\le i\le 2n+3}\big(C^-_i \circ C^+_i \circ C^-_i \circ G^-_{i,3} \circ D^-_i \circ D^+_i \circ D^-_i \circ G^-_{i,4}\big) \\
\circ C^-_{2n+3} \circ C^+_{2n+3} \circ C^-_{2n+3} \circ G^-_{2n+3,3} \circ D^-_{2n+3} \circ D^+_{2n+3}
\end{aligned}
$$

Now we can define the critical points $J^+_{2i-1,j}$ and $L^+_{2i+3,j}$ on the re-indexed trajectory. Recall from Section 2.6.1 that $J^+_j = (r' - \frac{j}{2n+2}\delta)\operatorname{cis}(\pi + (2W+4)\cdot\phi)$ is a point on $A^-_1 A^+_{1,1}$. Similarly, $L^+_j = (r' - \frac{2n+4-j}{2n+2}\delta)\operatorname{cis}(\pi)$ is a point on $D^- D^+_1$. Define $J^+_{2i-1,j}$ to coincide with J^+_j in the complex plane, so that $A^+_{2i-1,1}$, $J^+_{2i-1,j}$ and $A^-_{2i-1,1}$ are in that order along the trajectory T. Similarly, define $L^+_{2i+3,j}$ to coincide with L^+_j in the complex plane, so that D^-_{2i+3}, $L^+_{2i+3,j}$ and D^+_{2i+3} are in that order along the trajectory T.

Clearly, there are n^2 pairs of critical points $J^+_{2i-1,j}$ and $L^+_{2i+3,j}$. By considering the reference subtrajectory from s to t so that l_s passes through $J^+_{2i-1,j}$, and l_t passes through $L^+_{2i+3,j}$, we obtain the n^2 reference subtrajectories that are mentioned our third key component in Section 2.3.2. It suffices to show that if we pick l_s and l_t in this way, then all n^2 reference subtrajectories have Euclidean length equal to ℓ.

Lemma 42. *For all $1 \le i, j \le n$, the length of the subtrajectory from $J^+_{2i-1,2j-1}$ to $L^+_{2i+3,2j+1}$ is ℓ.*

Proof. We already have that the subtrajectory from $A^+_{2i-1,1}$ to D^+_{2i+3} has length λ, for all $1 \le i \le n$. The distance from $A^+_{2i-1,1}$ to J^+_j is $\frac{2j-1}{2n+2}\delta$. The distance from L^+_j to D^+_{2n+3} is $\frac{2n+4-2j-1}{2n+2}\delta$. Therefore, the length of the subtrajectory from J^+_j to L^+_j is $\lambda - \frac{2j-1}{2n+2}\delta - \frac{2n+4-2j-1}{2n+2}\delta = \lambda - \delta = \ell$ as required. □

This completes the second aim of this subsection. We also show one last property of the free space diagram $F_d(T,T)$. In particular, we show that l_s and l_t cannot be between the vertical lines through the x-coordinates corresponding to $H_{k,1}$ and $H_{k+1,2}$. It involves modifying the construction of the base case in Lemma 41.

Lemma 43. *For all $1 \leq k \leq n$, the length of the subtrajectory from $H_{k,1}$ to $H_{k+1,2}$ is less than ℓ.*

Proof. Consider the subtrajectory from $H_{k,1}H_{k+1,2}$. All points on this subtrajectory, other than $H_{k,1}$, $H_{k,2}$, $H_{k+1,1}$ and $H_{k+1,2}$ lie within a circle of radius r centered at the origin. We can set r' to be arbitrarily large relative to r, so that the length of the subtrajectory is dominated by the distances between the origin, $H_{k,1}$, $H_{k,2}$, $H_{k+1,1}$ and $H_{k+1,2}$. Therefore, the length of the subtrajectory is at most $40r'$, ignoring terms involving r. It suffices to show that $\ell > 40r'$.

Recall that the subtrajectory from $A^+_{2n-1,1}$ to D^+_{2n+3} is defined to have length $\lambda = \ell + \delta$. Recall that for all $1 \leq i \leq 2n+3$, P_i and Q_i were defined so that the subtrajectory from $A^+_{2i-1,1}$ to D^+_{2i+3} also has length λ. We will modify the subtrajectory $A^+_{2n-1,1}$ to D^+_{2n+3} so that $\lambda > 41r'$, and therefore, $\ell > 40r'$.

To do this, instead of initialising our induction in Lemma 41 with $P_{2n+1} = Q_{2n+1} = B^-$ we place P_{2n+1} and Q_{2n+1} at B^- and $G-$ respectively. Moreover, instead of placing three copies of each of P_{2n+1} and Q_{2n+1}, we place 41 copies. Each subtrajectory $P_{2n+1} \circ Q_{2n+1} \circ P_{2n+1}$ has length at least r', so $\lambda > 41r'$, as required. □

2.6.5 Free space diagram

In this section, we describe the free space diagram $F_d(T,T)$, in other words, the free space diagram between the trajectory T and itself, with distance parameter d. We use the antipodal property in Section 2.6.3 to show that our free space diagram is mostly free space, with regions of non-free space associated with antipodal pairs.

We begin with the following definition. For any pair of points x and y on the trajectory T, not necessarily vertices of T, define $f(x,y)$ to be the point in $F_d(T,T)$ with x-coordinate associated with point x and y-coordinate associated with point y. We introduce the following notation to describe the regions of non-free space in the free space diagram.

Definition 44. *Suppose $f(x,y)$ is not in the free space of $F_d(T,T)$. Define $\Diamond(x,y)$ to be the region of non-free space that contains $f(x,y)$.*

Next, we show the non-free space $\Diamond(x,y)$ is always associated with an antipodal pair.

Lemma 45. *All regions of non-free space in $F_d(T_2,T_1)$ are $\Diamond(v^+,v)$ where the pair (v^+,v) are antipodes.*

Proof. Suppose $f(x,y)$ is not in free space. Then by Lemma 39, there exists an endpoint of the segment containing x and an endpoint of the segment containing y that are antipodes. Let (v^+,v) be this pair of antipodes. Let x be on the segment v^-v^+ and let y be on the segment uv. We can verify that for all (v^+,v) that are antipodes in construction in Section 2.6.1, by moving from v^- to v^+ we move further away from (any points on) segment uv, and similarly, by moving from u to v we move further away from (any point on) segment v^-v^+.

Using this fact, we can prove that desired lemma. Since $f(x,y)$ is in non-free space, we have $||xy|| > d$. By moving from x to v^+, we move further away from y, which is on

uv. Hence, $||v^+y|| > d$, and all points between $f(x, y)$ and $f(v^+, y)$ are in non-free space. Finally, by moving from y to v, we move further away from v^+. Hence, all points between $f(v^+, y)$ and $f(v^+, v)$ are non-free space. Therefore, we have constructed a path of non-free space connecting $f(x, y)$ and $f(v^+, v)$, so $f(x, y)$ must be contained in $\Diamond(v^+, v)$. □

Now we show that the non-free space $\Diamond(v^+, v)$ is in fact a diamond. In particular, it is diamond shaped, with concave (inwards-curved) sides. See Figure 2.22.

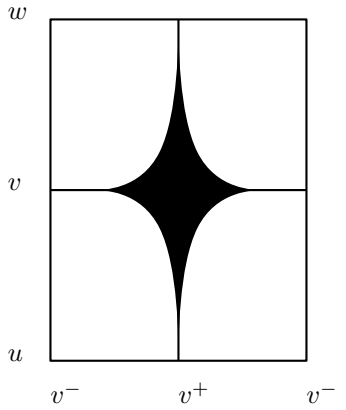


Figure 2.22: $\Diamond(v^+, v)$ is diamond shaped, with concave (inwards-curved) sides.

Lemma 46. *Suppose (v^+, v) are antipodes, $v \neq H_{u,1}$ and $v \neq H_{u,2}$ for any $1 \leq u \leq n$. Then $\Diamond(v^+, v)$ is diamond shaped, with concave (inwards-curved) sides. Moreover, the x-coordinates spanned by $\Diamond(v^+, v)$ corresponds to a subset of the neighbouring segments of v^+; the y-coordinates spanned by $\Diamond(v^+, v)$ corresponds to a subset of the neighbouring segments of v.*

Proof. Lemma 39 immediately implies that the x and y-coordinates of $\Diamond(v^+, v)$ is spanned by the neighbouring segments of v^+ and v respectively. It suffices to show that the shape of this non-free space is diamond shaped, with concave (inwards-curved) sides.

Draw a vertical line $\Diamond(v^+, v)$ through the x-coordinate v^+, and a horizontal line through the y-coordinate v. This divides $\Diamond(v^+, v)$ into four quadrants. We will show that each quadrant consists of a continuous, xy-monotone side. Then we will show that the sides are concave.

Let the neighbouring segments of v^+ in T_2 be v^-v^+ and v^+v^-. Let the neighbouring segments of v in T_1 be uv and vw. We can verify that for all (v^+, v) that are antipodes in our construction, that the point v^- is strictly closer to u, v, w than v^+. We can also verify for all antipodes that u and w are closer to v^+ than v.

Let us focus on the top right quadrant. This is the free space with x-coordinates associated with the segment v^+v^-, and y-coordinates associated with the segment vw. Along the edge v^+v^-, the distance to any fixed point on vw is strictly increasing. Similarly, along the edge vw, the distance to any fixed point on v^+v^- is strictly increasing. Therefore, the boundary of $\Diamond(v^+, v)$, which is the set of all $x \in v^+v^-$ and $y \in vw$ so that $||xy|| = d$, is a continuous curve starting at the same y-coordinate as v^+, and moving down and to the right. This gives one of the four curved sides of the diamond. Moreover, we know that in the free space diagram, the free space must be the intersection of an ellipse with the cell.

Hence, this xy-monotone side is concave, inward-facing, and the boundary of an ellipse. Repeating this for all four quadrants gives the four continuous, xy-monotone and concave (inwards-curved) sides of $\Diamond(v^+, v)$. □

Finally, we describe the free space and non-free space on the vertical line through the x-coordinate associated with the point $H_{k,2}$ for some $1 \leq k \leq n$. We the vertical line alternates between free and non-free space, where the free space corresponds with $H_{u,2}$ for some u, as shown in Figure 2.23.

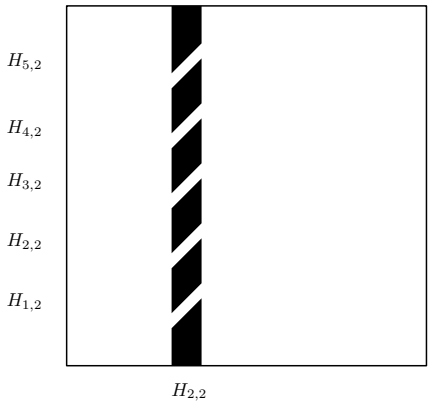


Figure 2.23: An example of the vertical line from $H_{k,2}$ passing through alternating regions of free and non-free space, where $k = 2$ and $n = 5$.

Lemma 47. *Consider the vertical line in the free space diagram $F_d(T, T)$ with x-coordinate corresponding to $H_{k,2}$. This vertical line consists of alternating regions of free space and non-free space, with n regions of contiguous free space and $n + 1$ regions of contiguous non-free space. Moreover, each of the n regions of contiguous free space contains the point $f(H_{k,2}, H_{u,2})$ for some $1 \leq u \leq n$.*

Proof. Consider the vertices of T. Divide the vertices into two sets, $H_2 = \{H_{u,2}$ for some $1 \leq u \leq n\}$, and the other vertices $T \setminus H_2$. For all $v \in T \setminus H_2$, the vertex v is within the ball of radius $4r'$ centered at the original, so the distance from v to $H_{k,2}$ is greater than d. However, for all $v \in H_2$, the distance from v to $H_{k,2}$ is zero.

Let x be an arbitrary point on T. If x is on a segment where both endpoints are in $T \setminus H_2$, then $f(H_{k,2}, x)$ is non-free space. If one of the endpoints are in $T \setminus H_2$, then x will only be free space if it is within distance d to $H_{k,2}$. These regions of close proximity only occur when one of the endpoints is $H_{u,2}$. So there are exactly n segments of free space, and $n + 1$ segments of non-free space on the vertical line with x-coordinate $H_{k,2}$. □

This completes Step 2 of our reduction, that is, to prove the various useful properties of the free space diagram $F_d(T, T)$. For the remainder of this chapter, we will focus on Step 3, to use the properties of the free space diagram $F_d(T, T)$ to prove that $(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$ is a YES instance if and only if (T, m, ℓ, d) is a YES instance.

2.6.6 Paths in free space

In the following section (Section 2.6.7), we will show that if $(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$ is a YES instance for 3OV, then (T, m, ℓ, d) is a YES instance for SC. This involves constructing a pair of vertical lines l_s and l_t, and $m-1$ monotone paths from l_s to l_t, with the property that the $m-1$ monotone paths intersect in their y-coordinates in at most one point, and the each of the $m-1$ monotone paths intersect with the y-interval from s to t in at most one point.

In this section, we compile a list of monotone paths, and prove their correctness. Some of these monotone paths are conditioned on one of the booleans $X_i[h]$, $Y_j[h]$ or $Z_k[h]$ being equal to zero. The list of monotone paths from this section will be used extensively in the subsequent section.

To aid with our description of monotone paths in both this section and the subsequence section, we use the following notation. Suppose (v^+, v) are antipodes, $v \neq H_{u,1}$ and $v \neq H_{u,2}$ for any $1 \leq u \leq n$. Define the top, bottom, left and right corners of $\Diamond(v^+, v)$ to be $\Diamond_T(v^+, v)$, $\Diamond_B(v^+, v)$, $\Diamond_L(v^+, v)$ and $\Diamond_R(v^+, v)$ respectively.

We say $f(x, y) \to f(w, z)$ if there is a monotone path from $f(x, y)$ to $f(w, z)$.

Lemma 48. *For $1 \leq i, j, k \leq n$ and $1 \leq h \leq W$:*

(a) $\Diamond_R(A^+_{2i-1,1}, A_{k,h,1}) \to \Diamond_T(B^+_{2i}, B_{k,h})$,

(b) $\Diamond_R(A^+_{2i-1,1}, A_{k,h,1}) \to \Diamond_T(B^+_{2i+1}, B_{k,h})$ *if* $X_i[h] = 0$,

(c) $\Diamond_T(B^+_i, B_{k,h}) \to \Diamond_B(C^+_{i+1}, C_{k,h,1})$,

(d) $\Diamond_T(B^+_{2i}, B_{k,h}) \to \Diamond_B(C^+_{2i+2}, C_{k,h,1})$ *if* $Z_k[h] = 0$,

(e) $\Diamond_B(C^+_{i+2}, C_{k,h,1}) \to \Diamond_L(D^+_{i+3}, D_{k,h,1})$,

(f) $\Diamond_R(A^+_{2i-1,1}, A_{k,h,1}) \to \Diamond_L(D^+_{2i+3}, D_{k,h,2})$.

(g) $\Diamond_R(A^+_{2i-1,1}, A_{k,h,1}) \to \Diamond_L(D^+_{2i+3}, D_{k,h,1})$ *if* $X_i[h] \cdot Z_k[h] = 0$.

(h) $\Diamond_L(A^+_{2i-1,2}, A_{k,h,2}) \to \Diamond_T(D^+_{2i+2}, D_{k,h,2})$.

(i) $\Diamond_B(E^+_{2j-1}, E_{k,h}) \to \Diamond_T(G^+_{2j,2}, G_{k,h})$.

(j) $\Diamond_B(E^+_{2j-1}, E_{k,h}) \to \Diamond_T(G^+_{2j+1,2}, G_{k,h})$ *if* $Y_j[h] = 0$.

(k) $\Diamond_L(E^+_1, E_{k,h}) \to \Diamond_T(G^+_{1,2}, G_{k,h})$.

(l) $f(E^+_1, D_{k,h,2}) \to \Diamond_T(G^+_{2n+1,2}, G_{k,h})$.

Proof.

(a) $\Diamond_R(A^+_{2i-1,1}, A_{k,h,1}) \to \Diamond_L(A^+_{2i,1}, A_{k,h,1}) \to \Diamond_L(A^+_{2i,2}, A_{k,h,2}) \to \ldots \to \Diamond_L(A^+_{2i,2W+1}, A_{k,h,2W+1}) \to \Diamond_T(B^+_{2i}, B_{k,h})$

(b) Since $X_i[h] = 0$, we have that $\Diamond(A^+_{2i,2h}, A_{k,h,2h})$ is non-existent by construction. Therefore, $\Diamond_T(A^+_{2i,2h-1}, A_{k,h,2h-1}) \to \Diamond_B(A^+_{2i,2h+1}, A_{k,h,2h+1})$. Hence, we have $\Diamond_R(A^+_{2i-1,1}, A_{k,h,1}) \to \Diamond_L(A^+_{2i,1}, A_{k,h,1}) \to \Diamond_L(A^+_{2i,2}, A_{k,h,2}) \to \ldots \to \Diamond_L(A^+_{2i,2h-1}, A_{k,h,2h-1}) \to \Diamond_T(A^+_{2i,2h-1}, A_{k,h,2h-1}) \to \Diamond_B(A^+_{2i,2h+1}, A_{k,h,2h+1}) \to \Diamond_L(A^+_{2i+1,2h}, A_{k,h,2h}) \to \Diamond_L(A^+_{2i+1,2h+1}, A_{k,h,2h+1}) \to \ldots \to \Diamond_L(A^+_{2i+1,2W+1}, A_{k,h,2W+1}) \to \Diamond_L(B^+_{2i+1}, B_{k,h}) \to \Diamond_T(B^+_{2i+1}, B_{k,h})$

(c) Define $I^+_{i-0.5}$ to be the midpoint of $I^+_i I^+_{i-1}$. From the definition of I_i, we have $||B^+_i I^+_{i-0.5}|| < d$ and $||C^+_{i+1} I^+_{i-0.5}|| < d$. The point $I^+_{i-0.5}$ lies on CB, however, since ε can be chosen to be arbitrarily small, we can place $I^+_{i-0.5}$ on $B_{k,h}C_{k,h,1}$ while maintaining $||B^+_i I^+_{i-0.5}|| < d$ and $||C^+_{i+1} I^+_{i-0.5}|| < d$. Let $I^+_{i-0.5,k,h}$ be the copy of $I^+_{i-0.5}$ on the segment $B_{k,h}C_{k,h,1}$. Hence, $\Diamond_T(B^+_i, B_{k,h}) \to f(B^+_i, I^+_{i-0.5,k,h}) \to f(C^+_{i+1}, I^+_{i-0.5,k,h}) \to \Diamond_B(C^+_{i+1}, C_{k,h,1})$.

(d) Since $Z_k[h] = 0$, we have $B_{k,h} = B_{k,W+1}$. Let $I_{i,k,h}$ be the copy of the point I_i on the segment $B_{k,h}C_{k,h,1}$. Then by our definition of I_i, we have $||B_i I_i|| = d$ and $||C_{i+2} I_i|| = d$. Hence, $\Diamond_T(B^+_{2i}, B_{k,h}) = f(I_{2i}, B_{k,h}) \to f(I_{2i}, C_{k,h,1}) = \Diamond_B(C^+_{2i+2}, C_{k,h,1})$.

(e) $\Diamond_B(C^+_{i+2}, C_{k,h,1}) \to \Diamond_R(C^+_{i+2}, C_{k,h,1}) \to \Diamond_L(C^+_{i+3}, C_{k,h,1}) \to \Diamond_L(D^+_{i+3}, D_{k,h,1})$.

(f) By *(a)*, $\Diamond_R(A^+_{2i-1,1}, A_{k,h,1}) \to \Diamond_T(B^+_{2i}, B_{k,h})$. By *(c)*, $\Diamond_T(B^+_{2i}, B_{k,h}) \to \Diamond_B(C^+_{2i+1}, C_{k,h,1})$. By *(e)*, $\Diamond_B(C^+_{2i+1}, C_{k,h,1}) \to \Diamond_L(D^+_{2i+2}, D_{k,h,1})$. Putting this together, $\Diamond_R(A^+_{2i-1,1}, A_{k,h,1}) \to$

$\Diamond_T(B^+_{2i}, B_{k,h}) \to \Diamond_B(C^+_{2i+1}, C_{k,h,1}) \to \Diamond_L(D^+_{2i+2}, D_{k,h,1}) \to \Diamond_L(D^+_{2i+3}, D_{k,h,2})$.

(g) We modify the proof of *(f)* in the case where $X_i[h] \cdot Z_k[h] = 0$. If $X_i[h] = 0$, we replace *(a)* with *(b)* to yield $\Diamond_R(A^+_{2i-1,1}, A_{k,h,1}) \to \Diamond_T(B^+_{2i+1}, B_{k,h}) \to \Diamond_B(C^+_{2i+2}, C_{k,h,1}) \to \Diamond_L(D^+_{2i+3}, D_{k,h,1})$. If $Z_k[h] = 0$, we replace *(c)* with *(d)* to yield $\Diamond_R(A^+_{2i-1,1}, A_{k,h,1}) \to \Diamond_T(B^+_{2i}, B_{k,h}) \to \Diamond_B(C^+_{2i+2}, C_{k,h,1}) \to \Diamond_L(D^+_{2i+3}, D_{k,h,1})$.

(h) $\Diamond_L(A^+_{2i-1,2}, A_{k,h,2}) \to \Diamond_T(B^+_{2i-1}, B_{k,h}) \to \Diamond_B(C^+_{2i}, C_{k,h,1}) \to \Diamond_L(D^+_{2i+1}, D_{k,h,1}) \to \Diamond_L(D^+_{2i+2}, D_{k,h,2})$.

(i) $\Diamond_B(E^+_{2j-1}, E_{k,h}) \to \Diamond_L(E^+_{2j-1}, E_{k,h}) \to \Diamond_L(E^+_{2j}, E_{k,h}) \to \Diamond_L(F^+_{2j,1}, F_{k,h,1}) \to \Diamond_L(F^+_{2j,2}, F_{k,h,2}) \to \ldots \to \Diamond_L(F^+_{2j,2W+1}, F_{k,h,2W+1}) \to \Diamond_T(G^+_{2j,2}, G_{k,h})$

(j) Since $Y_k[h] = 0$, we have that $\Diamond(F^+_{2j,2h}, F_{k,h,2h})$ is non-existent. Therefore, $\Diamond_T(F^+_{2j,2h-1}, F_{k,h,2h-1}) \to \Diamond_B(F^+_{2j,2h-1}, F_{k,h,2h-1})$. Hence, $\Diamond_B(E^+_{2j-1}, E_{k,h}) \to \Diamond_L(E^+_{2j-1}, E_{k,h}) \to \Diamond_L(E^+_{2j}, E_{k,h}) \to \Diamond_L(F^+_{2j,1}, F_{k,h,1}) \to \Diamond_L(F^+_{2j,2}, F_{k,h,2}) \to \ldots \to \Diamond_L(F^+_{2j,2h-1}, F_{k,h,2h-1}) \to \Diamond_T(F^+_{2j,2h-1}, F_{k,h,2h-1}) \to \Diamond_B(F^+_{2j,2h-1}, F_{k,h,2h-1}) \to \Diamond_L(F^+_{2j+1,2h}, F_{k,h,2h}) \to \Diamond_L(F^+_{2j+1,2h+1}, F_{k,h,2h+1}) \to \ldots \to \Diamond_L(F^+_{2j+1,2W+1}, F_{k,h,2W+1}) \to \Diamond_T(G^+_{2j+1,2}, G_{k,h})$

(k) $\Diamond_L(E^+_1, E_{k,h}) \to \Diamond_L(F^+_{1,1}, F_{k,h,1}) \to \Diamond_L(F^+_{1,2}, F_{k,h,2}) \to \ldots \to \Diamond_L(F^+_{1,2W+1}, F_{k,h,2W+1}) \to \Diamond_T(G^+_{1,2}, G_{k,h})$

(l) $f(E^+_1, D_{k,h,2}) \to \Diamond_L(D^+_1, D_{k,h,2}) \to \Diamond_L(E^+_{2n+1}, E_{k,h}) \to \Diamond_L(F^+_{2n+1,1}, F_{k,h,1}) \to \Diamond_L(F^+_{2n+1,2}, F_{k,h,2}) \to \ldots \to \Diamond_L(F^+_{2n+1,2W+1}, F_{k,h,2W+1}) \to \Diamond_T(G^+_{2n+1,2}, G_{k,h})$ □

2.6.7 YES instances

In this section, we show that if our input $(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$ to 3OV is a YES instance, then our constructed instance (T, m, ℓ, d) for SC is a YES instance.

Since $(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$ is a YES instance, there exists a triple of integers $1 \le \bar{i}, \bar{j}, \bar{k} \le n$, so that for all $1 \le h \le W$, we have $X_{\bar{i}}[h] \cdot Y_{\bar{j}}[h] \cdot Z_{\bar{k}}[h] = 0$. To show that (T, m, ℓ, d) is a YES instance, it suffices to construct a pair of vertical lines l_s and l_t in $F_d(T, T)$ so that, there

are $m-1$ distinct monotone paths starting at l_s and ending at l_t, where the y-coordinates of any two monotone paths overlap in at most one point, and where the subtrajectory from s to t has length at least ℓ. Moreover, each of the monotone paths intersect the y-interval from s to t in at most one point.

Choose the starting point $s = J^+_{2\bar{i}-1,2\bar{j}-1}$ and ending point $t = L^+_{2\bar{i}+3,2\bar{j}+1}$. By Lemma 42 the subtrajectory from s to t has length exactly ℓ. Next, we construct a set of $m-1 = 2nW+1$ monotone paths from l_s to l_t.

Define $K_{j,k,h}$ to be the point on $G_{k,h}A_{k,h,1}$ so that $||K_{j,k,h}J^+_{i,j}|| = d$ and $||K_{j,k,h}G^+_{2n+2-j,2}|| = d$. Define $M_{j,k,h}$ to be the point on $D_{k,h,2}E_{k,h}$ so that $||M_{j,k,h}L^+_{i,j}|| = d$ and $||M_{j,k,h}E^+_{2n+2-j}|| = d$. Define $M_{j,k,W+1}$ to be the point on $D_{k,W+1,2}C_{k,W+1,3}$ so that $||M_{j,k,W+1}L^+_{i,j}|| = d$.

Our set of constructed paths is as follows:

- For $1 \le k \le n$, $k \ne \bar{k}$, $1 \le h \le W$, construct $f(J^+_{2\bar{i}-1,2\bar{j}-1}, A_{k,h,2}) \to f(L^+_{2\bar{i}+3,2\bar{j}+1}, E_{k,h})$.
- For $1 \le k \le n$, $k \ne \bar{k}$, $1 \le h \le W$ construct $f(J^+_{2\bar{i}-1,2\bar{j}-1}, E_{k,h}) \to f(L^+_{2\bar{i}+3,2\bar{j}+1}, A_{k,h+1,1})$.
- For $k = \bar{k}$, $1 \le h \le W$, $Y_{\bar{j}}[h] = 0$, construct $f(J^+_{2\bar{i}-1,2\bar{j}-1}, K_{2\bar{j}-1,\bar{k},h}) \to f(L^+_{2\bar{i}+3,2\bar{j}+1}, M_{2\bar{j}+1,\bar{k},h})$.
- For $k = \bar{k}$, $1 \le h \le W$, $Y_{\bar{j}}[h] = 0$, construct $f(J^+_{2\bar{i}-1,2\bar{j}-1}, M_{2\bar{j}+1,\bar{k},h}) \to f(L^+_{2\bar{i}+3,2\bar{j}+1}, K_{2\bar{j}-1,\bar{k},h+1})$.
- For $k = \bar{k}$, $1 \le h \le W$, $Y_{\bar{j}}[h] \ne 0$, construct $f(J^+_{2\bar{i}-1,2\bar{j}-1}, K_{2\bar{j}-1,\bar{k},h}) \to f(L^+_{2\bar{i}+3,2\bar{j}+1}, C_{k,h,2})$.
- For $k = \bar{k}$, $1 \le h \le W$, $Y_{\bar{j}}[h] \ne 0$, construct $f(J^+_{2\bar{i}-1,2\bar{j}-1}, D_{k,h,2}) \to f(L^+_{2\bar{i}+3,2\bar{j}+1}, K_{2\bar{j}-1,\bar{k},h+1})$.
- Construct $f(J^+_{2\bar{i}-1,2\bar{j}-1}, K_{2\bar{j}-1,\bar{k},W+1}) \to f(L^+_{2\bar{i}+3,2\bar{j}+1}, M_{2\bar{j}+1,\bar{k},W+1})$

We prove the correctness of our constructed paths. First, we note that $f(J^+_{2\bar{i}-1,2\bar{j}-1}, K_{2\bar{j}-1,\bar{k},h})$ and $f(L^+_{2\bar{i}+3,2\bar{j}+1}, M_{2\bar{j}+1,\bar{k},h})$ are free points since $||K_{2\bar{j}-1,k,h}J^+_{i,2\bar{j}-1}|| = d$ and $||M_{2\bar{j}+1,k,h}L^+_{i,2\bar{j}+1}|| = d$, respectively. In total, we have $2nW+1$ paths. Next, we prove that these $2nW+1$ paths are indeed valid.

Lemma 49. $f(J^+_{2\bar{i}-1,2\bar{j}-1}, A_{k,h,2}) \to f(L^+_{2\bar{i}+3,2\bar{j}+1}, E_{k,h})$

Proof. $f(J^+_{2\bar{i}-1,2\bar{j}-1}, A_{k,h,2}) \to \Diamond_L(A^+_{2\bar{i}-1,2}, A_{k,h,2}) \to \Diamond_T(D^+_{2\bar{i}+2}, D_{k,h,2}) \to f(L^+_{2\bar{i}+3,2\bar{j}+1}, E_{k,h})$, where the second $\to$ in the chain is given by Lemma 48*(h)*. □

Lemma 50. $f(J^+_{2\bar{i}-1,2\bar{j}-1}, E_{k,h}) \to f(L^+_{2\bar{i}+3,2\bar{j}+1}, A_{k,h+1,1})$

Proof. $f(J^+_{2\bar{i}-1,2\bar{j}-1}, E_{k,h}) \to \Diamond_L(E^+_1, E_{k,h}) \to \Diamond_T(G^+_{1,2}, G_{k,h}) \to f(L^+_{2\bar{i}+3,2\bar{j}+1}, A_{k,h+1,1})$, where the second $\to$ in the chain is given by Lemma 48*(k)*. □

Lemma 51. $f(J^+_{2\bar{i}-1,2\bar{j}-1}, K_{2\bar{j}-1,\bar{k},h}) \to f(L^+_{2\bar{i}+3,2\bar{j}+1}, M_{2\bar{j}+1,\bar{k},h})$

Proof. $f(J^+_{2\bar{i}-1,2\bar{j}-1}, K_{2\bar{j}-1,\bar{k},h}) \to \Diamond_R(A^+_{2\bar{i}-1,1}, A_{k,h,1}) \to \Diamond_L(D^+_{2\bar{i}+3}, D_{k,h,2}) \to f(L^+_{2\bar{i}+3,2\bar{j}+1}, M_{2\bar{j}+1,\bar{k},h})$, where the second $\to$ in the chain is given by Lemma 48*(f)*. □

Lemma 52. $f(J^+_{2\bar{i}-1,2\bar{j}-1}, M_{2\bar{j}+1,\bar{k},h}) \to f(L^+_{2\bar{i}+3,2\bar{j}+1}, K_{2\bar{j}-1,\bar{k},h+1})$ *if* $Y_{\bar{j}}[h] = 0$.

Proof. $f(J^+_{2\bar{i}-1,2\bar{j}-1}, M_{2\bar{j}+1,\bar{k},h}) \to \Diamond_B(E^+_{2n+1-2\bar{j}}, E_{k,h}) \to \Diamond_T(G^+_{2n+3-2\bar{j},2}, G_{k,h}) \to f(L^+_{2\bar{i}+3,2\bar{j}+1}, K_{2\bar{j}-1,\bar{k},h+1})$, where the first $\to$ is given by $||M_{2\bar{j}+1,k,h}E^+_{2n+1-2\bar{j}}|| = d$, second $\to$ in the chain is given by Lemma 48*(j)*, and the third $\to$ is given by $||K_{2\bar{j}-1,k,h}G^+_{2n+3-2\bar{j},2}|| = d$. □

Lemma 53. $f(J^+_{2\bar{i}-1,2\bar{j}-1}, K_{2\bar{j}-1,\bar{k},h}) \to f(L^+_{2\bar{i}+3,2\bar{j}+1}, C_{k,h,2})$ *if* $Y_{\bar{j}}[h] \ne 0$.

Proof. $f(J^+_{2\bar{i}-1,2\bar{j}-1}, K_{2\bar{j}-1,\bar{k},h}) \to \Diamond_R(A^+_{2i-1,1}, A_{k,h,1}) \to \Diamond_L(D^+_{2i+3}, D_{k,h,1}) \to f(L^+_{2\bar{i}+3,2\bar{j}+1}, C_{k,h,2})$, where the second $\to$ in the chain is given by Lemma 48*(g)* (note $Y_{\bar{j}}[h] \neq 0 \implies X_i[h] \cdot Z_k[h] = 0$). □

Lemma 54. $f(J^+_{2\bar{i}-1,2\bar{j}-1}, D_{k,h,2}) \to f(L^+_{2\bar{i}+3,2\bar{j}+1}, K_{2\bar{j}-1,\bar{k},h+1})$

Proof. $f(J^+_{2\bar{i}-1,2\bar{j}-1}, D_{k,h,2}) \to f(E^+_1, D_{k,h,2}) \to \Diamond_T(G^+_{2n+1,2}, G_{k,h}) \to f(L^+_{2\bar{i}+3,2\bar{j}+1}, K_{2\bar{j}-1,\bar{k},h+1})$, where the second $\to$ in the chain is given by Lemma 48*(l)*. □

Lemma 55. $f(J^+_{2\bar{i}-1,2\bar{j}-1}, K_{2\bar{j}-1,\bar{k},W+1}) \to f(L^+_{2\bar{i}+3,2\bar{j}+1}, M_{2\bar{j}+1,\bar{k},W+1})$

Proof. The proof is essentially the same as the one for Lemma 51, except h is set to $W+1$. One subtlety is that $M_{j,k,W+1}$ is a point on $D_{k,W+1,2}C_{k,W+1,3}$. But the same monotone path persists since $\Diamond_L(D^+_{2\bar{i}+3}, D_{k,W+1,2})$ is the same shaped diamond as $\Diamond_L(D^+_{2\bar{i}+3}, D_{k,h,2})$ for $1 \leq h \leq W$. □

Hence, we have $m-1$ valid monotone paths that start at l_s and end at l_t. No pair of monotone paths overlap in y-coordinate in more than one point. No monotone path overlaps in y-coordinate with the y-interval from $s = J^+_{2\bar{i}-1,2\bar{j}-1}$ to $t = L^+_{2\bar{i}+3,2\bar{j}+1}$. These statements are true in both the $Y_{\bar{j}}[h] = 0$ and $Y_{\bar{j}}[h] \neq 0$ cases. Hence, we have shown that our constructed instance (T, m, ℓ, d) is a YES instance, yielding the following theorem.

Theorem 56. *If our input $(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$ is a YES instance for 3OV, then our constructed instance (T, m, ℓ, d) is a YES instance for SC.*

2.6.8 Cuts in free space

In the following section (Section 2.6.9), we will show that if $(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$ is a NO instance for 3OV, then (T, m, ℓ, d) is a NO instance for SC. This involves showing that for any pair of vertical lines l_s and l_t, there cannot be $m-1$ monotone paths from l_s to l_t with the property that the y-coordinate of the $m-1$ monotone paths overlap in at most one point, and the y-coordinates of the $m-1$ monotone paths intersect with the y-interval from s to t in at most one point. In actuality, we show the contrapositive of the above statement in Section 2.6.9, but the core idea of upper bounding the number of monotone paths remains the same.

To upper bound the number of monotone paths that can exist in our free space diagram, we require a significantly different (or in fact, opposite) strategy to Sections 2.6.6. In particular, we need a way to show that, for a point (s, z_1) on l_s and a point (t, z_2) on l_t, that there is no monotone path from $(s, z_1) \to (t, z_2)$. To show that there is no such monotone path, we construct a cutting sequence from (s, z_1) to (t, z_2). This is very similar in spirit to cuts constructed in Buchin et al.'s [42] lower bound.

First, let us define a cutting sequence. We say that $f(x, y)$ dominates $f(w, z)$ if $f(x, y)$ is strictly above and to the left of $f(w, z)$. In other words, $x > w$ and $y > z$. We say that $f(x, y)$ undermines $f(w, z)$ if $f(x, y)$ is vertically below $f(w, z)$, and the vertical segment between the two points is entirely in non-free space. We define a dominating sequence to be a sequence of points so that each point either dominates or undermines the next point in the sequence. We define a cutting sequence to be a dominating sequence where the first point dominates the second point, and the second to last point dominates the last point.

To simplify the description of our dominating and cutting sequences in this and subsequent sections, we use the following notation. Let $f(x, y) >_d f(w, z)$ denote that $f(x, y)$

dominates $f(w,z)$. Let $f(x,y) >_u f(w,z)$ denote that $f(x,y)$ undermines $f(w,z)$. Let $f(x,y) >_{du} f(w,z)$ denote that there is a dominating sequence from $f(x,y)$ to $f(w,z)$. See Figure 2.24.

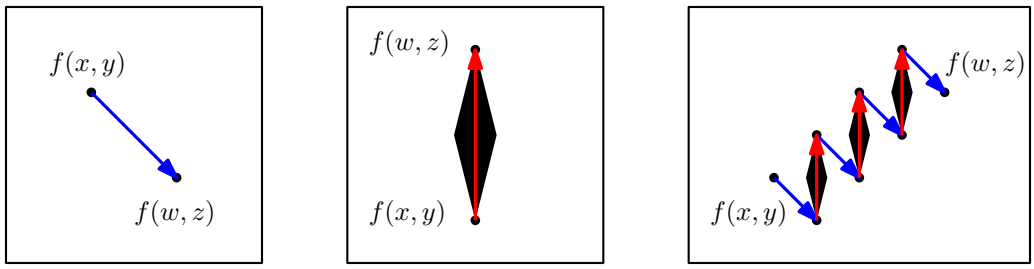


Figure 2.24: Left: $f(x,y) >_d f(w,z)$. Middle: $f(x,y) >_u f(w,z)$. Right: $f(x,y) >_{du} f(w,z)$.

With this notation in mind, we are ready to prove that if there is a cutting sequence from $f(x,y)$ to $f(w,z)$ that there cannot be a monotone path $f(x,y) \to f(w,z)$.

Lemma 57. *If there is a cutting sequence from $f(x,y)$ to $f(w,z)$, then there is no monotone path from $f(x,y)$ to $f(w,z)$.*

Proof. Let the cutting sequence be $f(x_1,y_1), f(x_2,y_2), \ldots, f(x_{i-1},y_{i-1}), f(x_i,y_i)$, where $f(x_1,y_1) = f(x,y)$ and $f(x_i,y_i) = f(w,z)$. Then $f(x_1,y_1) >_d f(x_2,y_2) >_{du} f(x_{i-1},y_{i-1}) >_d f(x_i,y_i)$. Suppose for the sake of contradiction that there is a monotone path from $f(x_1,y_1)$ to $f(x_i,y_i)$. See Figure 2.25.

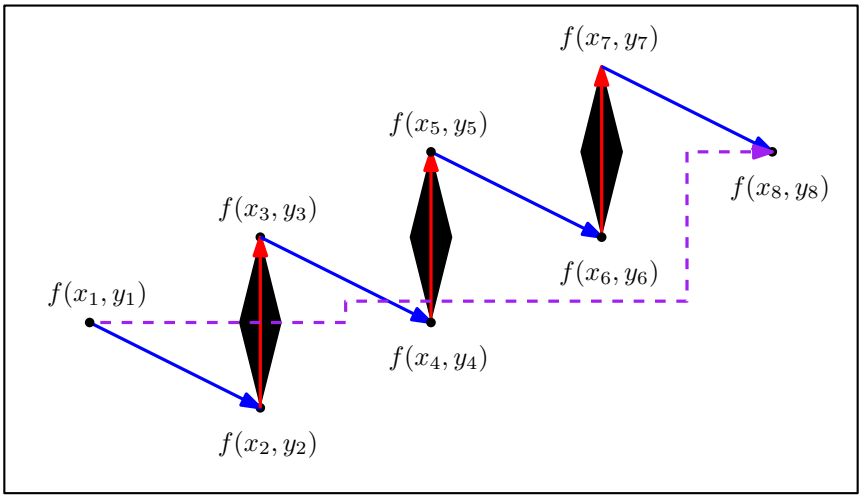


Figure 2.25: A cutting sequence from $f(x_1,y_1)$ to $f(x_i,y_i)$, for $i = 8$.

Since $f(x_1,y_1) >_d f(x_2,y_2)$, we know $f(x_2,y_2)$ is strictly below the monotone path $f(x_1,y_1) \to f(x_i,y_i)$. Since $f(x_{i-1},y_{i-1}) >_d (x_i,y_i)$, we know $f(x_{i-1},y_{i-1})$ is strictly above the monotone path $f(x_1,y_1) \to f(x_i,y_i)$. Therefore, a dominating sequence $f(x_2,y_2) >_{du} f(x_{i-1},y_{i-1})$ starts below the monotone path and ends above the monotone path.

If we ignore the y-coordinates and focus only on the x-coordinates, the dominating sequence $f(x_2, y_2) >_{du} f(x_{i-1}, y_{i-1})$ and the monotone path $f(x_1, y_1) \to f(x_i, y_i)$ are both weakly increasing. Therefore, there must be an x-coordinate where the dominating sequence $f(x_2, y_2) >_{du} f(x_{i-1}, y_{i-1})$ crosses from being below to being above the monotone path $f(x_1, y_1) \to f(x_i, y_i)$. At this point, the dominating sequence is increasing in y-coordinate. Therefore, we must have an undermining step at this x-coordinate. However, an undermining step only traverses non-free space, so it cannot cross the monotone path. This yields a contradiction. □

In this section, we compile a list of dominating sequences, and pairs of undermining points. We prove the correctness of these sequences. For now, we will not show any cutting sequences, but in Section 2.6.9, we will combine our dominating sequences and undermining points to form cutting sequences. Some of these dominating sequences are conditioned on one of the booleans $X_i[h]$, $Y_j[h]$ or $Z_k[h]$ being equal to one.

For several of our dominating sequences and undermining points, we require the following additional points. Let N_1, N_2 and N_3 be points on H_1H_2 so that the points H_1, N_1, N_2, N_3, H_2 are evenly spaced. Let $N_{k,1}$, $N_{k,2}$ and $N_{k,3}$ be the copies of N_1, N_2 and N_3 on the segment $H_{k,1}H_{k,2}$.

Lemma 58. *For $1 \le i, j, k, u \le n$ and $1 \le h \le W$:*

(a) $\Diamond_B(G^+_{1,1}, G_{k,h}) >_{du} \Diamond_B(C^+_1, C_{k,h,1})$

(b) $\Diamond_B(C^+_1, C_{k,h,1}) >_{du} \Diamond_T(G^+_{2n+1,2}, G_{k,h})$

(c) $\Diamond_B(A^+_{2i+1,1}, A_{k,h,1}) >_{du} \Diamond_T(C^+_{2i+3}, C_{k,h,1})$

(d) $\Diamond_B(E^+_1, E_{k,h}) >_{du} \Diamond_T(G^+_{1,2}, G_{k,h})$

(e) $\Diamond_B(A^+_{2i-1,2}, A_{k,h,2}) >_{du} \Diamond_T(D^+_{2i+2}, D_{k,h,2})$.

(f) $\Diamond_B(A^+_{2i,1}, A_{k,h,1}) >_{du} \Diamond_T(C^+_{2i+2}, C_{k,h,2})$ *if* $X_i[h] \cdot Z_k[h] = 1$.

(g) $\Diamond_B(E^+_{2j}, E_{k,h}) >_{du} \Diamond_T(G^+_{2j,2}, G_{k,h})$ *if* $Y_j[h] = 1$.

(h) $f(N_{k,3}, G_{1,1}) >_u f(N_{k,3}, N_{1,2})$

(i) $f(H_{k,1}, N_{u,1}) >_{du} f(N_{k,3}, N_{u+1,2})$

(j) $f(H_{k,1}, N_{n,1}) >_u f(H_{k,1}, D^+_{2n+3})$

(k) $f(G^+_{1,1}, H_{k-1,1}) >_u f(G^+_{1,1}, G_{k,h})$

(l) $f(H^-_1, H_{n,1}) >_{du} f(C^+_3, G^+_{2n+1,2})$

(m) $f(H^-_1, E^+_{2n+1}) >_u f(H^-_1, D^+_{2n+3})$

Proof.

(a) $\Diamond_B(G^+_{1,1}, G_{k,h}) >_u \Diamond_T(G^+_{1,1}, G_{k,h}) >_d \Diamond_B(A^+_{1,1}, A_{k,h,1}) >_u \Diamond_T(A^+_{1,1}, A_{k,h,1})$
$>_d \Diamond_B(A^+_{1,2}, A_{k,h,2}) >_u \Diamond_T(A^+_{1,2}, A_{k,h,2}) >_d \dots >_d \Diamond_B(A^+_{1,2W+1}, A_{k,h,2W+1})$
$>_u \Diamond_T(A^+_{1,2W+1}, A_{k,h,2W+1}) >_d \Diamond_B(C^+_1, C_{k,h,1})$

(b) $\Diamond_B(C_1^+, C_{k,h,1}) >_u \Diamond_T(C_1^+, C_{k,h,1}) >_d \Diamond_B(D_1^+, D_{k,h,1}) >_u \Diamond_T(D_1^+, D_{k,h,1}) >_d \Diamond_B(C_2^+, C_{k,h,2})$ $>_u \Diamond_T(C_2^+, C_{k,h,2}) >_d \Diamond_B(D_2^+, D_{k,h,2}) >_u \Diamond_T(D_2^+, D_{k,h,2}) >_d \Diamond_B(E_{2n+1}^+, E_{k,h}) >_u$ $\Diamond_T(E_{2n+1}^+, E_{k,h}) >_d \Diamond_B(F_{2n+1,1}^+, F_{k,h,1}) >_u \Diamond_T(F_{2n+1,1}^+, F_{k,h,1}) >_d \Diamond_B(F_{2n+1,2}^+, F_{k,h,2}) >_u$ $\Diamond_T(F_{2n+1,2}^+, F_{k,h,2}) >_d \ldots >_d \Diamond_B(F_{2n+1,2W+1}^+, F_{k,h,2W+1}) >_u \Diamond_T(F_{2n+1,2W+1}^+, F_{k,h,2W+1}) >_d$ $\Diamond_B(G_{2n+1,2}^+, G_{k,h+1}) >_u \Diamond_T(G_{2n+1,2}^+, G_{k,h+1})$

(c) $\Diamond_B(A_{2i+1,1}^+, A_{k,h,1}) >_u \Diamond_T(A_{2i+1,1}^+, A_{k,h,1}) >_d \Diamond_B(A_{2i+1,2}^+, A_{k,h,2}) >_u \Diamond_T(A_{2i+1,2}^+, A_{k,h,2})$ $>_d \ldots >_d \Diamond_B(A_{2i+1,2W+1}^+, A_{k,h,2W+1}) >_u \Diamond_T(A_{2i+1,2W+1}^+, A_{k,h,2W+1}) >_d \Diamond_B(B_{2i+1}^+, B_{k,h})$ $>_u \Diamond_T(B_{2i+1}^+, B_{k,h}) >_d \Diamond_B(C_{2i+3}^+, C_{k,h,1}) >_u \Diamond_T(C_{2i+3}^+, C_{k,h,1})$

(d) $\Diamond_B(E_1^+, E_{k,h}) >_u \Diamond_T(E_1^+, E_{k,h}) >_d \Diamond_B(E_1^+, E_{k,h}) >_u \Diamond_T(E_1^+, E_{k,h}) >_d \Diamond_B(F_{1,1}^+, F_{k,h,1})$ $>_u \Diamond_T(F_{1,1}^+, F_{k,h,1}) >_d \Diamond_B(F_{1,2}^+, F_{k,h,2}) >_u \Diamond_T(F_{1,2}^+, F_{k,h,2}) >_d \ldots >_d \Diamond_B(F_{1,2W+1}^+, F_{k,h,2W+1})$ $>_u \Diamond_T(F_{1,2W+1}^+, F_{k,h,2W+1}) >_d \Diamond_B(G_{1,2}^+, G_{k,h}) >_u \Diamond_T(G_{1,2}^+, G_{k,h})$

(e) $\Diamond_B(A_{2i-1,2}^+, A_{k,h,2}) >_u \Diamond_T(A_{2i-1,2}^+, A_{k,h,2}) >_d \Diamond_B(A_{2i-1,3}^+, A_{k,h,3}) >_u \Diamond_T(A_{2i-1,3}^+, A_{k,h,3})$ $>_d \ldots >_d \Diamond_B(A_{2i-1,2W+1}^+, A_{k,h,2W+1}) >_u \Diamond_T(A_{2i-1,2W+1}^+, A_{k,h,2W+1}) >_d \Diamond_B(B_{2i-1}^+, B_{k,h})$ $>_u \Diamond_T(B_{2i-1}^+, B_{k,h}) >_d \Diamond_B(C_{2i+1}^+, C_{k,h,1}) >_u \Diamond_T(C_{2i+1}^+, C_{k,h,1}) >_d \Diamond_B(D_{2i+1}^+, D_{k,h,1})$ $>_u \Diamond_T(D_{2i+1}^+, D_{k,h,1}) >_d \Diamond_B(C_{2i+2}^+, C_{k,h,2}) >_u \Diamond_T(C_{2i+2}^+, C_{k,h,2}) >_d \Diamond_B(D_{2i+2}^+, D_{k,h,2})$ $>_u \Diamond_T(D_{2i+2}^+, D_{k,h,2})$

(f) Since $X_i[h] = 1$, we have $\Diamond(A_{2i,2h}^+, A_{k,h,2h})$ is non-empty. Therefore, $\Diamond_T(A_{2i,2h-1}^+, A_{k,h,2h}) >_d$ $\Diamond_B(A_{2i,2h}^+, A_{k,h,2h}) >_u \Diamond_T(A_{2i,2h}^+, A_{k,h,2h}) >_d \Diamond_B(A_{2i,2h+1}^+, A_{k,h,2h})$. Since $Z_k[h] = 1$,

$\Diamond_T(B_{2i}^+, B_{k,h}) >_d \Diamond_B(C_{2i+1}^+, C_{k,h,1})$. Now, $\Diamond_B(A_{2i,1}^+, A_{k,h,1}) >_u \Diamond_T(A_{2i,1}^+, A_{k,h,1}) >_d$ $\Diamond_B(A_{2i,2}^+, A_{k,h,2}) >_u \Diamond_T(A_{2i,2}^+, A_{k,h,2}) >_d \ldots >_d \Diamond_B(A_{2i,2h}^+, A_{k,h,2h}) >_u \Diamond_T(A_{2i,2h}^+, A_{k,h,2h}) >_d$ $\ldots >_d \Diamond_B(A_{2i,2W+1}^+, A_{k,h,2W+1}) >_u \Diamond_T(A_{2i,2W+1}^+, A_{k,h,2W+1}) >_d \Diamond_B(B_{2i}^+, B_{k,h}) >_u$ $\Diamond_T(B_{2i}^+, B_{k,h})$
$>_d \Diamond_B(C_{2i+1}^+, C_{k,h,1}) >_u \Diamond_T(C_{2i+1}^+, C_{k,h,1}) >_d \Diamond_B(D_{2i+1}^+, D_{k,h,1}) >_u \Diamond_T(D_{2i+1}^+, D_{k,h,1})$
$>_d \Diamond_B(C_{2i+2}^+, C_{k,h,2}) >_u \Diamond_T(C_{2i+2}^+, C_{k,h,2})$

(g) Since $Y_j[h] = 0$, we have $\Diamond(F_{2j,2h}^+, F_{k,h,2h})$ is non-empty. Therefore, $\Diamond_T(F_{2j,2h-1}^+, F_{k,h,2h-1}) >_d$ $\Diamond_B(F_{2j,2h}^+, F_{k,h,2h}) >_u \Diamond_T(F_{2j,2h}^+, F_{k,h,2h}) >_d \Diamond_B(F_{2j,2h+1}^+, F_{k,h,2h+1})$. Now, $\Diamond_B(E_{2j}^+, E_{k,h})$ $>_u \Diamond_T(E_{2j}^+, E_{k,h}) >_d \Diamond_B(F_{2j,1}^+, F_{k,h,1}) >_u \Diamond_T(F_{2j,1}^+, F_{k,h,1}) >_d \Diamond_B(F_{2j,2}^+, F_{k,h,2}) >_u$ $\Diamond_T(F_{2j,2}^+, F_{k,h,2}) >_d \ldots >_d \Diamond_B(F_{2j,2h}^+, F_{k,h,2h}) >_u \Diamond_T(F_{2j,2h}^+, F_{k,h,2h}) >_d \ldots >_d$ $\Diamond_B(F_{2j,2n+1}^+, F_{k,h,2n+1}) >_u \Diamond_T(F_{2j,2n+1}^+, F_{k,h,2n+1}) >_d \Diamond_B(G_{2j,2}^+, G_{k,h}) >_u \Diamond_T(G_{2j,2}^+, G_{k,h})$

(h) Recall that $H_1 = 4r' \operatorname{cis}(3W+4) \cdot \phi$ and $H_2 = 8r' \operatorname{cis}(2W+3) \cdot \phi$. So the distance between consecutive pairs in H_1, N_1, N_2, N_3, H_2 is greater than d. The subtrajectory from $G_{1,1}$ to $N_{1,2}$ is closest to the point N_3 at its endpoint, but $||N_{1,2}N_3|| > d$. Hence, for all points x on the subtrajectory between $G_{1,1}$ to $N_{1,2}$, we have $||xN_{k,3}|| > r' > d$. Hence, $f(N_{k,3}, G_{1,1}) >_u f(N_{k,3}, N_{1,2})$.

(i) Let M be a point so that $H_{u,2} \prec M$ and $||H_{u,2}M|| = 2d$. We show that $f(H_{k,1}, N_{u,1}) >_u$ $f(H_{k,1}, M) >_d f(N_{k,3}, H_{u,2}) >_u f(N_{k,3}, N_{u+1,2})$. The subtrajectory from $N_{u,1}$ to M lies outside the circle of radius $2r'$ centered at the origin, but $H_{k,1}$ is within distance r' of the origin. Hence, $f(H_{k,1}, N_{u,1}) >_u f(H_{k,1}, M)$. Similarly, $f(N_{k,3}, H_{u,2}) >_u$ $f(N_{k,3}, N_{u+1,2})$.

(j) The distances between $H_{k,1}$ and the segments $N_{n,1}H_{2,n}$ and $H_{2,n}D_0^-$ is strictly greater than d. Of all the points on the subtrajectory from D_0^- to D_{2n+3}^+, the point closest to $H_{k,1}$ is H_1^-, as all other vertices lie below the real axis on the complex plane. Since $||H_1^- H_{k,1}|| > d$, we have, for every point x on segment $N_{n,1}H_{2,n}$, that $f(H_{k,1}, x)$ is in non-free space. Therefore, $f(H_{k,1}, N_{n,1}) >_u f(H_{k,1}, D_{2n+3}^+)$.

(k) Let x be a point on the segment $G_{k,h}H_{k,1}$. Note that as x moves from $G_{k,h}$ to $H_{k,1}$, the distance from x to $G_{1,1,}^+$ increases. But $||G_{k,h}G_{1,1}^+|| > d$, so $||xG_{1,1}^+|| > d$. Therefore, $f(G_{1,1}^+, x)$ is in non-free space for all x on the segment $G_{k,h}H_{k,1}$. Thus, $f(G_{1,1}^+, H_{k-1,1}) >_u f(G_{1,1}^+, G_{k,h})$.

(l) We show that $f(H_1^-, H_{n,1}) >_u f(H_1^-, D_2^+) >_d f(C_3^+, D_2^-) >_u f(C_3^+, G_{2n+1,2}^+)$. The point H_1^- is distance d away from the segments $H_{n,1}H_{n,2}$ and $H_{n,2}D_0^-$. The subtrajectory from D_0^- to D_2^+ lies entirely below the real axis in the complex plane, so the distance from H_1^- to any of these points is greater than d. Therefore, $f(H_1^-, H_{n,1}) >_u f(H_1^-, D_2^+)$. The distance from C_3^+ to any point on the subtrajectory starting at D_2^- and ending at $G_{2n+1,2}^+$ is greater than d. Therefore, $f(C_3^+, D_2^-) >_u f(C_3^+, G_{2n+1,2}^+)$. Putting this together yields the dominating sequence.

(m) The subtrajectory from E_{2n+1}^+ to D_{2n+3}^+ lies entirely below the real axis in the complex plane, so the distance from H_1^- to any of these points is at least d. Therefore, $f(H_1^-, x)$ is in non-free space for all points x on the segment $E_{2n+1}^+ D_{2n+3}^+$. This implies the claimed lemma. □

2.6.9 NO instances

In this section, we show that if our input $(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$ to 3OV is a NO instance, then our constructed instance (T, m, ℓ, d) for SC is a NO instance.

We do this by taking the contrapositive. We show that if our constructed instance (T, m, ℓ, d) is a YES instance for SC, then our input $(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$ is a YES instance for 3OV.

Since (T, m, ℓ, d) is a YES instance, there exists a pair of vertical lines l_s and l_t so that the subtrajectory from s to t has length at least ℓ, there are $m - 1 = 2nW + 1$ monotone paths from l_s to l_t, no two of these monotone paths overlap in y-coordinate, and no monotone path overlaps in y-coordinate with the y-interval from s to t.

The remainder of this section consists of five steps. The first step is to narrow down the starting position s of the reference subtrajectory. The second step is to show that there are $2W + 1$ monotone paths from l_s to l_t between the y-coordinates of $H_{k-1,1}$ and $H_{k,1}$, for some $1 \leq k \leq n$. The third step is to show that s is between $G_{2i-1,1}^+$ and $G_{2i+1,1}^+$, for some $1 \leq i \leq n$. The fourth step is to show that s is between $J_{2i-1,2j-1}^+$ and $J_{2i-1,2j+1}$ for some $1 \leq j \leq n$. The fifth step is to show that, for the integers i, j and k in our second, third and fourth steps, that X_i, Y_j, Z_k are orthogonal. Putting our five steps together shows that our input $(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$ is a YES instance, as required.

As mentioned above, our first step is to narrow down the position of the starting point s. We begin with some useful notation, and then prove that s is between $H_{n,1}$ and $G_{2n+1,1}^+$ in Lemma 60.

Definition 59. *We say $x \prec y$ if x precedes y in the trajectory T.*

Lemma 60. $H_{n,1} \prec s \prec G_{2n+1,1}^+$

Proof. We first show $s \prec G^+_{2n+1,1}$. By our construction, the subtrajectory from $A^+_{2n-1,1}$ to D^+_{2n+3} has length $\ell + \delta$. Hence, the subtrajectory from $G^+_{2n+1,1}$ to D^+_{2n+3} has length less than ℓ. Since D^+_{2n+3} is the final vertex of T, we have $t \preceq D^+_{2n+3}$. Since the subtrajectory from s to t has length at least ℓ, we have that $s \prec G^+_{2n+1,1}$.

Next, we show $H_{n,1} \prec s$. Suppose for the sake of contradiction that $s \preceq H_{n,1}$. Without loss of generality, let $H_{k-1,1} \preceq s \preceq H_{k,1}$ for some $1 \leq k \leq n$. By Lemma 43, we must have $s \preceq H_k \prec H_{k,2} \prec t$. Let the first $n+1$ monotone paths from l_s to l_t be $f(s, y_i) \to f(t, z_i)$ for $1 \leq i \leq n+1$.

We will prove by induction that $N_{i,2} \preceq z_i$ for all $1 \leq i \leq n$. In the base case, suppose for the sake of contradiction that $z_1 \prec N_{1,2}$. Then $f(s, y_1) >_d f(N_{k,3}, G_{1,1}) >_u f(N_{k,3}, N_{1,2}) >_d f(t, z_1)$ by Lemma 58*(h)*. There is a cutting sequence from $f(s, y_1)$ to $f(t, z_1)$, which yields a contradiction.

Now we prove the inductive case. Assume the inductive hypothesis that $N_{i,2} \preceq z_i$. Then $N_{i,2} \preceq y_{i+1}$, since our monotone paths do not overlap in y-coordinate. Suppose for the sake of contradiction that $z_{i+1} \prec N_{i+1,2}$. Then $f(s, y_{i+1}) >_d f(H_{k,1}, N_{i,1}) >_{du} f(N_{k,3}, N_{i+1}, 2) >_d f(t, z_{i+1})$ by Lemma 58*(i)*. There is a cutting sequence from $f(s, y_{i+1})$ to $f(t, z_{i+1})$, which is a contradiction. This completes our induction.

Setting $i = n$ in our induction, we get that $N_{n,1} \preceq z_n$. Therefore, $N_{n,1} \preceq y_{n+1}$. Now, by Lemma 58*(j)* we have the cutting sequence $f(s, y_{n+1}) >_d f(H_{k,1}, N_{n,1}) >_u f(H_{k,1}, D^+_{2n+3}) >_d f(t, z_{n+1})$, yielding the final contradiction. Therefore, our initial assumption that $s \preceq H_{n,1}$ cannot hold, and we are done. □

Next, we show that there is at most no monotone paths in the region of free space between y-coordinates of $H_{n,1}$ and D^+_{2n+3} that do not intersect the y-interval from s to t. This fact will be useful for the second step in this section, which was to show that there are $2W + 1$ monotone paths associated with the region between y-coordinates of $H_{k-1,1}$ and $H_{k,1}$, for some $1 \leq k \leq n$.

Lemma 61. *In the free space with bottom left corner $f(s, H_{n,1})$ and top right corner $f(t, D^+_{2n+3})$, any monotone path from l_s to l_t overlaps in y-coordinate with the y-interval from s to t in more than one point.*

Proof. Let $f(s, y_1) \to f(t, y_2)$ be a monotone path in the free space with bottom left corner $f(s, H_{n,1})$ and top right corner $f(t, D^+_{2n+3})$. Therefore, $H_{n,1} \preceq y_1 \preceq y_2 \preceq D^+_{2n+3}$. Suppose $y_2 \prec G^+_{2n+1,2}$. Then $f(s, y_1) >_d f(H^-_1, H_{n,1}) >_{du} f(C^+_3, G^+_{2n+1,2}) >_d f(t, y_2)$ by Lemma 58*(l)*. There is a cutting sequence from $f(s, y_1)$ to $f(t, y_2)$, which is a contradiction. Hence, $G^+_{2n+1,2} \preceq y_2$. Suppose, $E^+_{2n+1} \prec y_1$. By Lemma 58*(m)*, $f(s, y_1) >_d f(H^-_1, E^+_{2n+1}) >_u f(H^-_1, D^+_{2n+3}) >_d f(t, y_2)$, which contradicts the fact that $f(s, y_1) \to f(t, y_2)$ is a monotone path. Hence, $y_1 \preceq E^+_{2n+1}$. Therefore $H_{n,1} \preceq y_1 \preceq E^+_{2n+1} \prec G^+_{2n+1,2} \preceq y_2 \preceq D^+_{2n+3}$. But we also have $H_{n,1} \preceq s \prec G^+_{2n+1,1}$, so $E^+_{2n+1} \prec t \prec D^+_{2n+3}$. Hence, the y-interval of the monotone path $f(s, y_1) \to f(t, y_2)$ intersects the y-interval from s to t in more than one point. □

Now we are ready to prove the second step of this section.

Lemma 62. *Let $H_{0,1} = G_{1,1}$. There exists $1 \leq k \leq n$ so that, in the free space with bottom left corner $f(s, H_{k-1,1})$ and top right corner $f(t, H_{k,1})$, there are at least $2W + 1$ monotone paths with non-overlapping y-coordinates.*

Proof. There are $m-1 = 2nW+1$ monotone in the free space with bottom left corner $f(s, H_{0,1})$ and top right corner $f(t, D^+_{2n+3})$. By Lemma 61 there are no monotone paths in the free space with bottom left corner $f(s, H_{n,1})$ and top right corner $f(t, D^+_{2n+3})$ that do not intersect the y-interval from s to t. Therefore, all $2nW+1$ monotone paths are in the free space with bottom left corner $f(s, H_{0,1})$ and top right corner $f(t, H_{n,1})$. By pigeonhole principle, there exists $1 \leq k \leq n$ so that there are $2W+1$ monotone paths in the free space with bottom left corner $f(s, H_{k-1,1})$ and top right corner $f(t, H_{k,1})$. □

For the remainder of this section, assume that $1 \leq k \leq n$ is an integer so that there are at least $2W+1$ monotone paths in the free space with bottom left corner $f(s, H_{k-1,1})$ and top right corner $f(t, H_{k,1})$. Using these paths we can narrow down the position of s further.

Lemma 63. $G^+_{1,1} \preceq s \prec G_{2n+1,1}$

Proof. Suppose for the sake of contradiction that $H_{n,1} \prec s \prec G^+_{1,1}$. Let the monotone paths in the free space with bottom left corner $f(s, H_{k-1,1})$ and top right corner $f(t, H_{k,1})$ be $f(s, y_i) \to f(t, z_i)$ for $1 \leq i \leq 2W+1$.

We will prove that $G_{k,i+1} \prec z_i$ for $1 \leq i \leq W$ by induction. Suppose for the sake of contradiction that $z_1 \preceq G_{k,2}$. Then by Lemmas (k), (a) and (b) we have the cutting sequence $f(s, y_1) >_d f(G^+_{1,1}, H_{k-1,1}) >_u f(G^+_{1,1}, G_{k,h}) >_{du} \Diamond_B(C^+_1, C_{k,h,1} >_{du} \Diamond_T(G^+_{2n+1,2}, G_{k,h+1}) >_d f(t, z_1)$, contradicting $f(s, y_1) \to f(t, z_1)$.

Now we prove the inductive case. Assume the inductive hypothesis $G_{k,i+1} \prec z_i$. Then $G_{k,i+1} \prec y_{i+1}$. Suppose for the sake of contradiction that $z_{i+1} \preceq G_{k,i+2}$. Then by Lemma 58*(a)* and *(b)* we have the cutting sequence $f(s, y_{i+1}) >_d \Diamond_B(G^+_{1,1}, G_{k,i+1}) >_{du} \Diamond_B(C^+_1, C_{k,h,1} >_{du} \Diamond_T(G^+_{2n+1,2}, G_{k,i+2}) >_d f(t, z_{i+1})$, contradicting the fact that $f(s, y_{i+1}) \to f(t, z_{i+1})$. This completes the induction.

Setting $i = W$ yields $G_{k,W+1} \prec z_W$. So $G_{k,W+1} \prec y_{W+1}$. But now, by Lemma, $f(s, y_{W+1}) >_d f(G^+_{1,1}, G_{k,W+1}) >_{du} f(C^+_3, H_{k,1}) >_d f(t, z_{W+1})$, contradicting $f(s, y_{W+1}) \to f(t, z_{W+1})$. This yields a contradiction on our initial assumption, so $G^+_{1,1} \preceq s$ as required. □

We can now assume without loss of generality that $G^+_{2i-1,1} \prec s \preceq G^+_{2i+1,1}$ for some $1 \leq i \leq n$. Recall that this was the third step of this section. Next, we prove a similar result towards the fourth step.

Lemma 64. $J^+_{2i-1,1} \prec s \preceq J^+_{2i-1,2n+1}$

Proof. It suffices to rule out the cases $G^+_{2i-1,1} \prec s \preceq J^+_{2i-1,1}$, and $J^+_{2i-1,2n+1} \prec s \preceq G^+_{2i+1,1}$. We will focus on the $2W+1$ paths in the free space with bottom left corner $f(s, H_{k-1,1})$ and top right corner $f(t, H_{k,1})$ be $f(s, y_i) \to f(t, z_i)$. The reason we can do this is that, for our final lemma, Lemma 65, we only require that $J^+_{2i-1,1} \prec s \preceq J^+_{2i-1,2n+1}$ and that there are $2W+1$ paths.

We start with the $G^+_{2i-1,1} \prec s \preceq J^+_{2i-1,1}$ case. We will modify the $2W+1$ monotone paths so that $s = J^+_{2i-1,1}$. By Lemma 42, we have $C^+_{2n+3} \prec t \preceq L^+_{2i+3,3}$. Let the monotone paths in the free space with bottom left corner $f(s, H_{k-1,1})$ and top right corner $f(t, H_{k,1})$ be $f(s, y_i) \to f(t, z_i)$ for $1 \leq i \leq 2W+1$. Define $y'_i = A_{k,h,2}$ if $G_{k,h} \prec y_i \prec A_{k,h,2}$, otherwise $y'_i = y_i$. Define $z'_i = C_{k,h,2}$ if $D_{k,h,1} \preceq z_i \prec D_{k,h,2}$, define $z'_i = K^+_{1,k,h}$ if $D_{k,h,2} \preceq z_i \preceq K^+_{1,k,h}$ otherwise $z'_i = z_i$. We claim that if $f(s, y_i) \to f(t, z_i)$, then $f(J^+_{2i-1,1}, y'_i) \to f(L^+_{2i+3,3}, z'_i)$. If $G_{k,h} \prec y_i \prec A_{k,h,2}$, then $f(s, y_i)$ is to the right of $\Diamond(A^+_{2i-1,1}, A_{k,h})$ and we can replace the

starting point $f(s, y_i)$ with $f(J^+_{2i-1,1}, A_{k,h,2})$. Otherwise, the segment between $f(s, y_i)$ and $f(J^+_{2i-1,1}, y_i)$ is free space, so we can replace the starting point $f(s, y_i)$ with $f(J^+_{2i-1,1}, y_i)$. Hence, we have $f(J^+_{2i-1,1}, y'_i) \to f(t, z_i)$. If $D_{k,h,1} \preceq z_i \prec D_{k,h,2}$, then $f(t, z_i)$ is to the right of $\Diamond(C^+_{2i+3}, C_{k,h,2})$, so the monotone path $f(J^+_{2i-1,1}, y'_i) \to f(t, z_i)$ must have passed under $\Diamond_B(C^+_{2i+3}, C_{k,h,2})$. Hence, we can replace the ending point $f(t, z_i)$ with $f(L^+_{2i+3,3}, C_{k,h,2})$. If $D_{k,h,2} \preceq z_i \preceq K^+_{1,k,h}$ then we append the monotone path $f(t, z_i) \to f(L^+_{2i+3,3}, K^+_{1,k,h})$ to obtain the monotone path $f(J^+_{2i-1,1}, y'_i) \to f(L^+_{2i+3,3}, K^+_{1,k,h})$. Otherwise, the horizontal segment $f(t, z_i) \to f(L^+_{2i+3,3}, z_i)$ is free space, so we can append this to our path. Hence, we have modified all $2W + 1$ monotone paths so that $s = J^+_{2i-1,1}$ and $t = L^+_{2i+3,3}$ as required.

Next we consider the $J^+_{2i-1,2n+1} \prec s \preceq G^+_{2i+1,1}$. First, note that the subtrajectory from $J^+_{2n-1,2n+1}$ to D^+_{2n+3} has length less than ℓ, so we only need to consider the case where $1 \le i < n$. The remainder of this proof is very similar to the previous case. We modify the $2W+1$ monotone paths. Our modification moves the starting x-coordinate to $J^+_{2i-1,2n+1}$ and the ending x-coordinate to $L^+_{2i+3,2n+1}$. For the starting x-coordinate, we observe that since $s \le G^+_{2i+1,1}$, the point $f(s, y_i)$ is to the left of $\Diamond(G^+_{2i+1,1}, G_{k,h})$. Therefore, the horizontal segment from $f(J^+_{2i-1,2n+1}, y_i)$ to $f(s, y_i)$ must be free space. Therefore, we can prepend this to the monotone path to obtain $f(J^+_{2i-1,2n+1}, y_i) \to f(t, z_i)$. We truncate the monotone path at the y-coordinate $L^+_{2i+3,2n+1}$ to obtain $f(J^+_{2i-1,2n+1}, y_i) \to f(L^+_{2i+3,2n+1}, z'_i)$ for some $z'_i \prec z_i$. Therefore, we modified all $2W + 1$ monotone paths so that $s = J^+_{2i-1,1}$ and $t = L^+_{2i+3,3}$ as required.

Putting this all together, we obtain that there are $2W + 1$ paths in the free space with bottom left corner $f(s, H_{k-1,1})$ and top right corner, and $J^+_{2i-1,1} \prec s \preceq J^+_{2i-1,2n+1}$ for some $1 \le i \le n$. □

For the remainder of this section, we can assume that $J_{2i-1,2j-1} \prec s \preceq J_{2i-1,2j+1}$ for some $1 \le j \le n$. Recall that this was the fourth step of this section.

Now we are ready to prove the fifth and final step of this section. We use the $2W + 1$ paths from Lemma 62 and the cutting sequences from Lemma 57 to show that that X_i, Y_j and Z_k are orthogonal, and therefore that $(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$ is a YES instance.

Lemma 65. *Suppose $J_{2i-1,2j-1} \prec s \preceq J_{2i-1,2j+1}$ and there are $2W + 1$ monotone paths in the free space diagram with bottom left corner $f(s, H_{k-1})$ and top right corner $f(t, H_k)$. Then*

$$X_i[h] \cdot Y_j[h] \cdot Z_k[h] = 0 \text{ for all } 1 \le h \le W.$$

Proof. Let the $2W + 1$ monotone paths be $f(s, y_i) \to f(t, z_i)$ for $1 \le i \le 2W + 1$. Suppose for the sake of contradiction that there exists $1 \le \bar{h} \le W$ such that $X_i[\bar{h}] \cdot Y_j[\bar{h}] \cdot Z_k[\bar{h}] = 1$. We will prove the following series of statements by induction:

(i) for $1 \le h < \bar{h}$, we have $C_{k,h,1} \preceq z_{2h-1}$, $G_{k,h} \prec z_{2h}$,

(ii) for $h = \bar{h}$, we have $M_{2j+1,k,\bar{h}} \prec z_{2\bar{h}-1}$, $K_{2j-1,k,\bar{h}+1} \prec z_{2\bar{h}}$,

(iii) for $\bar{h} < h \le W$, we have $E_{k,h,2} \preceq z_{2h-1}$, $A_{k,h+1,1} \prec z_{2h}$.

After proving these three statements, we will use them to yield a contradiction. But first, we will prove *(i)*, *(ii)* and *(iii)*, in order:

(i) Let $1 \le u \le 2\bar{h} - 2$. We prove the following by induction on u: if u is odd, let $u = 2h - 1$, and we show that $C_{k,h,1} \preceq z_{2h-1}$, whereas if u is even, let $u = 2h$, and we show that $G_{k,h} \prec z_{2h}$.

We start with the base case $u = 1$. Suppose for the sake of contradiction that $z_1 \prec C_{k,1,2}$. Then by Lemma 58*(c)*, we have $f(s, y_1) >_d f(G^+_{3,1}, H_{k-1,1}) >_u \Diamond_T(G^+_{3,1}, G_{k,h}) >_d \Diamond_B(A^+_{2i+1,1}, A_{k,h,1}) >_{du} \Diamond_T(C^+_{2i+3}, C_{k,h,1}) >_d f(t, z_1)$, contradicting the claim that $f(s, y_1) \to f(t, z_1)$. This completes the base case. Next, we prove the inductive case of (i).

We split the inductive case into two subcases, either u is odd or u is even.

In the inductive case, if u is odd, we assume the inductive hypothesis for $u-1$, which is even. Let $u = 2h-1$. Our inductive hypothesis states that $G_{k,h-1,2} \prec z_{2h-2}$, from which we get $G_{k,h-1,2} \prec y_{2h-1}$. Suppose for the sake of contradiction that $z_{2h-1} \prec C_{k,h,1}$. Then by Lemma 58*(c)*, we have $f(s, y_{2h-1}) >_d \Diamond_B(A^+_{2i+1,1}, A_{k,h,1}) >_{du} \Diamond_T(C^+_{2i+3}, C_{k,h,1}) >_d f(t, z_{2h-1})$, contradicting $f(s, y_{2h-1}) \to f(t, z_{2h-1})$. So $C_{k,h,1} \preceq z_{2h-1}$, as required.

In the inductive case, if u is even, we assume the inductive hypothesis for $u-1$, which is odd. Let $u = 2h$. Our inductive hypothesis states that $C_{k,h,1} \preceq z_{2h-1}$, from which we get $C_{k,h,1} \preceq y_{2h}$. Suppose for the sake of contradiction that $z_{2h} \preceq G_{k,h}$. Then by Lemma 58*(b)*, $f(s, y_{2h}) >_d \Diamond_B(C^+_1, C_{k,h,1}) >_{du} \Diamond_T(G^+_{2n+1,2}, G_{k,h}) >_d f(t, z_{2h})$, contradicting $f(s, y_{2h}) \to f(t, z_{2h})$. So $G_{k,h} \prec z_{2h}$, as required.

This completes the proof by induction of (i).

(ii) For $h = \bar{h}$, recall the following definitions. Recall that $\bar{h}$ is defined so that $X_i[\bar{h}] \cdot Y_j[\bar{h}] \cdot Z_k[\bar{h}] = 1$. Recall that $K_{2j-1,k,h}$ is the point on $G_{k,h}A_{k,h,1}$ so that $||K_{2j-1,k,h}J^+_{i,2j-1}|| = d$ and $||K_{2j-1,k,h}G^+_{2n+3-2j,2}|| = d$. Recall that $M_{2j+1,k,h}$ is the point on $D_{k,h,2}E_{k,h}$ so that $||M_{2j+1,k,h}L^+_{i,2j+1}|| = d$ and $||M_{2j+1,k,h}E^+_{2n+1-2j}|| = d$. Recall that $J_{2i-1,2j-1} \prec s \preceq J_{2i-1,2j+1}$, so by Lemma 42, we have $L_{2i+3,2j+1} \prec t \preceq L_{2i+3,2j+3}$.

We begin by showing $M_{2j+1,k,\bar{h}} \prec z_{2\bar{h}-1}$. Assume for the sake of contradiction that $z_{2\bar{h}-1} \preceq M_{2j+1,k,\bar{h}}$. Since $L_{2i+3,2j+1} \prec t$, we have $f(t, C_{k,h,2}) >_u f(t, M_{2j+1,k,\bar{h}})$. Therefore, $z_{2\bar{h}-1} \prec C_{k,h,2}$. By (i) we have $G_{k,h} \prec z_{2\bar{h}-2} \preceq y_{2\bar{h}-1}$. So $f(s, y_{2\bar{h}-1}) >_d \Diamond_B(A^+_{2i,1}, A_{k,h,1})$. Since $X_i[h] \cdot Z_k[h] = 1$, by Lemma 58*(f)*, we have that $\Diamond_B(A^+_{2i,1}, A_{k,h,1}) >_{du} \Diamond_T(C^+_{2i+2}, C_{k,h,2})$. Finally, since $z_{2\bar{h}-1} \prec C_{k,h,2}$, we have $\Diamond_T(C^+_{2i+2}, C_{k,h,2}) >_d f(s, z_{2\bar{h}-1})$. Putting this together we obtain the cutting sequence $f(s, y_{2\bar{h}-1}) >_d \Diamond_B(A^+_{2i,1}, A_{k,h,1}) >_{du} \Diamond_T(C^+_{2i+2}, C_{k,h,2}) >_d f(s, z_{2\bar{h}-1}$, contradicting the fact that $f(s, y_{2\bar{h}-1}) \to f(s, z_{2\bar{h}-1})$. Therefore, $M_{2j+1,k,\bar{h}} \prec z_{2\bar{h}-1}$, as required.

Next, we show $K_{2j-1,k,\bar{h}+1} \prec z_{2\bar{h}}$. Assume for the sake of contradiction that $z_{2\bar{h}} \preceq K_{2j-1,k,\bar{h}+1}$. We know that $M_{2j+1,k,\bar{h}} \prec z_{2\bar{h}-1}$. Therefore, $M_{2j+1,k,\bar{h}} \prec y_{2\bar{h}}$. Since $Y_j[h] = 1$, by Lemma 58*(g)*, we have that $f(s, y_{2\bar{h}}) >_d \Diamond_B(E^+_{2n+1-2j}, E_{k,\bar{h}}) >_{du} \Diamond_T(G^+_{2n+1-2j,2}, G_{k,\bar{h}}) >_d f(t, z_{2\bar{h}})$, contradicting the fact that $f(s, y_{2\bar{h}}) \to f(t, z_{2\bar{h}})$. Therefore, $K_{2j-1,k,\bar{h}+1} \prec z_{2\bar{h}}$, as required.

(iii) Let $2\bar{h}+1 \leq u \leq 2W$. We prove the following by induction on u: if u is odd, let $u = 2h-1$, and we show that $E_{k,h,2} \preceq z_{2h-1}$, whereas if u is even, let $u = 2h$, and we show that $A_{k,h+1,1} \prec z_{2h}$.

We start with the base case $u = 2\bar{h}+1$. Suppose for the sake of contradiction that $z_{2\bar{h}+1} \prec E_{k,\bar{h}+1,2}$. By (ii) we know that $K_{2j-1,k,\bar{h}+1} \prec z_{2\bar{h}}$, which implies that $K_{2j-1,k,\bar{h}_1} \prec y_{2\bar{h}+1}$. Since $s \preceq J_{2i-1,2j+1}$, we have $f(s, K_{2j-1,k,\bar{h}+1}) >_u f(s, A_{k,\bar{h}+1,2})$. Therefore, $A_{k,\bar{h}+1,2} \prec y_{2\bar{h}+1}$. Now, by Lemma 58*(e)*, $f(s, y_{2\bar{h}+1}) >_d \Diamond_B(A^+_{2i-1,2}, A_{k,\bar{h}+1,2})$

$>_{du} \Diamond_T(D^+_{2i+2}, D_{k,\bar{h}+1,2}) >_d f(t, z_{2\bar{h}+1})$, contradicting $f(s, y_{2\bar{h}+1}) \to f(t, z_{2\bar{h}+1})$. Hence, $E_{k,\bar{h}+1,2} \preceq z_{2\bar{h}+1}$ as required. This completes the base case. Next, we prove the inductive case of *(iii)*.

We split the inductive case into two subcases, either u is odd or u is even.

In the inductive case, if u is odd, we assume the inductive hypothesis for $u-1$, which is even. Since the base case $u = 2\bar{h}+1$ is already handled, we can assume that $2\bar{h}+2 \leq u-1$. Let $u = 2h-1$. Our inductive hypothesis states that $A_{k,h,1} \prec z_{2h-2}$, which implies $A_{k,h,1} \prec y_{2h-1}$. Suppose for the sake of contradiction that $z_{2h-1} \prec E_{k,h,2}$. Then by Lemma 58*(e)* we have $f(s, y_{2h-1}) >_d \Diamond_B(A^+_{2i-1,2}, A_{k,h,2}) >_{du} \Diamond_T(D^+_{2i+2}, D_{k,h,2}) >_d f(t, z_{2h-2})$, contradicting the claim that $f(s, y_{2h-1}) \to f(t, z_{2h-1})$. Hence, $E_{k,h,2} \preceq z_{2h-1}$, as required.

In the inductive case, if u is even, we assume the inductive hypothesis for $u-1$, which is odd. Let $u = 2h$. Our inductive hypothesis states that $E_{k,h,2} \preceq z_{2h-1}$, which which we get $E_{k,h,2} \preceq y_{2h}$. Suppose for the sake of contradiction that $z_{2h} \preceq A_{k,h+1,1}$. Then by Lemma 58*(d)* we get $f(s, y_{2h}) >_d \Diamond_B(E^+_1, E_{k,h}) >_{du} \Diamond_T(G^+_{1,2}, G_{k,h}) >_d f(t, z_{2h})$, contradicting the fact that $f(s, y_{2h}) \to f(t, z_{2h})$. Therefore, $A_{k,h+1,1} \prec z_{2h}$, as required.

This completes the proof by induction of *(iii)*.

Finally, set $h = W$ in statement *(iii)* to get $A_{k,W+1,1} \prec z_{2W}$. Therefore, $A_{k,W+1,1} \prec y_{2W+1}$. By Lemma 58*(e)*, $f(s, y_{2W+1}) >_d \Diamond_B(A^+_{2i-1,2}, A_{k,W+1,2}) >_{du} \Diamond_T(D^+_{2i+2}, D_{k,W+1,2}) >_u f(D^+_{2i+2}, H_{k,1}) >_d f(t, z_{2W+1})$. This contradicts the fact that $f(s, y_{2W+1}) \to f(t, z_{2W+1})$.

Hence, our initial assumption that there exists $1 \leq \bar{h} \leq W$ such that $X_i[\bar{h}] \cdot Y_j[\bar{h}] \cdot Z_k[\bar{h}] = 1$ cannot hold. Therefore, X_i, Y_j and Z_k are orthogonal. □

Putting this all together yields the main theorem of Section 2.6.9.

Theorem 66. *If our input $(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$ is a NO instance for 3OV, then our constructed instance (T, m, ℓ, d) is a NO instance for SC.*

2.6.10 Putting it all together

Finally, we combine Theorems 56 and 66 to obtain our main theorem of Section 2.6.

Theorem 7. *There is no $O(n^{3-\varepsilon})$ time algorithm for SC under the continuous Fréchet distance, for any $\varepsilon > 0$, unless SETH fails.*

Proof. Suppose for the sake of contradiction that there is an $O(n^{3-\varepsilon})$ time algorithm for SC under the continuous Fréchet distance, for some $\varepsilon > 0$. We will use this to construct an $O(n^{3-\varepsilon}W^{O(1)})$ time algorithm for 3OV, which cannot occur unless SETH fails [165].

Our algorithm for 3OV is as follows. We obtain our input $(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$ for 3OV. We construct the instance (T, m, ℓ, d) as described in Section 2.6.1. Next, we run our algorithm for SC under the continuous Fréchet distance. If our algorithm returns YES for (T, m, ℓ, d), we return YES for $(\mathcal{X}, \mathcal{Y}, \mathcal{Z})$, likewise in the NO case. This completes the description of our algorithm.

The correctness of our algorithm follows directly from Theorems 56 and 66. Finally, we analyse the running time of our 3OV algorithm. Constructing the instance (T, m, ℓ, d) takes $O(nW)$ time, and the complexity of T is $O(nW)$. Our SC algorithm for the input (T, m, ℓ, d) takes $O(n^{3-\varepsilon}W^{3-\varepsilon})$ time. Therefore, the overall running time is $O(n^{3-\varepsilon}W^{O(1)})$, which cannot occur unless SETH fails. □

Chapter 3

Map matching queries on realistic input graphs under the Fréchet distance

3.1 Introduction

Location-aware devices have enabled the tracking of vehicle trajectories. In urban environments, vehicle trajectories align with an underlying road network. However, imprecision in the Global Positioning System introduces errors into the trajectory data. Map matching aims to mitigate the effects of these errors by computing a path on the underlying road network that best represents the vehicle's trajectory. See Figure 3.1.

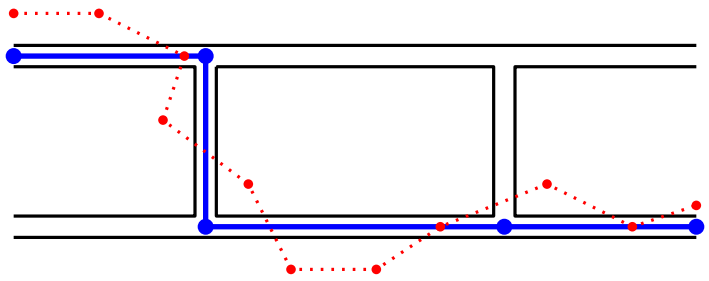


Figure 3.1: A road network (black), a noisy trajectory (red), and its matched path (blue).

Map matching is a common preprocessing step for analysing vehicle trajectories. As such, numerous map matching algorithms have been proposed across multiple communities (e.g. in the Urban Planning, Geographic Information Systems, and Databases communities). Map matching was the focus of the 2012 ACM SIGSPATIAL Cup [10]. For an overview of the extensive literature on map matching, see the surveys [52, 111, 118, 137, 162, 169]. In the theory community, by far the most popular approach is to embed the road network and the trajectory into the Euclidean plane, and to compute a path on the road network that is the most spatially similar to the trajectory [11, 25, 49, 53, 103, 145, 163], where spatial similarity is measured using the Fréchet distance [12]. Formally, the map matching problem

under the Fréchet distance is defined as follows.

Problem 1 (Map matching)**.** *Given a graph P and a trajectory Q in the plane, compute a path π in P that minimises $d_F(\pi, Q)$, where $d_F(\cdot, \cdot)$ denotes the Fréchet distance.*

In a seminal paper by Alt, Efrat, Rote and Wenk [11], the authors study Problem 1 on geometric planar graphs. They provide an $O(pq \log p)$ time algorithm, where p is the complexity of the graph and q is the complexity of the trajectory. Their idea is to construct a free space surface, which is a generalisation of the free space diagram [12], and then to perform a sweep line algorithm where a set of reachable points is maintained at the sweep line's current position.

Alt et al. [11]'s algorithm forms the basis of several existing implementations [25, 53, 145, 161, 163]. Brakatsoulas et al. [25] implement Alt et al. [11]'s algorithm and experimentally compare it to a linear-time heuristic and an algorithm minimising the weak Fréchet distance. In their experiments, forty-five vehicle trajectories, each with approximately one hundred edges, are mapped onto an underlying road network with approximately ten thousand edges. Their experiments conclude that out of the three algorithms, Alt et al. [11]'s provides the best map matching results but is the slowest. Subsequent papers focus on improving the practical running time of the algorithm [145, 163]. We show that a significantly faster algorithm for geometric planar map matching is unlikely to exist, unless SETH fails.

Traditional analysis focuses on worst case instances, which are unlikely to occur in practice. By making realistic input assumptions, we can circumvent these worst-case instances, and provide bounds that better reflect running time on realistic input. In computational movement analysis, the most popular realistic input assumption is c-packedness. A set of edges is c-packed if the total length of edges inside any ball is at most c times the radius of the ball. Given two c-packed trajectories of complexity n, one can $(1+\varepsilon)$-approximate their Fréchet distance in $O(c\varepsilon^{-1/2} \log(1/\varepsilon)n + cn \log n)$ time [31, 68], circumventing the $\Omega(n^{2-\delta})$ lower bound for all $\delta > 0$ implied by SETH [28, 42].

Chen, Driemel, Guibas, Nguyen and Wenk [53] study map matching on realistic input graphs and realistic input trajectories. They provide a $(1+\varepsilon)$-approximation algorithm that runs in $O((p+q)\log(p+q) + (\phi q + cp)\log pq \log(p+q) + (\phi\varepsilon^{-2}q + c\varepsilon^{-1}p)\log(pq))$ time, where the graph is ϕ-low-density and has complexity p, and the trajectory is c-packed and has complexity q. A graph is ϕ-low-density if, for any ball of radius r, the number of edges with length at least r that intersect the ball is at most ϕ. Note that a c-packed graph is $2c$-low-density [68]. Chen et al. [53] implement their algorithm to map trajectories, each with at most one hundred edges, onto an underlying road network with approximately one million edges. Their experiments show significant running time improvements compared to previous work. On large road networks, their map matching algorithm runs in under a second, whereas previous algorithms [25, 163] require several hours.

For both general graphs and realistic input graphs, a shortcoming of the existing map matching algorithms is that, every time a trajectory is matched, the entire road network needs to be reprocessed from scratch. This is reflected in the linear dependence on p in the running times of Alt et al. [11] and Chen et al. [53], where p is the complexity of the input graph. If we were to build a data structure for efficient map matching queries, then we would remove the need to reprocess the graph every time a new trajectory is mapped. Formally, the query version of the map matching problem is given below.

Problem 2 (Map matching queries)**.** *Given a graph P in the plane, construct a data structure so that given a query trajectory Q in the plane, the data structure returns $\min_\pi d_F(\pi, Q)$, where π ranges over all paths in P and $d_F(\cdot, \cdot)$ denotes the Fréchet distance.*

To the best of our knowledge, we are the first to study map matching queries under the Fréchet distance. Obtaining a data structure for Problem 2 with query time that is sublinear in the complexity of the graph is stated as an open problem in [53] and in [103].

3.1.1 Contributions

In this chapter, we investigate map matching queries under the Fréchet distance. An open problem proposed independently by [53] and [103] asks whether it is possible to preprocess a graph into a data structure so that map matching queries can be answered in sublinear time.

We provide a negative result in the case of geometric planar graphs. We show that, unless SETH fails, there is no data structure that can be constructed in polynomial preprocessing time, that answers map matching queries in $O((pq)^{1-\delta})$ query time for any $\delta > 0$, where p and q are the complexities of the graph and the query trajectory, respectively. Our negative result shows that preprocessing does not help for geometric planar map matching.

We provide the first positive result in the case of realistic input graphs. Our data structure has near-linear size in terms of p, and its query time is polylogarithmic in terms of p. We consider the following theorem to be the main result of our chapter.

Theorem 3. *Given a c-packed graph P of complexity p, one can construct a data structure of $O(p\log^2 p + c\varepsilon^{-4}\log(1/\varepsilon)p\log p)$ size, so that given a query trajectory Q of complexity q, the data structure returns in $O(q\log q\cdot(\log^4 p + c^4\varepsilon^{-8}\log^2 p))$ query time a $(1+\varepsilon)$-approximation of $\min_\pi d_F(\pi, Q)$ where π ranges over all paths in P and $d_F(\cdot,\cdot)$ denotes the Fréchet distance. The preprocessing time is $O(c^2\varepsilon^{-4}\log^2(1/\varepsilon)p^2\log^2 p)$.*

The most closely related results are [53] and [103]. We briefly compare the realistic input assumptions of these related works to our result. In Chen et al. [53], the graph is ϕ-low-density and the trajectory is c-packed. In Gudmundsson and Smid [103], the graph is a c-packed tree with long edges, and the trajectory has long edges. In our result, the graph is c-packed, but surprisingly, we require no input assumptions on the query trajectory.

3.1.2 Related work

The Fréchet distance is a popular similarity measure for trajectories. To compute the Fréchet distance between a pair of trajectories of complexity n, Alt and Godau [12] provide an $O(n^2\log n)$ time algorithm, which Buchin et al. [37] improve to an $O(n^2\sqrt{\log n}(\log\log n)^{3/2})$ algorithm. Conditioned on the Strong Exponential Time Hypothesis (SETH), for all $\delta > 0$, Bringmann [28] shows an $\Omega(n^{2-\delta})$ lower bound for computing the Fréchet distance in two or more dimensions. Buchin et al. [42] generalise the lower bound to one or more dimensions.

Variants of Problem 1 have been considered. Seybold [144] and Chambers et al. [49] consider finding the shortest map matching paths in geometric graphs. Chen et al. [55] study map matching under the weak Fréchet distance, whereas Wylie and Zhu [167] and Fu et al. [89] consider map matching under the discrete Fréchet distance. Wei et al. [161] and Chen et al. [55] combine the Fréchet distance approach with a Hidden Markov Model approach to obtain a hybrid algorithm.

A problem closely related to Problem 2 is to preprocess a trajectory for Fréchet distance queries. Driemel and Har-Peled [67] preprocess a trajectory Z of complexity n in $O(n\log^3 n)$ time and $O(n\log n)$ space, so that given a query trajectory Q with complexity k, and a query subtrajectory $Z[u, v]$ where u and v are points on Z, one can return in $O(k^2\log n\log(k\log n))$

time a constant factor approximation of the Fréchet distance between $Z[u, v]$ and Q. In this chapter we show that, even with polynomial time preprocessing time on the trajectory Z, one cannot hope to answer Fréchet distance queries in truly subquadratic time, unless SETH fails. As such, special cases have been considered. For $k = 2$, one can answer $(1 + \varepsilon)$-approximate [67] or exact [46, 63, 106] queries in polylogarithmic query time, by constructing a data structure of subquadratic size. Discrete Fréchet distance queries for small values of k have also been studied [71, 83, 86].

Gudmundsson and Smid [103] preprocess a c-packed tree for Fréchet distance queries. We regard this to be one of the most relevant results to our work. Given a c-packed tree T, and a positive real number Δ, the authors show how to construct a data structure of size $O(cn)$ in $O(n \log^2 n + cn \log n)$ preprocessing time, so that given a polygonal curve Q with k vertices, one can decide in $O(c^4 k \log^2 n)$ time whether there exists a path $\pi \in T$ so that $d_F(\pi, Q) \leq 3.001 \cdot \Delta$, or that $d_F(\pi, Q) > \Delta$ for all paths $\pi \in T$, where $d_F(\cdot, \cdot)$ denotes the Fréchet distance. The authors assume that the edges of T and Q have length $\Omega(\Delta)$. Since Δ is fixed at preprocessing time, it is unclear whether it is possible to minimise the Fréchet distance to solve Problem 2.

Related structures that have received considerable attention include range searching and approximate nearest neighbour searching under the Fréchet distance [4, 18, 29, 39, 62, 70, 74, 84, 97, 114].

3.2 Preliminaries

Let $P = (V, E)$ be an undirected graph embedded in the Euclidean plane $\mathbb{R}^2$. An edge $uv \in E$ is a segment between $u, v \in V$, with length equal to the Euclidean distance, i.e. $|uv| = d(u, v)$. Let $p = |V| + |E|$ be the complexity of the graph P. We assume that P is connected, otherwise, our map matching queries can be handled for each connected component independently. A path $\pi \in P$ is defined to be a sequence of vertices $u_1, \ldots, u_k \in V$ so that $u_i u_{i+1} \in E$ for all $1 \leq i < k$. In particular, for the purposes of this chapter we consider only paths π in P that start and end at vertices of P. Given a pair of vertices $u, v \in P$, the graph metric d_P is defined so that $d_P(u, v)$ equals the total length of the shortest path between u and v in the graph P. The graph P is c-packed if, for every ball B_r of radius r in the Euclidean plane, the total length of edges in E inside B_r is upper bounded by cr. Formally, $\sum_{e \in E} |e \cap B_r| \leq cr$.

A trajectory is a sequence of vertices in the Euclidean plane. Given vertices $a_1, \ldots, a_q$, the polygonal curve Q is a piecewise linear function $Q : [1, q] \to \mathbb{R}^2$ satisfying $Q(i) = a_i$ for all $1 \leq i \leq q$, and $Q(i+\mu) = (1-\mu)a_i + \mu a_{i+1}$ for all integers $1 \leq i \leq q-1$ and reals $0 \leq \mu \leq 1$. Let $\Gamma(q)$ be the space of all continuous non-decreasing functions for $[0, 1] \to [1, q]$. For a pair of polygonal curves Q_1 and Q_2 of complexities n_1 and n_2, we define the Fréchet distance between Q_1 and Q_2 to be $d_F(Q_1, Q_2) = \inf_{(\alpha_1, \alpha_2) \in \Gamma(n_1) \times \Gamma(n_2)} \max_{\mu \in [0,1]} d(Q_1(\alpha_1(\mu)), Q_2(\alpha_2(\mu)))$, where $d(\cdot, \cdot)$ denotes the Euclidean distance.

Let $0 < \varepsilon < 1$ be a constant that is fixed at preprocessing time.

3.3 Technical Overview

In Section 3.3.1, we give an overview of our data structure for map matching queries on c-packed graphs. In Section 3.3.2, we give an overview of our lower bound for map matching queries on geometric planar graphs. Full proofs are provided in Sections 3.4-3.7.

3.3.1 Data structure for c-packed graphs

Our data structure for c-packed graphs is built in three stages. In Stage 1, we construct a data structure for straightest path queries, which we will define in due course. In Stage 2, we construct a data structure for map matching queries, in the special case that the trajectory is a segment. In Stage 3, we construct a data structure for map matching queries in general. Each stage builds upon and generalises the previous stage. We provide an overview of Stages 1, 2 and 3 in Sections 3.3.1, 3.3.1 and 3.3.1 respectively.

Stage 1: Straightest path queries

For every pair of vertices in the graph, we are interested in precomputing a path between them that is as straight as possible. We define straightness using the Fréchet distance. Formally, given a pair of vertices u and v, we define a straightest path between u and v to be a path $\pi \in P$ between u and v that minimises the Fréchet distance $d_F(\pi, uv)$. This leads us to the following definition for straightest path queries.

Subproblem 4 (Straightest path queries)**.** *Given a graph P in the plane, construct a data structure so that given any pair of vertices $u, v \in P$, the data structure returns $\min_\pi d_F(\pi, uv)$ where π ranges over all paths in P between u and v. See Figure 3.2.*

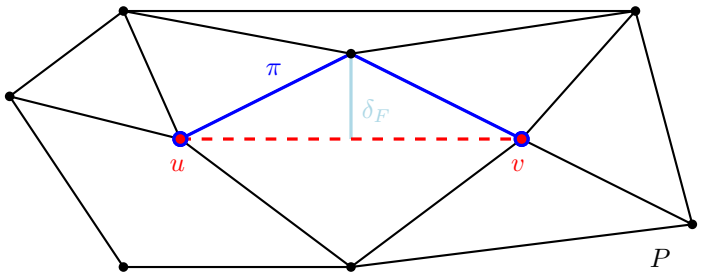


Figure 3.2: Given a pair of query vertices u, v (red), the data structure in Subproblem 4 returns $\min_\pi d_F(\pi, uv)$ (light-blue) where π ranges over all paths between u and v (blue) in the graph P (black).

Note that Subproblem 4 can be viewed as a special case of map matching queries, where the query trajectory must be a segment between two graph vertices, and the path must connect the endpoints of the query segment. A naïve way to answer straightest path queries is, for every pair of vertices, to precompute the Fréchet distance for its straightest path. Unfortunately, storing the precomputed Fréchet distance for all pairs of vertices requires $\Omega(p^2)$ space.

Instead, we use a semi-separated pair decomposition [1] to reduce the number of pairs we need to consider. We define the transit vertices of a semi-separated pair to be a set of vertices so that any path between the two components of the semi-separated pair must pass through at least one of the transit vertices. We define the set of transit pairs of a semi-separated pair to be pairs of vertices where one vertex is in the semi-separated pair, and one vertex is a transit vertex. For c-packed graphs, we show that there are at most $O(cp \log p)$ transit pairs. By storing the minimum Fréchet distance for each transit pair, we reduce the storage requirement of our data structure to $O(cp \log p)$.

Finally, we answer straightest path queries by dividing the path $u \to v$ into two paths. Specifically, we divide $u \to v$ into $u \to w \to v$, where w is a transit vertex of the semi-separated pair separating u and v. Having precomputed the minimum Fréchet distance for transit pairs (u, w) and (w, v), we use these Fréchet distances to obtain a constant factor approximation for the Fréchet distance of the straightest path between u and v.

Putting this all together, we obtain Theorem 5. For a full proof see Section 3.4.

Theorem 5. *Given a c-packed graph P of complexity p, one can construct a data structure of $O(cp \log p)$ size, so that given a pair of query vertices $u, v \in P$, the data structure returns in $O(\log p)$ query time a 3-approximation of $\min_\pi d_F(\pi, uv)$, where π ranges over all paths in P between u and v. The preprocessing time is $O(cp^2 \log^2 p)$.*

Stage 2: Map matching segment queries

Our next step is to answer map matching queries where the query trajectory is an arbitrary segment.

Subproblem 6 (Map matching segment queries)**.** *Given a graph P in the plane, construct a data structure so that given a query segment Q in the plane, the data structure returns $min_\pi d_F(\pi, Q)$ where π ranges over all paths in P that start and end at a vertex of P. See Figure 3.3.*

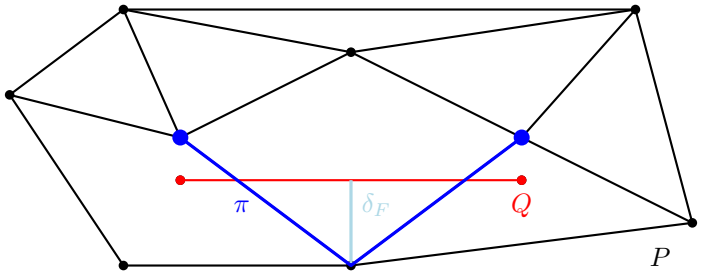


Figure 3.3: Given a query segment Q (red), the data structure in Subproblem 6 returns $\min_\pi d_F(\pi, Q)$ (light-blue) where π ranges over all paths (blue) in the graph P (black).

Subproblem 6 can be viewed as a generalisation of Subproblem 4, where the endpoints of the segment Q are not necessarily graph vertices, and the starting and ending points of the path are not given and must instead be computed. To answer map matching segment queries, we combine two data structures. The first data structure is an extension of the data structure in Theorem 5, which we modify to handle query segments that do not necessarily have their endpoints at graph vertices. The second data structure is to build a simplification of the c-packed graph so that one can efficiently query the starting and ending points of the path.

To build our first data structure, we use the result of Driemel and Har-Peled [67], which states that one can preprocess a trajectory in near-linear time and space, so that given a query segment, one can $(1 + \varepsilon)$-approximate the Fréchet distance from the query segment to any subcurve of the trajectory in constant time. Let $\varepsilon > 0$ and $\chi = \varepsilon^{-2} \log(1/\varepsilon)$. By combining Theorem 5 with their result, we obtain a data structure of $O(c\chi^2 p \log p)$ size, so that given a pair of query vertices $u, v \in P$ and a query segment ab in the plane, the data

structure returns in $O(\log p + c\varepsilon^{-1})$ time a $(1+\varepsilon)$-approximation of $\min_\pi d_F(\pi, ab)$, where π ranges over all paths in P between u and v. The preprocessing time is $O(c\chi^2 p^2 \log^2 p)$.

To build our second data structure, we use graph clustering to simplify the c-packed graph. We first consider the decision version of the problem. Given a Fréchet distance r, we guarantee that all edges in our simplified graph have length at least εr. By c-packedness, the number of simplified graph vertices inside a disk of radius r is at most a constant. Given a query segment ab, this reduces the number of candidate starting and ending points of the matched path to a constant. We use an orthogonal range searching data structure to efficiently query for the candidate starting and ending points of the path. Finally, we apply parametric search to minimise the Fréchet distance r.

By combining our two data structures, we obtain Theorem 7. For a full proof see Section 3.5.

Theorem 7. *Given a c-packed graph P of complexity p, one can construct a data structure of $O(c\varepsilon^{-4}\log^2(1/\varepsilon)\cdot p\log p)$ size, so that given a query segment ab in the plane, the data structure returns in $O(c^4\varepsilon^{-4}\cdot\log^2 p)$ time a $(1+\varepsilon)$-approximation of $\min_\pi d_F(\pi, ab)$, where π ranges over all paths in P that start and end at a vertex of P. The preprocessing time is $O(c\varepsilon^{-4}\log^2(1/\varepsilon)\cdot p^2\log^2 p)$.*

Stage 3: Map matching queries

Finally, we consider general map matching queries, which we restate for convenience. See Figure 3.4.

Problem 2 (Map matching queries)**.** *Given a graph P in the plane, construct a data structure so that given a query trajectory Q in the plane, the data structure returns $\min_\pi d_F(\pi, Q)$, where π ranges over all paths in P and $d_F(\cdot,\cdot)$ denotes the Fréchet distance.*

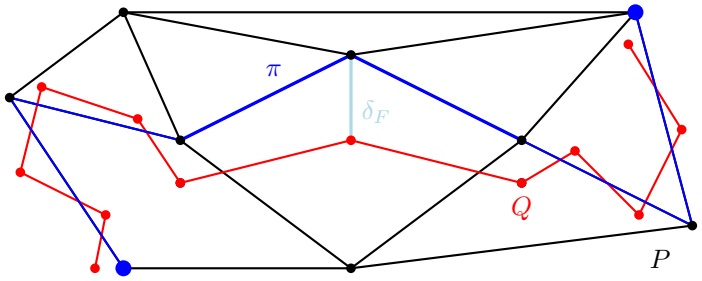


Figure 3.4: Given a query trajectory Q (red), the data structure in Problem 2 returns $\min_\pi d_F(\pi, Q)$ (light-blue) where π ranges over all paths (blue) in the graph P (black).

Let the vertices of Q be $a_1, \ldots, a_q$, and let the Fréchet distance for the decision version be r. We compute a set of points T_i that a_i can match to. Each point $b_{i,j} \in T_i$ is either a vertex of P, or a point along an edge of P. The size of T_i is at most a constant, depending on c and ε. For a point $b_{i,j}$ that is a vertex of P, we construct it in the same way as in Stage 2. For a point $b_{i,j}$ that is on an edge of P, we construct it by sampling along the edges of P that have length at least $\varepsilon r/2$ and are within a distance of r to a_i. To efficiently query the edges of P with these two properties, we build a three-dimensional low-density environment and use the range searching data structure of Schwarzkopf and Vleugels [141].

The final step is to build a directed graph on the set of points $\cup_{i=1}^{q} T_i$. For each $b_{i,j} \in T_i$ and $b_{i+1,k} \in T_{i+1}$, we use the map matching segment query from the previous section to compute a $(1+\varepsilon)$-approximation of the minimum Fréchet distance $\min_\pi(\pi, a_i a_{i+1})$ where π ranges over all paths between $b_{i,j}$ and $b_{i+1,k}$. We set the capacity of the directed edge from $b_{i,j}$ to $b_{i+1,k}$ to be this minimum Fréchet distance. We decide whether there is a directed path from a point in T_1 to a point in T_q so that the capacities of all edges on the directed path are at most $(1+\varepsilon)r$. Finally, we use parametric search to minimise r.

Putting this all together, we obtain Theorem 3, which we restate for convenience. For a full proof see Section 3.6.

Theorem 3. *Given a c-packed graph P of complexity p, one can construct a data structure of $O(p\log^2 p + c\varepsilon^{-4}\log(1/\varepsilon)p\log p)$ size, so that given a query trajectory Q of complexity q, the data structure returns in $O(q\log q\cdot(\log^4 p + c^4\varepsilon^{-8}\log^2 p))$ query time a $(1+\varepsilon)$-approximation of $\min_\pi d_F(\pi, Q)$ where π ranges over all paths in P and $d_F(\cdot,\cdot)$ denotes the Fréchet distance. The preprocessing time is $O(c^2\varepsilon^{-4}\log^2(1/\varepsilon)p^2\log^2 p)$.*

3.3.2 Lower bound for geometric planar graphs

In the final section, we investigate lower bounds for map matching queries on graphs that are not c-packed. Our lower bounds attempt to explain why answering map matching queries is such a difficult problem in general. In particular, we show that unless SETH fails, there is no data structure that can be constructed in polynomial preprocessing time, that answers map matching queries on geometric planar graphs in truly subquadratic time. Note that the upper bound of Alt et al.'s [11] matches this lower bound up to lower-order factors, which implies that preprocessing does not help for geometric planar map matching, unless SETH fails.

To build towards our lower bound for map matching queries, we consider a warm-up problem, which is to preprocesses a trajectory, so that given a query trajectory, the data structure can efficiently answer the Fréchet distance between the query trajectory and the preprocessed trajectory. Buchin et al. [46] claim that this is an extremely difficult problem, which is why the special case of query segments is considered in their chapter. We provide evidence towards Buchin et al.'s [46] claim. We show that preprocessing does not help with Fréchet distance queries on trajectories unless SETH fails. In particular, there is no data structure with polynomial preprocessing time that can answer Fréchet distance queries significantly faster than computing the Fréchet distance without processing. To show our lower bound, we modify the Bringmann's [28] construction to answer the offline version of the data structure problem in a similar fashion to Bringmann et al. [29] and Rubinstein [139].

Next, we prove a lower bound for Problem 1. We show that unless SETH fails, there is no truly subquadratic time for map matching on geometric planar graphs. This shows that the algorithm by Alt et al. [11] for geometric planar map matching is optimal up to lower-order factors, unless SETH fails. Finally, we combine the ideas from our warm-up problem and our lower bound for Problem 1 to rule out truly subquadratic query times for map matching queries on geometric planar graphs, unless SETH fails.

Putting this all together, we obtain Theorem 8. For a full proof see Section 3.7.

Theorem 8. *Given a geometric planar graph of complexity p, there is no data structure that can be constructed in $\mathrm{poly}(p)$ time, that when given a query trajectory of complexity q, can answer 2.999-approximate map matching queries in $O((pq)^{1-\delta})$ query time for any $\delta > 0$, unless SETH fails. This holds for any polynomial restrictions of p and q.*

This completes the overview of the main results of our chapter.

3.4 Stage 1: Straightest path queries

The first stage of our data structure for c-packed graphs is to construct a straightest path query data structure. Recall that the straightest path between u and v is a path $\pi \in P$ from u to v that minimises the Fréchet distance $d_F(\pi, uv)$. A data structure for straightest path queries is defined as follows. Given a pair of query vertices u and v, the data structure returns the minimum Fréchet distance $d_F(\pi, uv)$ where π ranges over all paths between u and v. As stated in the technical overview, we avoid storing a quadratic number of Fréchet distances by using a semi-separated pair decomposition (SSPD) to reduce the number of pairs of vertices we need to consider.

Definition 9 (SSPD)**.** *Let V be a set of vertices. A semi-separated pair decomposition of V with separation constant $s \in \mathbb{R}^+$ is a collection $\{(A_i, B_i)\}_{i=1}^k$ of pairs of non-empty subsets of V so that*

$$\min(\text{diameter}(A_i), \text{diameter}(B_i)) \leq s \cdot d(A_i, B_i),$$

and for any two distinct points u and v of V, there is exactly one pair (A_i, B_i) in the collection, such that (i) $u \in A_i$ and $v \in B_i$, or (ii) $v \in A_i$ and $u \in B_i$.

Note that for sets A, B, we define $d(A, B) = \min_{(a,b)\in A\times B} d(a, b)$, where $d(a, b)$ denotes the Euclidean distance. The total weight of $\{(A_i, B_i)\}_{i=1}^k$ is defined as $\sum_{i=1}^k (|A_i| + |B_i|)$. Abam et al. [1] show how to construct an SSPD of V with separation constant s in $O(ns^{-2} + n \log n)$ time, that has $O(ns^{-2})$ pairs, and total weight $O(ns^{-2} \log n)$, where n is number of vertices in V.

Although not explicitly stated in [1], given any two distinct points u and v of V, one can query the SSPD in $O(\log n)$ time to retrieve the pair (A_i, B_i) satisfying either (i) $u \in A_i$ and $v \in B_i$ or (ii) $v \in A_i$ and $u \in B_i$. We provide a sketch of the query procedure. The SSPD in [1] is constructed using a BAR tree [73], where each node in the balanced tree has an associated weight class. Each leaf of the BAR tree is associated with a point in V, and has a weight class of $O(\log n)$. Given any two distinct points u and v, we simultaneously traverse the BAR tree, from the root to the leaf nodes associated with u and v. The invariant maintained by the simultaneous traversal is that the weight class along the two traversals remains the same. The semi-separated pair (A_i, B_i) that we return is the pair of nodes in the BAR tree with minimum weight class that satisfies the semi-separated property $\min(\text{diameter}(A_i), \text{diameter}(B_i) \leq s \cdot d(A_i, B_i)$. Putting this together, we obtain the following observation.

Observation 10. *Given a pair of distinct points u and v of V, one can query the SSPD of [1] in $O(\log n)$ time to obtain a semi-separated pair (A_i, B_i) satisfying either (i) $u \in A_i$ and $v \in B_i$ or (ii) $v \in A_i$ and $u \in B_i$.*

We construct an SSPD of the vertices of P with separation constant $1/2$. For each semi-separated pair (A_i, B_i) in our SSPD, we select a set of $O(c)$ vertices of P to be *transit vertices*. The transit vertices have the property that any path from A_i to B_i must pass through a transit vertex. In Lemma 11, we show how to compute transit vertices.

Lemma 11. *Let $P = (V, E)$ and let $\{(A_i, B_i)\}_{i=1}^k$ be an SSPD of V with separation constant $1/2$. For each pair (A_i, B_i) of the SSPD, one can compute a set of vertices $C_i \subset V$ in $O(cp)$*

time satisfying (i) $|C_i| \leq 2c$*, and* (ii) *any path starting at a vertex in* A_i *and ending at a vertex in* B_i *must pass through a vertex in* C_i*.*

Proof. First, we construct the set C_i. Then we prove that C_i satisfies the required properties. Finally, we analyse the running time of our algorithm.

Without loss of generality, suppose diameter$(A_i) \leq$ diameter(B_i). Let a_0 be a vertex in A_i. Let D_1 be a disk with centre at a_0 with radius diameter(A_i), and let D_2 be a disk with centre at a_0 with radius $2 \cdot$ diameter(A_i). See Figure 3.5. All vertices of A_i are in D_1. All vertices of B_i are outside D_2, since $d(a_0, B_i) \geq d(A_i, B_i) \geq 2 \cdot$ diameter$(A_i) =$ radius(D_2), where the second inequality comes from the separation constant of the SSPD being $1/2$.

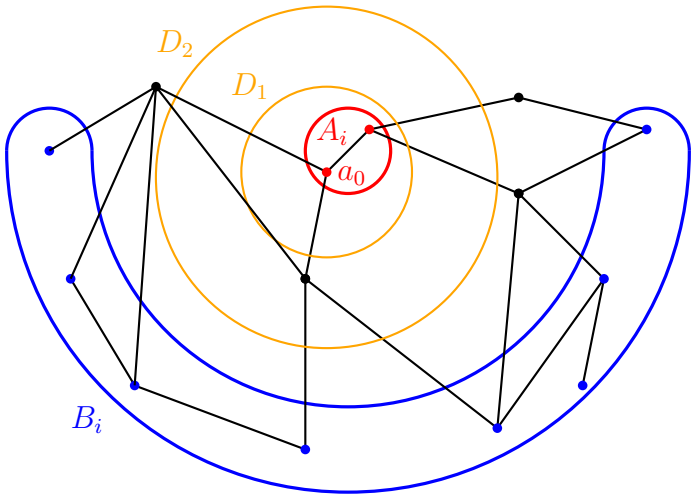


Figure 3.5: A semi-separated pair A_i (red) and B_i (blue). The circles D_1 and D_2 (orange) are centred at a vertex $a_0 \in A_i$, and have radius diameter(A_i) and 2 diameter(A_i) respectively. The value of the max-flow/min-cut in the figure is $\ell = 4$, so $|C_i| = 4$.

Next, we set up a max-flow instance. Set the capacity of each edge of $P = (V, E)$ to 1. Set the vertices in A_i to be sources, and set the vertices in B_i to be sinks. The max-flow of the instance is equal to its min-cut. Let the minimum cut be a set of edges $e_1, e_2, \ldots, e_\ell$. Choose one endpoint for each edge $e_1, e_2, \ldots, e_\ell$ to form the set C_i. This completes the construction of C_i.

We show that our construction of C_i satisfies the properties (i) $|C_i| \leq 2c$, and (ii) any path starting at a vertex in A_i and ending at a vertex in B_i must pass through a vertex in C_i. Property (ii) follows from $e_1, e_2, \ldots, e_\ell$ being a cut. This is because removing all the edges in the cut would disconnect the sources from the sinks, so all paths from A_i to B_i must pass through one of $e_1, e_2, \ldots, e_\ell$ and one of the vertices in C_i. Property (i) follows from c-packedness. In the max-flow instance, the capacity of the max-flow is ℓ. Since all edges have capacity 1, there are ℓ edge-disjoint paths from A_i to B_i. Each edge-disjoint path has one endpoint in D_1, and one endpoint outside D_2. So each path intersects both the inner and outer boundaries of the annulus $D_2 \setminus D_1$. The width of the annulus $D_2 \setminus D_1$ is equal to diameter(A_i). Therefore, there are ℓ edge disjoint paths in $D_2 \setminus D_1$ that each have length at least diameter(A_i). Since the graph is c-packed, the total length of edges in the ball D_2 is at most c times the radius of D_2, which is $2c \cdot$ diameter(A_i). Therefore, $\ell \cdot$ diameter$(A_i) \leq 2c \cdot$ diameter(A_i). Hence, $|C_i| = \ell \leq 2c$ as required.

Finally, we analyse the running time of our algorithm, which is dominated by computing the max-flow. The running time of the Ford-Fulkerson algorithm is equal to the number of edges in P times the max-flow. Since $\ell \leq 2c$, the max-flow is $\leq 2c$. Moreover, there are $O(p)$ edges in P. Therefore, the overall running time of the algorithm is $O(cp)$. □

The set of transit vertices for a semi-separated pair (A_i, B_i) is defined to be the set C_i constructed above. Next, we define transit pairs. Given a semi-separated pair (A_i, B_i), a transit pair for the semi-separated pair (A_i, B_i) is a pair of vertices (u, w) so that $u \in A_i \cup B_i$ and $w \in C_i$, where C_i is the set of transit vertices for (A_i, B_i) defined in Lemma 11. Now, we bound the total number of transit vertices and pairs.

Lemma 12. *There are $O(cp)$ transit vertices and $O(cp \log p)$ transit pairs in P, over all semi-separated pairs in the SSPD.*

Proof. There are $O(p)$ semi-separated pairs in the SSPD in [1]. By Lemma 11, there are $O(c)$ transit vertices per semi-separated pair. Therefore, there are $O(cp)$ transit vertices in total. For a semi-separated pair (A_i, B_i), let (u, w) be a transit pair. There are $|A_i| + |B_i|$ choices for u, and at most $2c$ choices for w. Therefore, the number of transit pairs over all semi-separated pairs is at most $\sum_{i=1}^{k} 2c(|A_i| + |B_i|) = O(cp \log p)$, since $\sum_{i=1}^{k}(|A_i| + |B_i|) = O(p \log p)$ is the weight of the SSPD in [1]. □

Our next step is to precompute and store the minimum Fréchet distance $d_F(\pi, uw)$ for each transit pair (u, w), where π ranges over all paths in P between u and w. For this, we use a modification of the algorithm by Alt et al. [11].

Lemma 13. *Let $u, w \in P$ be a pair of vertices, and let ab be a segment. One can compute $\min_\pi d_F(\pi, ab)$ in $O(p \log p)$ time, where π ranges over all paths in P between u and w.*

Proof (Sketch). Our proof is essentially the same as in [11], except that we replace the sweepline algorithm with a simple Dijkstra search [65]. The fact that the endpoints u and w are given makes this simplification possible. Furthermore, by replacing the sweepline with Dijkstra, we do not require P to be planar.

For the sake of completeness, we provide a proof sketch of our result. We set up a free space diagram for the decision problem in the same way as in [11]. Let $P = (V, E)$. For each edge $e \in E$, let $FD(e, ab)$ be the free space diagram between e and ab. Note that the x- and y-coordinates of $FD(e, ab)$ denote the positions along e and ab respectively. Moreover, orient $FD(e, ab)$ so that a has the minimum y-coordinate and b has the maximum y-coordinate. Similarly to [11], there is a path π between vertices u and w satisfying $d_F(\pi, ab) \leq d$ if and only if there is a sequence of free space diagrams $\{FD(e_i, ab)\}_{i=1}^{k}$ with a monotone path in the free space from (u, a) to (w, b). See Figure 3.6.

We avoid using a sweepline algorithm, and instead perform a Dijkstra search between u and w. First, check that (u, a) and (w, b) are in the free space. Next, construct a priority queue on points (x, y) in the free space diagram, where x is a vertex of P, and y is any point on the segment ab. The priority of (x, y) is y. The invariant maintained by the priority queue is that for all (x, y) in the priority queue, there is a monotone path from (u, a) to (x, y). Initially, the priority queue contains only (u, a). In each iteration, we pop a point from the priority queue with minimum y-coordinate. Let this point be (x, y). For all neighbours x' of $x \in P$, add (x', y') to the priority queue, where y' is the minimum y-coordinate so that there is a monotone path from (x, y) to (x', y') in $FD(xx', ab)$. This maintains the invariant of the priority queue. Halt the priority queue if (w, b') is in the priority queue, which occurs

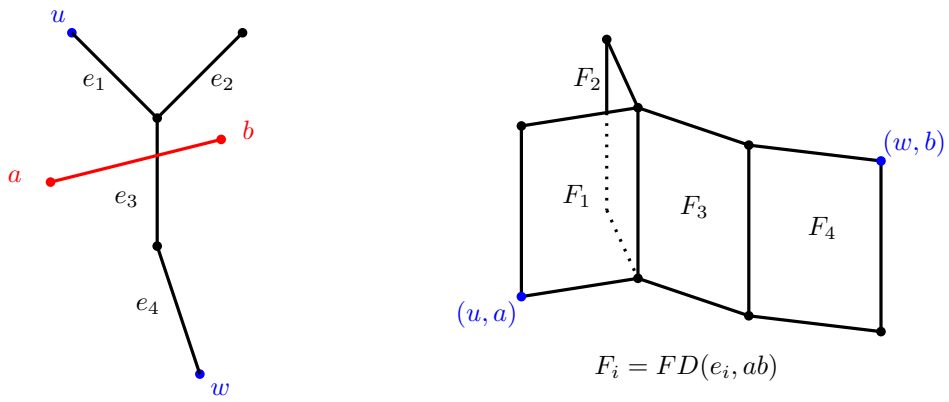


Figure 3.6: The x-coordinates of $FD(e_i, ab)$ denote the position along e_i, and the y-coordinates denote the position along ab. There is a path π between u and w with $d_F(\pi, ab) \leq d$ if and only if there is a monotone path from (u, a) to (w, b) in the free space surface.

if and only if there is a monotone path from (u, a) to (w, b') to (w, b). Finally, we apply parametric search to minimise the Fréchet distance in the same way as in [11].

We analyse the running time. Constructing the free space diagrams takes $O(p)$ time. Running Dijkstra's algorithm takes $O(|E| + |V| \log |V|) = O(p \log p)$ time. Finally, applying parametric search [126] with Cole's optimisation [59] takes $O(p \log p)$ time. □

We are now ready to build the data structure for straightest path queries, which is the main result of this section.

Theorem 5. *Given a c-packed graph P of complexity p, one can construct a data structure of $O(cp \log p)$ size, so that given a pair of query vertices $u, v \in P$, the data structure returns in $O(\log p)$ query time a 3-approximation of $\min_\pi d_F(\pi, uv)$, where π ranges over all paths in P between u and v. The preprocessing time is $O(cp^2 \log^2 p)$.*

Proof. First we describe the preprocessing procedure. Construct an SSPD of the vertices of P, with separation constant $1/2$. For each semi-separated pair (A_i, B_i), let C_i be its set of transit vertices as defined in Lemma 11. Recall that if $u \in A_i \cup B_i$ and $w \in C_i$, then (u, w) is a transit pair of (A_i, B_i). For each transit pair (u, w), we set $ab = uw$ in Lemma 13 to compute the straightest path between u and w, and we store the minimum Fréchet distance.

Next, we describe the query procedure. Given a pair of query vertices u and v, we query our SSPD for the semi-separated pair (A_i, B_i) so (i) $u \in A_i$ and $v \in B_i$, or (ii) $v \in A_i$ and $u \in B_i$. Let C_i be the transit vertices for (A_i, B_i). For each $w \in C_i$, define π_{uw} to be the straightest path between u and w, and define $D_{uw} = d_F(\pi_{uw}, uw)$. Define π_{wv} and D_{wv} analogously. Define t to be the orthogonal projection of w onto uv and define D_w to be the orthogonal distance. See Figure 3.7. Finally, return $\min_{w \in C_i}(\max(D_{uw}, D_{wv}) + D_w)$ as a 3-approximation for the minimum Fréchet distance of the shortest path between u and v.

We prove that the query procedure returns a 3-approximation. Our proof is inspired by the proof of Lemma 5.5 in [67]. Let π_{uv} be the straightest path between u and v. Then, for any transit vertex $w \in C_i$, we have

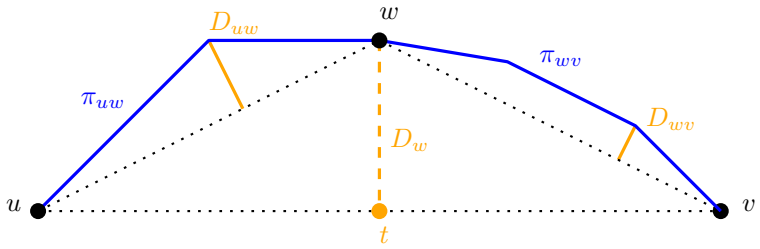


Figure 3.7: Vertices u, v and transit vertex w (black), the straightest paths π_{uw} and π_{wv} (blue) with Fréchet distances D_{uw}, D_{wv} (orange), and the orthogonal distance D_w from w to uv (orange, dashed).

$$\begin{aligned} d_F(\pi_{uv}, uv) &\leq d_F(\pi_{uw} \circ \pi_{wv}, uv) \\ &\leq \max(d_F(\pi_{uw}, ut), d_F(\pi_{wv}, tv))) \\ &\leq \max(d_F(\pi_{uw}, uw) + d_F(uw, ut), d_F(\pi_{wv}, wv) + d_F(wv, tv)) \\ &= \max(D_{uw} + D_w, D_{wv} + D_w) \\ &= \max(D_{uw}, D_{wv}) + D_w \end{aligned}$$

Therefore, $d_F(\pi_{uv}, uv) \leq \min_{w \in C_i}(\max(D_{uw}, D_{wv}) + D_w)$. Next, using Lemma 11, assume that $w^* \in C_i$ is a transit vertex so that $w^* \in \pi_{uv}$. Then clearly $D_{w^*} \leq d_F(\pi_{uv}, uv)$. By Lemma 5.3 in [67], $D_{uw^*} \leq 2 \cdot d_F(\pi_{uv}, uv)$. Putting this together, we obtain $\max(D_{uw^*}, D_{w^*v}) + D^*_w \leq 3 \cdot d_F(\pi_{uv}, uv)$. Therefore, our query procedure returns a 3-approximation of $d_F(\pi_{uv}, uv)$, as required.

Finally, we analyse the running time and space of our preprocessing and query procedures. Constructing the SSPD takes $O(p \log p)$ time [1]. By Lemma 11, all transit vertices and transit pairs can be computed in $O(cp^2 \log p)$ time. Computing the minimum Fréchet distance for all transit pairs takes $O(cp^2 \log^2 p)$ time. Therefore, our data structure can be constructed in $O(cp^2 \log^2 p)$ time. Storing the Fréchet distance for all transit pairs requires $O(cp \log p)$ space, by Lemma 12. By Observation 10, querying the SSPD for the semi-separated pair containing the query vertices takes $O(\log p)$ time. There are $O(c)$ transit vertices to check. For a transit vertex w, looking up the values D_{uw} and D_{wv} in our data structure takes constant time. Computing D_w takes constant time. Putting this all together, we obtain the stated theorem. □

3.5 Stage 2: Map matching segment queries

Recall that a data structure for map matching segment queries is defined as follows. Given a query segment ab in the plane, the data structure returns the minimum Fréchet distance $d_F(\pi, ab)$ as π ranges over all paths in P that start and end at a vertex of P.

As stated in the technical overview, we build two data structures in this section. The first data structure in this section is an extension of Theorem 5, which we modify to handle arbitrary query segments in the plane.

Lemma 14. *Given a c-packed graph P of complexity p, one can construct a data structure of $O(cp \log p)$ size, so that given a pair of query vertices $u, v \in P$ and a query segment ab in the plane, the data structure returns in $O(\log p)$ query time a 3-approximation of*

$\min_\pi d_F(\pi, ab)$, *where* π *ranges over all paths in* P *between* u *and* v. *The preprocessing time is* $O(cp^2 \log^2 p)$.

Proof (Sketch). Our proof is the exactly same as the proof of Theorem 5, except that (i) we define $D_w = d_F(uw \circ wv, ab)$, and (ii) we define t to be the point on ab that matches to w, under the minimum Fréchet distance matching between ab and $uw \circ wv$. □

This completes the construction of the first data structure, however, its approximation ratio is 3. Next, we improve the approximation ratio to $(1 + \varepsilon)$. We first prove a useful lemma.

Lemma 15. *Let* $u, w \in P$ *be a fixed pair of vertices. Let* $\varepsilon > 0$ *and* $\chi = \varepsilon^{-2} \log(1/\varepsilon)$. *One can construct a data structure of* $O(\chi^2)$ *space, so that given a query segment* ab *in the plane, the data structure returns in constant time a* $(1+\varepsilon)$*-approximation of* $\min_\pi d_F(\pi, ab)$, *where* π *ranges over all paths in* P *between* u *and* w. *The preprocessing time is* $O(\chi^2 p \log p)$.

Proof (Sketch). Our proof is the same as the proof of Lemma 5.8 in [67], except that instead of computing the Fréchet distance between a curve and a segment joining a pair of grid points, we use Lemma 13 to minimise the Fréchet distance over all paths between u and w. □

Now, we use Lemma 15 to improve the approximation ratio of Lemma 14 to $(1 + \varepsilon)$.

Lemma 16. *Let* $\varepsilon > 0$ *and* $\chi = \varepsilon^{-2} \log(1/\varepsilon)$. *One can construct a data structure of* $O(c\chi^2 p \log p)$ *size, so that given a pair of query vertices* $u, v \in P$ *and a query segment* ab *in the plane, the data structure returns in* $O(\log p + c\varepsilon^{-1})$ *time a* $(1 + \varepsilon)$*-approximation of* $\min_\pi d_F(\pi, ab)$, *where* π *ranges over all paths in* P *between* u *and* v. *The preprocessing time is* $O(c\chi^2 p^2 \log^2 p)$.

Proof (Sketch). Our proof is essentially the same as the proof of Theorem 5.9 in [67], except that (i) we replace subcurves with transit pairs, (ii) we replace Lemma 5.8 in [67] with Lemma 15, and (iii) we replace Theorem 5.6 in [67] with Lemma 14.

For the sake of completeness, we provide a proof sketch of our result. We first describe the preprocessing procedure. We construct the data structure in Lemma 14. For each transit pair, we construct the data structure in Lemma 15. Next, we describe the query procedure. Given a pair of query vertices (u, v) and a query segment ab, we use Lemma 14 to compute a real value r so that $\min_\pi d_F(\pi, ab) \leq r \leq 3 \cdot \min_\pi d_F(\pi, ab)$, where π ranges over all paths between u and v. Next, we iterate over all transit vertices w associated with the semi-separated pair containing (u, v). Define $B(w, 3r)$ to be a ball with radius $3r$ centred at w. If $B(w, 3r)$ does not intersect ab, we skip the transit vertex w and move onto the next one. Hence, we may assume that $B(w, 3r)$ intersects ab. We compute $O(\varepsilon^{-1})$ evenly spaced vertices on the chord $B(w, 3r) \cap ab$. Let t be one of these vertices. See Figure 3.8.

We use Lemma 15 to compute a $(1+\varepsilon)$-approximation of $\min_\pi d_F(\pi, at)$ as π ranges over all paths between u and w (resp. tb and paths between w and v). We take the larger Fréchet distance out of the at and tb cases, and return it as a $(1+\varepsilon)$-approximation of $\min_\pi d_F(\pi, ab)$ as π ranges over all paths between u and v assuming $w \in \pi$ and w matches to t. Finally, we minimise over all transit vertices w and the evenly spaced vertices $t \in B(w, 3r) \cap ab$ to obtain a $(1 + \varepsilon)$-approximation of $\min_\pi d_F(\pi, ab)$. The proof of correctness follows from Theorem 5.9 in [67].

We analyse the preprocessing time and space. There are $O(cp \log p)$ transit pairs by Lemma 12. By Lemma 14 and Lemma 15, the data structure has $O(c\chi^2 p \log p)$ size, and can

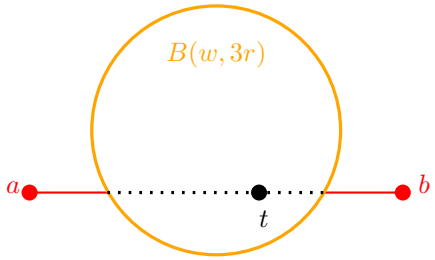


Figure 3.8: The $O(\varepsilon^{-1})$ evenly spaced vertices, including t (black) on the chord $B(w, 3r) \cap ab$ which is the intersection of segment ab (red) and the ball centred at w with radius $3r$ (orange).

be constructed in $O(c\chi^2 p^2 \log^2)$ preprocessing time. We analyse the query time. Computing a 3-approximation takes $O(\log p)$ query time. Iterating over all choices of w and t takes $O(c\varepsilon^{-1})$ time. Putting this together yields the claimed lemma. □

This improves the approximation ratio of the first data structure to $(1 + \varepsilon)$. Next, we consider the second data structure, which can efficiently query the starting and ending points of the path.

For our second data structure, we simplify the c-packed graph using graph clustering. The clustering algorithm we use is Gonzales' algorithm [94]. Let $P = (V, E)$ be the graph, which from Section 3.2 is assumed to be connected. For a pair of vertices $u, v \in V$, let $d_P(u, v)$ be the shortest path between u and v in P. For $k = 1, \ldots, p$, we compute a k-centre clustering of V under the graph metric d_P. For $k = 1$, choose an arbitrary vertex v_1 to be the 1-centre. Mark v_1 as a cluster centre, and let r_1 be the radius of the 1-centre clustering. For $k \geq 2$, compute the vertex v_k that is the furthest from all existing cluster centres $v_1, \ldots, v_{k-1}$. Mark v_k as a new centre, and let r_k be the radius of the k-centre clustering. After all vertices are marked as cluster centres, we have computed a list $[(v_1, r_1), \ldots, (v_p, r_p)]$ of cluster centres and cluster radii.

We use the cluster centres and cluster radii to construct a hierarchy of simplifications of the graph P. In particular, define V_r to be the set of vertices $\{v_i \in V : r_i \geq \varepsilon r\}$. We show that for any square S with side length $2r$, there are at most $O(c\varepsilon^{-1})$ vertices in $V_r \cap S$.

Lemma 17. *Let $P = (V, E)$ be a c-packed graph and let S be a square in the plane with side length $2r$. Then there exists a set of vertices $T \subseteq V$ satisfying (i) $|T| = O(c\varepsilon^{-1})$ and (ii) for all vertices $v \in V \cap S$, there exists $t \in T$ so that $d_P(v, t) \leq \varepsilon r$.*

Proof. Run the clustering algorithm described above to compute a list $[(v_1, r_1), \ldots, (v_p, r_p)]$ of cluster centres and their clustering radii. Recall that V_r is the set of vertices in V satisfying $r_i \geq \varepsilon r$. Let S' be a square concentric with S, but has side length $4r$. Define $T_1 = V_r \cap S'$. First, we show that T_1 satisfies Property (i). Then we add a single vertex to T_1 to construct T_2, and show that T_2 satisfies both Properties (i) and (ii).

For Property (i), if there exists a vertex $t \in T_1$ so that $d_P(t, t') \leq \varepsilon r$ for all $t' \in V$, then defining $T_1 = \{t\}$ clearly satisfies both properties. Otherwise, for all $t \in T_1$ there exists a vertex t' so that $d_P(t, t') \geq \varepsilon r$. Construct the shortest path from t to t' under the graph metric d_P, and let t'' be the point (not necessarily a vertex) on the shortest path between t and t' so that $d_P(t, t'') = \varepsilon r/3$. Let the shortest path from t to t'' be π_t. Construct the set of paths $\{\pi_t\}_{t \in T_1}$. First, we will show that the set of paths $\{\pi_t\}_{t \in T_1}$ are edge disjoint, and

all lie in a square with side length $5r$. See Figure 3.9. Then, we will use the c-packedness property in the square with side length $5r$ to prove Property (i).

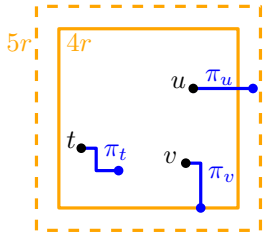


Figure 3.9: The vertices $T_1 = \{t, u, v\}$ (black) are in a square of side length $4r$ (orange, solid). The paths $\{\pi_t, \pi_u, \pi_v\}$ (blue) are edge disjoint and lie in a square with side length $5r$ (orange, dashed).

First, we show $\{\pi_t\}_{t \in T_1}$ is edge disjoint. Suppose for the sake of contradiction that $s, t \in T_1$, and π_s and π_t share an edge. Using this shared edge, by the triangle inequality we have $d_P(s,t) < \varepsilon r$. Let $s = v_i$ have cluster radius r_i, and $t = v_j$ have cluster radius r_j, so that $i < j$. Then we have the inequality

$$r_j \leq r_{j-1} = \max_{v \in P} \min_{k<j} d_P(v_k, v) = \min_{k<j} d_P(v_k, v_j) \leq d_P(v_i, v_j) = d_P(s,t) < \varepsilon r,$$

where $\max_{v \in P} \min_{k<j} d_P(v_k, v) = \min_{k<j} d_P(v_k, v_j)$ comes from the fact that v_j was the furthest vertex from all existing cluster centres in the j^{th} round of Gonzales' algorithm. But $t \in V_r$, so $r_j \geq \varepsilon r$. This is a contradiction, so $\{\pi_t\}_{t \in T_1}$ is edge disjoint.

Now, we will use the c-packedness property to show that $|T_1| = O(c\varepsilon^{-1})$. Each path π_t is a shortest path between a pair of points that are distance $\varepsilon r/3$ away, so the total path length of $\{\pi_t\}_{t \in T_1}$ is $|T_1| \cdot \varepsilon r/3$. Each vertex $t \in T_1$ is in a square with side length $4r$. Each path π_t has length at most $\varepsilon r < r$, since $0 < \varepsilon < 1$. Therefore, all edges in $\{\pi_t\}_{t \in T_1}$ are in a square with side length $5r$. Finally, by c-packedness, we have that $c \cdot 5r$ is an upper bound on the total edge length in the square of side length $5r$, which is guaranteed to contain the edges of $\{\pi_t\}_{t \in T_1}$. Therefore, $c \cdot 5r \geq |T_1| \cdot \varepsilon r/3$, which implies $|T_1| \leq 15c\varepsilon^{-1}$. This concludes the proof of Property (i).

For Property (ii), let $V_r = [v_1, \ldots, v_i]$. If V_r consists of all vertices of V, then T_1 consists of all vertices inside S', and (ii) is trivially true. Otherwise, suppose $v_{i+1} \notin V_r$. So $r_{i+1} < \varepsilon r$. Therefore, after $i+1$ rounds of Gonzales' algorithm, the cluster radius is at most εr. So all vertices $v \in P$ are within distance εr from one of the vertices $[v_1, \ldots, v_{i+1}]$. Define $T_2 = [v_1, \ldots, v_{i+1}] \cap S'$. Then $|T_2| \leq |T_1| + 1 = O(c\varepsilon^{-1})$. Moreover, for all vertices $v \in P \cap S$, v is within distance εr from one of the vertices $[v_1, \ldots, v_{i+1}]$. Without loss of generality, let $j \leq i+1$ so that $d_P(v, v_j) < \varepsilon r$. Since $v \in S$ and $d_P(v, v_j) < \varepsilon r$, we have $v_j \in S'$. Therefore, $v_j \in T_2 = [v_1, \ldots, v_{i+1}] \cap S'$. As a result, T_2 satisfies both Properties (i) and (ii), as required. □

Next, we build a data structure so that, given any square S in the plane, the data structure can efficiently return a set of vertices T that satisfies the properties in Lemma 17.

Lemma 18. *Let $P = (V, E)$, and let $\varepsilon > 0$. One can construct a data structure of $O(p \log p)$ size, so that given a query square S in the plane with side length $2r$, the data structure returns in $O(\log p + c\varepsilon^{-1})$ time a set of vertices T satisfying (i) $|T| = O(c\varepsilon^{-1})$ and (ii) for*

all vertices $v \in V \cap S$, there exists $t \in T$ so that $d_P(v,t) \leq \varepsilon r$. The preprocessing time is $O(p^2 \log p)$.

Proof. We show how to efficiently query the set T_2 given in Lemma 17. We run the clustering algorithm described in this section to compute a list $[(v_1, r_1), \ldots, (v_p, r_p)]$ of cluster centres and their clustering radii. We build an orthogonal range searching data structure for three-dimensional points. Specifically, for each pair (v_i, r_i) in the list, we insert the point (x_i, y_i, r_i) into the orthogonal range searching data structure, where (x_i, y_i) are the x- and y-coordinates of the point v_i, respectively. Given a square S, we perform an orthogonal range search for all vertices (v_i, r_i) so that $v_i \in S$, and $r_i \geq \varepsilon r$. We return this set of vertices as T_2. Lemma 17 proves that T_2 satisfies Properties (i) and (ii).

Next, we analyse the running time and space of the preprocessing and query procedures. We use the orthogonal range searching data structure of Afshani et al. [3], and we use the clustering algorithm of Gonzales [94].

The storage requirement of the orthogonal range searching data structure is $O(p \log^2 p)$. Computing the distance matrix under d_P takes $O(p^2 \log p)$ time. Performing Gonzales' clustering algorithm to compute p centres takes $O(p^2)$ time. Constructing the orthogonal range searching data structure takes $O(p \log^2 p)$ time. Therefore, the overall preprocessing time is $O(p^2 \log p)$. The query time of the orthogonal range searching data structure is $O(\log p + |T|)$. Therefore, the overall query time is $O(\log p + c\varepsilon^{-1})$. This proves the stated lemma. □

Finally, we are ready to combine the two data structures in Lemmas 16 and 18 to answer map matching segment queries. The theorem below is the main result of this section.

Theorem 7. *Given a c-packed graph P of complexity p, one can construct a data structure of $O(c\varepsilon^{-4} \log^2(1/\varepsilon) \cdot p \log p)$ size, so that given a query segment ab in the plane, the data structure returns in $O(c^4 \varepsilon^{-4} \cdot \log^2 p)$ time a $(1+\varepsilon)$-approximation of $\min_\pi d_F(\pi, ab)$, where π ranges over all paths in P that start and end at a vertex of P. The preprocessing time is $O(c\varepsilon^{-4} \log^2(1/\varepsilon) \cdot p^2 \log^2 p)$.*

Proof. The preprocessing procedure is to construct the data structure in Lemmas 16 and 18. For both data structures, we use the parameter $\varepsilon' = \varepsilon/6$ instead of ε.

Next, we consider the query procedure for the decision problem. Given a segment ab in the plane and a Fréchet distance of r, the decision problem is to decide whether $r^* \leq r$ or $r^* \geq r$, where $r^* = \min_\pi d_F(\pi, ab)$ where π ranges over all paths in P that start and end at a vertex of P. We construct a disk centred at a with radius r, and we enclose this disk in a square with side length $2r$. We query the data structure in Lemma 18 to obtain a set of vertices T_a. We query the set of vertices T_b analogously. For every $(u, v) \in T_a \times T_b$, we query the data structure in Lemma 16 for a value r_{uv} which is a $(1+\varepsilon')$-approximation of $\min_\pi d_F(\pi, ab)$, where π ranges over all paths between u and v. See Figure 3.10.

Let $r' = \min_{(u,v) \in T_a \times T_b} r_{uv}$. We distinguish three cases (a), (b) and (c):

(a) If $r' \leq r$, we return that $r^* \leq r$.
(b) If $r' \geq (1+\varepsilon')^2 r$, we return that $r^* \geq r$.
(c) If $r' \in [r, (1+\varepsilon')^2 r]$, we return that $r^* \in [(1-\varepsilon')r, (1+\varepsilon')^2 r]$.

The third case does not technically answer the decision problem, as it does not return $r^* \leq r$ or $r^* \geq r$. However, in this case, we will show that $(1+\varepsilon')r$ is a $(1+\varepsilon)$-approximation of r^*, as required by the stated theorem. This completes the description of the query procedure

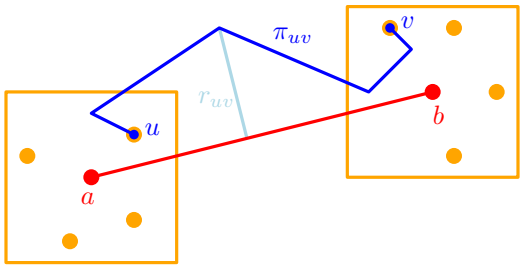


Figure 3.10: The query segment ab (red), the squares centred at a and b with side length $2r$ (orange), the sets of vertices T_a and T_b including u and v (orange), and the path π_{uv} (blue) between u and v minimising the Fréchet distance r_{uv} (light-blue) up to a factor of $(1+\varepsilon)$.

for the decision problem. Next, we prove its correctness, which we separate into cases (a), (b) and (c).

(a) We know $r^* \leq r_{uv}$ for all vertices u, v in P. Therefore, $r^* \leq r'$. So if $r' \leq r$, then $r^* \leq r$.

(b) Given a pair of vertices u, v in P, define r^*_{uv} to be $\min_\pi d_F(\pi, ab)$ where π ranges over all paths between u and v. Clearly, $r^* \leq r^*_{uv}$ for all vertices u, v in P. Moreover, by Lemma 16, since r_{uv} is a $(1+\varepsilon')$-approximation, we have $r^* \leq r^*_{uv} \leq r_{uv} \leq (1+\varepsilon')r^*_{uv}$. Let π^* be the path that attains r^*, i.e. $r^* = d_F(\pi^*, ab)$. Let the starting and ending points of π^* be u^* and v^* respectively. By Lemma 17, there exists a graph vertex $u \in T_a$ so that $d_P(u^*, u) \leq \varepsilon' r$. Define the vertex $v \in T_b$ analogously. See Figure 3.11. Consider the path π^*_{uv} obtained by concatenating the paths uu^*, π^*, and v^*v. Note that π^*_{uv} is a valid path between u and v, moreover, $d_F(\pi^*_{uv}, ab) \leq \max(d_F(uu^*, a), d_F(\pi^*, ab), d_F(v^*v, b))$. But $d_F(u^*, a) \leq d_F(\pi^*, ab) = r^*$, and $d_P(u^*, u) \leq \varepsilon' r$. Hence, $d_F(uu^*, a) \leq r^* + \varepsilon' r$. Similarly, $d_F(v^*v, b) \leq r^* + \varepsilon' r$, so $d_F(\pi^*_{uv}, ab) \leq r^* + \varepsilon' r$. Therefore, $r^*_{uv} \leq d_F(\pi^*_{uv}, ab) \leq r^* + \varepsilon' r$. Now, $r' \leq r_{uv} \leq (1+\varepsilon')r^*_{uv} \leq (1+\varepsilon')(r^* + \varepsilon' r)$. If $r' \geq (1+\varepsilon')^2 r$, then $(1+\varepsilon')(r^* + \varepsilon' r) \geq (1+\varepsilon')^2 r$, so $r^* + \varepsilon' r \geq r + \varepsilon' r$, and therefore $r^* \geq r$.

(c) From the proof of the first case, we have $r^* \leq r'$. If $r' \leq (1+\varepsilon')^2 r$, then $r^* \leq (1+\varepsilon')^2 r$. From the proof of the second case, we have $r' \leq r^* + \varepsilon' r$. If $r' \geq r$, then $r^* \geq (1-\varepsilon')r$. Putting this together, if $r' \in [r, (1+\varepsilon')^2 r]$, then $r^* \in [(1-\varepsilon')r, (1+\varepsilon')^2 r]$. In particular, $(1+\varepsilon')^2 r \geq r^*$, and $(1+\varepsilon')^2 r \leq (1+\varepsilon')^3 r^* \leq (1+\varepsilon')(1+3\varepsilon')r^* \leq (1+6\varepsilon')r^* = (1+\varepsilon)r^*$, so $(1+\varepsilon')r$ is a $(1+\varepsilon)$-approximation of r^*, as required.

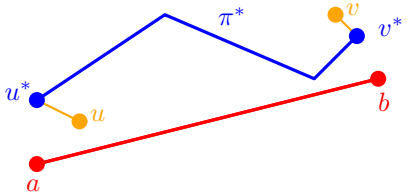


Figure 3.11: Given segment ab (red), the path π^* (blue) minimises its Fréchet distance to ab. We define a new path that concatenates vertex u, path π^*, and vertex v, where $u \in T_a$ and $v \in T_b$.

Next, we apply parametric search to the decision problem, which we call $D(r)$. Define $D(r)$ to be TRUE if $r' \leq r$, and define $D(r)$ to be FALSE if $r' \geq (1+\varepsilon')r$. If $r' \in [r, (1+\varepsilon')r]$, we immediately halt the parametric search, and return $(1+\varepsilon')r$ as a $(1+\varepsilon)$-approximation of r^*. It suffices to show (i) that $D(r)$ is monotone, and (ii) that all operations in $D(r)$ are either independent of r, or can be made equivalent to a constant number of comparisons $\{r > c_i\}$ where c_i is a critical value. First, we show (i). Suppose $D(r_1)$ evaluates to TRUE, and $r_1 < r_2$. Then $r^* \leq r_1 \leq r_2$, and we cannot have $D(r_2)$ evaluating to FALSE. So either $D(r_2)$ is also TRUE, or we halt the parametric search and obtain a $(1+\varepsilon)$-approximation. Similarly, if $D(r_1)$ evaluates to FALSE, and $r_1 > r_2$, then we cannot have $D(r_2)$ evaluating to TRUE. Therefore, $D(r)$ is monotone. Next, we show (ii). The first step of $D(r)$ is to query the data structure in Lemma 18 for the set T_a. The data structure is a three-dimensional orthogonal range searching data structure. All operations that depend on r can be evaluated by comparing r to a difference between x-, y- or z-coordinates. As an example, if $a = (x_a, y_a)$, then a point (x_i, y_i, r_i) lies in the orthogonal range if and only if $|x_a - x_i| \leq r$, $|y_a - y_i| \leq r$ and $r_i \geq \varepsilon' r$. In particular, $|x_a - x_i|$, $|y_a - y_i|$ and r_i/ε' are critical values. The next step is to query Lemma 16 to obtain r'. This step is independent of r and generates no critical values. Finally, we compare r' with r and $(1+\varepsilon')r$. Therefore, r' and $r'/(1+\varepsilon')$ are critical values. This completes the proof of (i) and (ii).

We analyse the construction time and space of the data structure. Let $\chi = \varepsilon^{-2}\log(1/\varepsilon)$. The data structure in Lemma 16 has $O(c\chi^2 p \log p)$ size, whereas the data structure in Lemma 18 has $O(p \log p)$ size. The data structure in Lemma 16 requires $O(c\chi^2 p^2 \log^2 p)$ preprocessing time, whereas the data structure in Lemma 18 requires $O(p^2 \log p)$ preprocessing time. Therefore, the overall data structure has $O(c\chi^2 p \log p)$ size, and requires $O(c\chi^2 p^2 \log^2 p)$ preprocessing time.

We analyse the query time of the decision problem. Querying the data structure in Lemma 18 takes $O(\log p + c\varepsilon^{-1})$ time. There are $O(c^2\varepsilon^{-2})$ pairs $(u, v) \in T_a \times T_b$. For each pair (u, v), querying the data structure in Lemma 16 takes $O(\log p)$ time. Therefore, the overall query time of the decision version is $O(c^2\varepsilon^{-2}\log p)$. We analyse the running time of parametric search. The decision version forms both the sequential and parallel algorithms. The running time of parametric search [126] is $O(N_p T_p + T_p T_s \log N_p)$, where T_s is the running time of the sequential algorithm, N_p is the number of processors for the parallel algorithm, and T_p is the number of parallel steps in the parallel algorithm. By setting $N_p = 1$, we obtain $T_s = T_p = O(c^2\varepsilon^{-2}\log p)$. Therefore, the overall running time of the parametric search step is $O(c^4\varepsilon^{-4}\log^2 p)$. □

3.6 Stage 3: Map matching queries

We start by considering the decision problem of the map matching query, in which we are given a trajectory Q in the plane and a Fréchet distance r, and we are to decide whether $r \leq r^*$ or $r \geq r^*$, where r^* is the minimum value of $d_F(\pi, Q)$ where π ranges over all paths in P that start and end at a vertex of P. Let Q have vertices $a_1, \ldots, a_q$. The first step is to compute a constant number of points on P that can match to a_i. We have two cases, either the point matching to a_i is a vertex of P, or it is a point along an edge of P. The points along the edges of P are defined as follows.

Definition 19. *Given a graph $P = (V, E)$ embedded in the Euclidean plane, define the set F to be points that lie on an edge of E. Formally, $F = \{f \in \mathbb{R}^2 : f \in e, e \in E\}$.*

The trajectory vertex a_i must match to a point in V or F. If the point is in V, we use Lemma 17 to compute a set of $O(c\varepsilon^{-1})$ vertices that can match to a_i. If the point is in F, we prove a generalisation of Lemma 17 to compute a set of $O(c\varepsilon^{-2})$ points that can match to a_i. The generalisation is stated below.

Lemma 20. *Let $P = (V, E)$ and let $F = \{f \in \mathbb{R}^2 : f \in e, e \in E\}$. Let S be a square in the plane with side length $2r$. Then there exists a set of points $T \subset F$ satisfying (i) $|T| = O(c\varepsilon^{-2})$ and (ii) for all points $f \in F \cap S$, there exists $t \in T$ so that $d_P(f, t) \leq \varepsilon r$.*

Proof. We use Lemma 17 to construct a set of graph vertices T_2 so that $|T_2| = O(c\varepsilon^{-1})$, and for all vertices $v \in V \cap S$, there exists $t_2 \in T_2$ so that $d_P(v, t) \leq \varepsilon r/2$. Let E_r be the set of edges E with length at least $\varepsilon r/2$. Let S' be a square that is concentric with S, but has side length $4r$. For each $e \in E_r$, choose $O(\varepsilon^{-1})$ evenly spaced points on the chord $e \cap S'$, so that the distance between consecutive points is at most $\varepsilon r/2$. Add these $O(\varepsilon^{-1})$ points to the set T_3, for each $e \in E_r$. We will show that the set $T_2 \cup T_3$ satisfies both Properties (i) and (ii).

We first prove Property (i). By Lemma 17, $|T_2| = O(c\varepsilon^{-1})$. Then for $e \in E_r$, the length of the edge $e \cap S'$ is at least $\varepsilon r/2$. By the c-packedness property on S', we have $c \cdot 4r \geq |E_r| \cdot \varepsilon r/2$. Therefore, $|E_r| = O(c\varepsilon^{-1})$. The set T_3 consists of $O(\varepsilon^{-1})$ points per edge in $|E_r|$, so $|T_3| = O(c\varepsilon^{-2})$. This completes the proof of Property (i).

Next we prove Property (ii). Let $f \in F \cap S$. We have three cases, either f is a graph vertex, f is on an edge with length $\leq \varepsilon r/2$, or f is on an edge with length $\geq \varepsilon r/2$. If f is a graph vertex, then Lemma 17 implies that there exists $t \in T_2$ so that $d_P(f, t) \leq \varepsilon r/2$. If f is on an edge with length $\leq \varepsilon r/2$, let v be one of the endpoints of the edge. There exists $t \in T_2$ so that $d_P(v, t) \leq \varepsilon r/2$. Therefore, $d_P(f, t) \leq d_P(f, v) + d_P(v, t) \leq \varepsilon r$. Finally, if f is on an edge with length $\geq \varepsilon r/2$, then let the edge be e. There exists $O(\varepsilon^{-1})$ evenly spaced points on the chord $e \cap S'$ in T_3. Since the distance between consecutive points is $\leq \varepsilon r/2$, there exists a point $t_3 \in T_3$ so that $d_P(f, t_3) \leq \varepsilon r/2$. This completes the proof of Property (ii) and we are done. □

Our next step is to build a data structure analogous to Lemma 18, but for computing points that a_i can match to. To build this data structure, we first construct a three-dimensional low-density environment.

Definition 21. *A set of objects in $\mathbb{R}^3$ is k-low-density if, for every axis-parallel cube H_r with side length r, there are at most k objects that satisfy (i) the object intersects H_r, and (ii) the object has size at least r. The size of an object is the side length of the smallest axis-parallel cube that encloses the object.*

Definition 22. *Given a segment $e \subset \mathbb{R}^2$ and $\varepsilon \in \mathbb{R}^+$, we define* $\text{trough}(e, \varepsilon) \subset \mathbb{R}^3$ *to be*

$$\text{trough}(e, \varepsilon) = \{(x, y, z) : d((x, y), e) \leq 4z \leq 8\varepsilon^{-1}|e|\}$$

The three-dimensional object $\text{trough}(e, \varepsilon) \subset \mathbb{R}^3$ is the union of two half-cones and a triangular prism. See Figure 3.12.

The base of the half-cones have a radius of $8\varepsilon^{-1}|e|$. The half-cones and the triangular prism have a height of $2\varepsilon^{-1}|e|$. Next, we show that if $P = (V, E)$ is a c-packed graph, then the set of troughs $\{\text{trough}(e, \varepsilon) : e \in E\}$ is a low-density environment.

Lemma 23. *Let $P = (V, E)$ be a c-packed graph, and let $0 < \varepsilon < 1$. Then $\{\text{trough}(e, \varepsilon) : e \in E\}$ is an $O(c\varepsilon^{-1})$-low-density environment.*

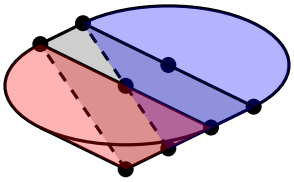


Figure 3.12: A trough is the union of two half-cones (red, blue) and a triangular prism (grey).

Proof. First, we show that trough(e, ε) has size at most $18\varepsilon^{-1}|e|$. Let $(x, y, z) \in$ trough(e, ε). Then $0 \leq z \leq 2\varepsilon^{-1}|e|$. Moreover, $d((x, y), e) \leq 8\varepsilon^{-1}|e|$, so (x, y) must lie inside a circle centred at the midpoint of e, with radius $9\varepsilon^{-1}|e|$. Therefore, (x, y) lies inside a circle with radius $9\varepsilon^{-1}|e|$. So (x, y, z) lies in a cylinder with radius $9\varepsilon^{-1}|e|$ and height $2\varepsilon^{-1}|e|$. So the size of trough(e, ε) is at most $18\varepsilon^{-1}|e|$, as claimed.

Let H_r be any axis parallel cube with side length r. Let z_{min} be the minimum z-coordinate of H_r. We can assume without loss of generality that $z_{min} \geq 0$. Suppose trough(e, ε) intersects with H_r and trough(e, ε) has size at least r. Let $(x, y, z) \in$ trough$(e, \varepsilon) \cap H_r$. Let h be the projection of the centre of H_r onto the hyperplane defined by $z = 0$. See Figure 3.13.

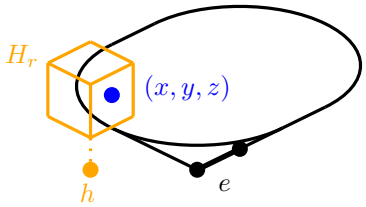


Figure 3.13: The point (x, y, z) (blue) is in the intersection of trough(e, ε) (black) and the cube H_r (orange). The point h (orange) and the edge e (black) are on the hyperplane $z = 0$.

Then,

$$\begin{aligned} d(h, e) &\leq d(h, (x, y)) + d((x, y), e) \\ &\leq r + 4z \\ &\leq 5r + 4z_{min} \end{aligned}$$

where the first inequality is the triangle inequality, the second comes from $(x, y, z) \in H_r \cap$ trough(e, ε), and the third comes from the maximum z-coordinate of H_r being $z_{min} + r$.

The maximum z-coordinate of trough(e, ε) is $2\varepsilon^{-1}|e|$, whereas the minimum z-coordinate of H_r is z_{min}. Therefore, $z_{min} \leq 2\varepsilon^{-1}|e|$. Since trough$(e, \varepsilon)$ has size at least r and at most $18\varepsilon^{-1}|e|$, we have $r \leq 18\varepsilon^{-1}|e|$. Putting this together, we have $5r + 4z_{min} \leq 100\varepsilon^{-1}|e|$.

Consider the ball $B(h, 10r + 8z_{min})$ centred at h with radius $10r + 8z_{min}$. Since $d(h, e) \leq 5r + 4z_{min}$, the length of e that is contained in $B(h, 10r + 8z_{min})$ is at least $\min(|e|, 5r + 4z_{min})$. But $100\varepsilon^{-1}|e| \geq 5r + 4z_{min}$. So the length of e that is contained in $B(s, 10r + 8z_{min})$ is at least $\varepsilon \cdot (5r + 4z_{min})/100$.

Finally, suppose there were k edges $\{e_i\}_{i=1}^{k}$ so that each e_i satisfies (i) trough(e_i) intersects H_r, and (ii) trough(e_i) has size at least r. Then by definition, the environment is k-low-density. It suffices to upper bound k. The total length of edges inside $B(h, 10r + 8z_{min})$ is at least $k\varepsilon \cdot (5r + 4z_{min})/100$. By the c-packedness of P, we have

$c \cdot (10r + 8z_{min}) \geq k\varepsilon \cdot (5r + 4z_{min})/100$, so $k \leq 50c\varepsilon^{-1}$. Therefore, for all H_r there are at most $50c\varepsilon^{-1}$ troughs that intersect H_r and have size at least r. This proves the stated lemma. □

Next, we use the result of Schwarzkopf and Vleugels [141] to build a range searching data structure for the low-density environment. Note that trough(e, ε) has constant description complexity.

Lemma 24 (Theorem 3 in [141])**.** *Let $\mathcal{E}$ be a set of n objects in $\mathbb{R}^3$, where each object has constant description complexity. Suppose that $\mathcal{E}$ is a k-low-density environment. Then $\mathcal{E}$ can be stored in a data structure of size $O(n \log^2 n + kn)$, such that it takes $O(\log^2 n + k)$ time to report all objects that contain a given query point $q \in \mathbb{R}^3$. The data structure can be computed in $O(n \log^2 n + kn \log n)$ time.*

We use Lemma 24 to construct a data structure for computing points that the vertices of the query trajectory (i.e. a_i for $1 \leq i \leq q$) can match to. In particular, for any square S in the plane, the following data structure returns a set of points $T \subset F$ satisfying the properties in Lemma 20.

Lemma 25. *Let $P = (V, E)$, let $F = \{f \in \mathbb{R}^2 : f \in e, e \in E\}$, and let $\varepsilon > 0$. One can construct a data structure in $O(p^2 \log p)$ time of size $O(p \log^2 p)$, so that given a query square S in the plane with side length $2r$, the data structure returns in $O(\log^2 p + c\varepsilon^{-2})$ time a set of points $T \subset F$ satisfying (i) $|T| = O(c\varepsilon^{-2})$ and (ii) for all points $f \in F \cap S$, there exists $t \in T$ so that $d_P(f, t) \leq \varepsilon r$.*

Proof. We construct the data structure in Lemma 18. For each edge $e \in E$, we construct trough(e, ε), and we use Lemma 24 to construct a range searching data structure on the set of troughs. Note that troughs form a low-density environment by Lemma 23. This completes the construction procedure.

Given a query square S, we use Lemma 18 to query a set T_2. Next, we state the query for T_3. Let the centre of S be (x, y), and its side length be $2r$. Query the data structure in Lemma 24 for all troughs that contain the query point (x, y, r). Suppose the data structure returns $\{\text{trough}(e_i)\}_{i=1}^{k}$. Let S' be the square concentric with S, but with side length $4r$. For each e_i, choose $O(\varepsilon^{-1})$ evenly spaced points on the chord $e_i \cap S'$, so that the distance between consecutive points is $\leq \varepsilon r/2$. This completes the query for set T_3.

Next, we prove the correctness of the query. For T_2, the proof of correctness follows from Lemma 18. For T_3, we require all edges with length at least $\varepsilon r/2$ that intersect S'. It suffices to show that querying Lemma 24 for all troughs containing the query point (x, y, r) is sufficient to obtain all such edges. Recall from the definition of the trough that $(x, y, r) \in \text{trough}(e, \varepsilon)$ if and only if $d((x, y), e) \leq 4r$ and $4r \leq 8\varepsilon^{-1}|e|$. Note that $d((x, y), e) \leq 4r$ covers all edges that intersect S', and $4r \leq 8\varepsilon^{-1}|e|$ covers all edges with length at least $\varepsilon r/2$. Therefore, the queries for T_2 and T_3 are correct. Lemma 20 proves that $T_2 \cup T_3$ satisfies Properties (i) and (ii) in the stated lemma.

The data structure in Lemma 18 has $O(p \log p)$ size. The data structure in Lemma 24 has $O(p \log^2 p + cp) = O(p \log^2 p)$ size. Therefore, the overall size of our data structure is $O(p \log^2 p)$.

The preprocessing time of Lemma 18 and Lemma 24 are $O(p^2 \log p)$ and $O(p \log^2 p + c\varepsilon^{-1} p \log p)$ respectively. Therefore, the overall preprocessing time is $O(p^2 \log p)$.

Finally, the query time of Lemma 18 is $O(\log p + c\varepsilon^{-1})$. The query time of Lemma 24 is $O(\log^2 n + c\varepsilon^{-1})$ time. Constructing the evenly spaced points takes $O(\varepsilon^{-1})$ time per

chord, and there are $O(c\varepsilon^{-1})$ chords overall. Therefore, the overall query time is $O(\log^2 p + c\varepsilon^{-2})$. □

Finally, we are ready to construct the map matching data structure for general trajectory queries on c-packed graphs.

Theorem 3. *Given a c-packed graph P of complexity p, one can construct a data structure of $O(p\log^2 p + c\varepsilon^{-4}\log(1/\varepsilon)p\log p)$ size, so that given a query trajectory Q of complexity q, the data structure returns in $O(q\log q\cdot(\log^4 p + c^4\varepsilon^{-8}\log^2 p))$ query time a $(1+\varepsilon)$-approximation of $\min_\pi d_F(\pi, Q)$ where π ranges over all paths in P and $d_F(\cdot,\cdot)$ denotes the Fréchet distance. The preprocessing time is $O(c^2\varepsilon^{-4}\log^2(1/\varepsilon)p^2\log^2 p)$.*

Proof. The preprocessing procedure is to build the data structures in Lemmas 16 and 25. For both data structures, we use the parameter $\varepsilon' = \varepsilon/9$ instead of ε.

Given a query trajectory Q and a Fréchet distance of r, the decision problem is to decide whether $r^* \le r$ or $r^* \ge r$, where $r^* = \min_\pi d_F(\pi, Q)$ as π ranges over all paths in P that start and end at a vertex of P. Recall that the vertices of Q are $a_1, \ldots, a_q$. We divide the decision problem into three steps. In step one, we construct a square of side length $2r$ centred at a_i. For $1 \le i \le q$, we query Lemma 25 with this square to obtain a set of points T_i that can match to a_i. In step two, we build a directed graph over $\cup_{i=1}^q T_i$, which we define as follows. Let $b_{i,j} \in T_i$ and $b_{i+1,k} \in T_{i+1}$. Let $c_{i,j}$ and $d_{i,j}$ be graph vertices so that $b_{i,j}$ is on the edge $c_{i,j}d_{i,j}$. Define $c_{i+1,k}$ and $d_{i+1,k}$ analogously. For now, we suppose that the path π passes through the pair of endpoints $(c_{i,j}, c_{i+1,k})$. Analogous arguments can be made if the path π instead passes through the pairs of endpoints $(c_{i,j}, d_{i+1,k})$, $(d_{i,j}, c_{i+1,k})$ or $(d_{i,j}, d_{i+1,k})$. Let a_i' be the point on a_ia_{i+1} that is the closest to a_i and satisfies $d_F(b_{i,j}c_{i,j}, a_ia_i') \le r$. Let a_{i+1}' be the point on a_ia_{i+1} that is the closest to a_{i+1} and satisfies $d_F(c_{i+1,k}b_{i+1,k}, a_{i+1}'a_{i+1}) \le r$. See Figure 3.14.

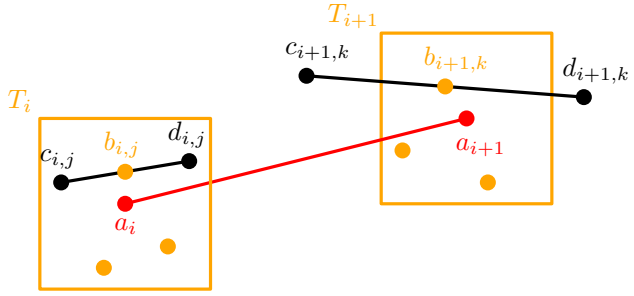


Figure 3.14: If $b_{i,j} \in T_i$ (orange) is a point that can match to a_i (red), we let the edge containing $b_{i,j}$ be $c_{i,j}d_{i,j}$ (black). The analogous edge $c_{i+1,k}d_{i+1,k}$ is defined for $b_{i+1,k}$.

We query the data structure in Lemma 16. From this, we obtain a $(1+\varepsilon')$-approximation of $\min_\pi d_F(\pi, a_i'a_{i+1}')$ where π ranges over all paths between the graph vertices $c_{i,j}$ and $c_{i+1,k}$. In our directed graph over $\cup_{i=1}^q T_i$, define the capacity of the directed edge from $b_{i,j}$ to $b_{i+1,k}$ to be the minimum of the four $(1+\varepsilon')$-approximations of $\min_\pi d_F(\pi, a_i'a_{i+1}')$ where π ranges over all paths between the pairs of vertices $(c_{i,j}, c_{i+1,k})$, $(c_{i,j}, d_{i+1,k})$, $(d_{i,j}, c_{i+1,k})$ and $(d_{i,j}, d_{i+1,k})$. This completes step two, that is, building the directed graph. The third step is, for $r' \in \{r, (1+\varepsilon')r\}$, to decide whether there exists a path in the directed graph from T_1 to T_q, so that the capacity of each edge in the path is at most r'. We distinguish three cases (a), (b) and (c):

(a) If there exists a path in the case $r' = r$, we return that $r^* \leq r$.
(b) If there does not exist a path in the case $r' = (1+\varepsilon')^2 r$, we return that $r^* \geq r$.
(c) If there exists a path in the case $r' = (1+\varepsilon')^2 r$ but not for the case $r' = r$, we return that

$$r^* \in [(1+\varepsilon')^{-2} r, (1+\varepsilon')^2 r].$$

Note that the third case does not technically answer the decision problem, however, in this case, we will show that $(1+\varepsilon')^2 r$ is a $(1+\varepsilon)$-approximation of r^*, as required by the theorem statement. This completes the description of the query procedure in the decision version. Next, we prove its correctness, which we separate into cases (a), (b) and (c).

(a) Suppose there exists a path with capacity at most r', where $r' \geq r$. Let this path be $b_1, \ldots, b_q$, where $b_i \in T_i$ for $1 \leq i \leq q$. Let the capacity of the directed edge from b_i to b_{i+1} be C_i. Then $C_i \leq r'$, by definition, so there exists graph vertices c_i and c_{i+1}, and points a_i' and a_{i+1}' on $a_i a_{i+1}$ satisfying $d_F(b_i c_i, a_i a_i') \leq r$, $d_F(b_{i+1} c_{i+1}, a_{i+1}' a_{i+1}) \leq r$, and $d_F(\pi_i, a_i' a_{i+1}') \leq C_i$ for some path π_i between c_i and c_{i+1}. Define π_i' to be the concatenation of $b_i c_i$, π_i and $c_{i+1} b_{i+1}$. So $d_F(\pi_i', a_i a_{i+1}) \leq \max(C_i, r) \leq r'$. Define π' to be the concatenation of π_i' for all $1 \leq i \leq q$. Then $d_F(\pi', Q) \leq r'$. Therefore, $r^* = \min_\pi d_F(\pi, Q) \leq d_F(\pi', Q) \leq r'$, so $r^* \leq r'$. In the first case, there exists a path for $r' = r$, so $r^* \leq r$, as required.
(b) Suppose there does not exist a path with capacity at most r'. Let π^* be the path in P so that $r^* = d_F(\pi^*, Q)$. Let the points on π^* that match to $a_1, \ldots, a_q \in Q$ be $u_1^*, \ldots, u_q^* \in P$. By Lemma 25, there exist points $b_i \in T_i$ so that $d_P(b_i, u_i^*) \leq \varepsilon' r$, for all $1 \leq i \leq q$. Let r_i be the minimum Fréchet distance $d_F(\pi, a_i a_{i+1})$ where π ranges over all paths between b_i and b_{i+1}. Then

$$r_i \leq \max(d_F(b_i u_i^*, a_i), d_F(\pi^*[u_i^*, u_{i+1}^*], a_i a_{i+1}), d_F(u_{i+1}^* b_{i+1}, a_{i+1})),$$

since the concatenation of $b_i u_i^*$, the subtrajectory $\pi^*[u_i^*, , u_{i+1}^*]$ of π^*, and $u_{i+1}^* b_{i+1}$ is a valid path from b_i to b_{i+1}. See Figure 3.15.
Note that $d_F(b_i u_i^*, a_i) \leq d_F(u_i^*, a_i) + d_P(b_i, u_i^*) \leq r^* + \varepsilon' r$. Therefore, $r_i \leq r^* + \varepsilon' r$. Then, the capacity of the edge from b_i to b_{i+1} is at most $(1+\varepsilon') r_i \leq (1+\varepsilon')(r^* + \varepsilon' r)$. Putting this together, there exists a path from T_1 to T_q with capacity at most $(1+\varepsilon')(r^* + \varepsilon' r)$. In the second case, there does not exist a path with capacity $r' = (1+\varepsilon')^2 r$. Therefore, $(1+\varepsilon')^2 r \leq (1+\varepsilon')(r^* + \varepsilon' r)$ which implies $(1+\varepsilon') r \leq r^* + \varepsilon' r$ and $r \leq r^*$, as required.
(c) From proof of the first case, $r^* \leq r'$, if there exists a path for r'. Therefore, $r^* \leq (1+\varepsilon')^2 r$. From the proof of the second case, $r^* \geq r'$ if there exists a path for $(1+\varepsilon')^2 r'$. Therefore, $r^* \geq (1+\varepsilon)^{-2} r$. Putting this together, we get $r^* \in [(1+\varepsilon')^{-2} r, (1+\varepsilon')^2 r]$. In particular, we have $(1+\varepsilon)^2 r \geq r^*$, and $(1+\varepsilon)^2 r \leq (1+\varepsilon')^4 r^* \leq (1+3\varepsilon')^2 r^* \leq (1+9\varepsilon') r^* = (1+\varepsilon) r^*$, so $(1+\varepsilon')^2 r$ is a $(1+\varepsilon)$-approximation of r^*, as required.

For the minimisation version, we apply parametric search. Define the decision problem $D(r)$ to be TRUE if there exists a path for $r' = r$, and FALSE if there does not exist a path for $r' = (1+\varepsilon')^2 r$.

It suffices to show (i) that $D(r)$ is monotone and (ii) that all operations in $D(r)$ are either independent of r, or can be made equivalent to a constant number of comparisons $\{r > c_i\}$ where c_i is a critical value. First we show (i). Suppose $D(r_1)$ evaluates to TRUE, and $r_1 < r_2$. Then $r^* \leq r_1 \leq r_2$ and we cannot have $D(r_2)$ evaluation to FALSE. So either

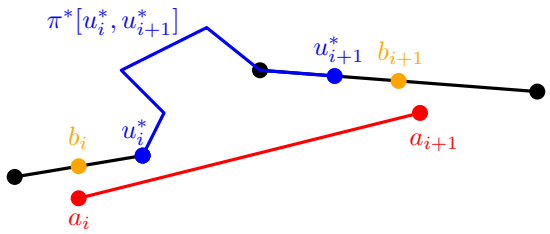


Figure 3.15: The trajectory vertices a_i and a_{i+1} (red) match to u_i^* and u_{i+1}^* on the optimal path π^* (blue). We construct a path from b_i to b_{i+1} (orange) by concatenating the edge $b_i u_i^*$, the subtrajectory $\pi^*[u_i^*, u_{i+1}^*]$ and $u_{i+1}^* b_{i+1}$.

$D(r_2)$ is also TRUE, or we halt the parametric search and obtain a $(1+\varepsilon)$-approximation of r^*. Similarly, if $D(r_1)$ evaluates to FALSE, and $r_1 > r_2$, then we cannot have $D(r_2)$ evaluating to TRUE. Therefore, $D(r)$ is monotone.

Next, we show (ii). The first step is to query Lemma 16 to obtain a set of points T_i that can match to a_i. The critical values of this step are when the query point (x, y, r) lies on the boundary of the troughs in the low-density environment. This can be evaluated as a low order polynomial in terms of r. The second step is to compute the points a_i and a_i', and query the data structure in Lemma 16 to obtain a $(1+\varepsilon')$-approximation of $\min_\pi d_F(\pi, a_i' a_{i+1}')$. The critical values occur when a_i and a_i' match to different pairs of grid points in Lemma 15, which can be resolved using a low order polynomial in terms of r. The third step is to decide whether there exists a path in the directed graph where the capacity of each edge is at most r'. The critical values are when the capacity of an edge is exactly r', which can be resolved as a low order polynomial of r. This completes the proof of (i) and (ii).

We analyse the space and preprocessing time of the data structure. First, we analyse the space. The data structures in Lemmas 16 and 25 require $O(c\varepsilon^{-4}\log(1/\varepsilon)p\log p)$ and $O(p\log^2 p)$ space respectively. The overall space requirement is $O(c\varepsilon^{-4}\log(1/\varepsilon)p\log p + p\log^2 p)$. Next, the preprocessing times of Lemmas 16 and 25 are $O(c^2\varepsilon^{-4}\log^2(1/\varepsilon)p^2\log^2 p)$ and $O(p^2\log p)$ respectively. The overall preprocessing time is $O(c^2\varepsilon^{-4}\log^2(1/\varepsilon)p^2\log^2 p)$.

For the query time of the decision version, for each $1 \le i \le q$ we query the data structure in Lemma 20 to construct the set T_i. In total, this takes $O(q\cdot(\log^2 p + c\varepsilon^{-2}))$ time. Next, we build a directed graph on $\cup_{i=1}^q T_i$. There are $O(q\cdot c^2\varepsilon^{-4})$ directed edges between T_i and T_{i+1}, for $1 \le i < q$. Computing the capacity of the directed edge, by querying Lemma 16, takes $O(\log p + c\varepsilon^{-1})$ time. Finally, deciding whether there is a directed path from T_1 to T_q takes $O(q\cdot c^2\varepsilon^{-4})$ time. Therefore, the overall running time of the decision version is $O(q\cdot(\log^2 p + c^2\varepsilon^{-4}\log p))$.

Finally, we analyse the running time of parametric search. The running time of parametric search [126] is $O(N_p T_p + T_p T_s \log N_p)$, where T_s is the running time of the sequential algorithm, N_p is the number of processors for the parallel algorithm, and T_p is the number of parallel steps in the parallel algorithm. The sequential algorithm is the same as the decision algorithm, so $T_s = O(q\cdot(\log^2 p + c^2\varepsilon^{-4}\log p))$. The parallel algorithm is to simulate the decision algorithm on $N_p = q$ processors. The first two steps can be parallelised to run on q processors in $O(\log^2 p + c^2\varepsilon^{-4}\log p)$ parallel steps. In the third step, it suffices to check if each of the edges has capacity at most r'. Computing the directed path generates no additional critical values, so it does not need to be simulated by the parallel algorithm.

The third step can be parallelised onto q processors to run in $O(c^2\varepsilon^{-4})$ parallel steps. The total number of parallel steps is $T_p = O(\log^2 p + c^2\varepsilon^{-4}\log p)$. Therefore, substituting these values into the running time of parametric search, we get that the overall running time of parametric search is $O(q \log q \cdot (\log^4 p + c^4\varepsilon^{-8}\log^2 p))$. This completes the proof of the theorem. □

3.7 Lower bound for geometric planar graphs

In this section, we no longer assume that the graph P is c-packed. The main result of this section is that for geometric planar graphs, unless SETH fails, no data structure can preprocess a graph in polynomial time to answer map matching queries in $O((pq)^{1-\delta})$ for any $\delta > 0$, and for any polynomial restrictions of p and q. In Section 3.7.1, we construct a lower bound for the warm-up problem of Fréchet distance queries on trajectories. In Section 3.7.2, we construct a lower bound for map matching queries on geometric planar graphs.

3.7.1 Fréchet distance queries on trajectories

Our warm-up problem is to extend the lower bound of Bringmann [28] to Fréchet distance queries on trajectories. The lower bound assumes a weaker version of SETH.

Definition 26 (SETH′)**.** *The CNF-SAT problem is to decide whether a formula φ on N variables $x_1, \dots x_N$ and M clauses $C_1, \dots, C_M$ has a satisfying assignment. SETH′ states that there is no $\tilde{O}((2-\delta)^N)$ algorithm for CNF-SAT for any $\delta > 0$, where $\tilde{O}$ hides polynomial factors in N and M.*

If SETH′ fails, then so does SETH [164]. Next, we provide a proof sketch of Theorem 1.2 in [28].

Lemma 27 (Theorem 1.2 in [28])**.** *Let n and m denote the complexities of a pair of trajectories. There is no 1.001-approximation with running time $O((nm)^{1-\delta})$ for the Fréchet distance for any $\delta > 0$, unless SETH fails. This holds for any polynomial restrictions of n and m.*

Proof (Sketch). Suppose for the sake of contradiction that there exists a positive constant δ so that there is a 1.001-approximation with running time $O((nm)^{1-\delta})$ for the Fréchet distance. We summarise the main steps of Theorem 1.2 in [28], which generalises the $m = n$ case to the $m \neq n$ case.

Suppose $m = \Theta(n^\gamma)$ for some $\gamma > 0$. We are given a CNF-SAT instance φ with variables $x_1, \dots, x_N$ and clauses $C_1, \dots, C_M$. We partition its variables $x_1, \dots, x_N$ into $V_1 = \{x_1, \dots, x_\ell\}$ and $V_2 = \{x_{\ell+1}, \dots, x_N\}$, where $\ell = N/(\gamma+1)$. Let A_k be all the assignments of V_k for $k \in \{1, 2\}$. Using the same method as the $m = n$ case in [28], we construct curves P_1 and P_2 so that $|P_1| = \Theta(M \cdot |A_1|)$ and $|P_2| = \Theta(M \cdot |A_2|)$. Moreover, by Lemma 3.7 and Lemma 3.9 in [28], if $A_1 \times A_2$ contains a satisfying assignment, then $d_F(P_1, P_2) \leq 1$, whereas if $A_1 \times A_2$ contains no satisfying assignment, then $d_F(P_1, P_2) \geq 1.001$. Note that if $n = |P_1|$, then $m = |P_2| = \Theta(n^\gamma)$.

Therefore, any 1.001-approximation of $d_F(P_1, P_2)$ with running time $O((nm)^{1-\delta})$ yields an algorithm for CNF-SAT with running time $O((M \cdot |A_1|)^{1-\delta}(M \cdot |A_2|)^{1-\delta}) = O(M^2 2^{(1-\delta)N})$, contradicting SETH′ and SETH. □

Next, we consider the problem of preprocessing a trajectory such that, given a query trajectory, their Fréchet distance can be computed efficiently. This is stated as an extremely challenging problem in Buchin et al. [46]. In Lemma 28 we show that preprocessing essentially does not help with computing the Fréchet distance. In particular, we show that even with polynomial preprocessing time, one cannot obtain a truly subquadratic query time for computing the Fréchet distance. We prove this by considering the offline version of the data structure problem, in a similar fashion to Bringmann et al. [29] and Rubinstein [139].

Lemma 28. *Let n denote the complexity of a trajectory. There is no data structure that can be constructed in* $\text{poly}(n)$ *time, that when given a query trajectory of complexity m, can answer* 1.001*-approximate Fréchet distance queries in $O((nm)^{1-\delta})$ query time for any $\delta > 0$, unless SETH fails. This holds for any polynomial restrictions of n and m.*

Proof. Suppose for the sake of contradiction that there exists positive constants α and δ so that one can construct a data structure in $O(n^\alpha)$ preprocessing time to answer 1.001-approximate Fréchet distance queries with a query time of $O((nm)^{1-\delta})$.

Suppose $m = \Theta(n^\gamma)$ for some $\gamma > 0$. We take two cases. In the first case, $\gamma \geq 2\alpha$. Given a pair of trajectories with complexities n and m, we can preprocess the first trajectory in $O(n^\alpha)$ time, and query a 1.001-approximation of its Fréchet distance with the second trajectory in $O((nm)^{1-\delta})$ time. But $O(n^\alpha) = O(m^{1/2})$, so the overall running time is $O((nm)^{1-\delta})$. This contradicts Lemma 27.

In the second case, $\gamma \leq 2\alpha$. We follow the same steps as Lemma 27. We are given a CNF-SAT instance φ with variables $x_1, \ldots, x_N$ and clauses $C_1, \ldots, C_M$. We partition its variables $x_1, \ldots, x_N$ into $V_1 = \{x_1, \ldots, x_\ell\}$ and $V_2 = \{x_{\ell+1}, \ldots, x_N\}$, where $\ell = N/(2\alpha + 1)$. Let A_k be all the assignments of V_k for $k \in \{1, 2\}$. Note that if $n = |A_1|$, then $m = |A_2| = \Theta(n^{2\alpha})$. Partition the set A_2 into subsets $B_1, \ldots, B_K$ so that $|B_i| = \Theta(|A_1|^\gamma)$, for all $1 \leq i \leq K$, and $K = O(|A_1|^{2\alpha-\gamma})$. Note that $A_1 \times A_2$ contains a satisfying assignment if and only if there exists $1 \leq i \leq K$ so that $A_1 \times B_i$ contains a satisfying assignment. Using the same method as the $m = n$ case in [28], we construct curves P_1 and Q_i so that $|P_1| = \Theta(M|A_1|)$ and $|Q_i| = \Theta(M|B_i|)$ for $1 \leq i \leq K$. Moreover, by Lemma 3.7 and Lemma 3.9 in [28], if $A_1 \times B_i$ contains a satisfying assignment, then $d_F(P_1, Q_i) \leq 1$, whereas if $A_1 \times B_i$ contains no satisfying assignment, then $d_F(P_1, Q_i) \geq 1.001$. Note that if $n = |P_1|$, then $m = |Q_i| = \Theta(n^\gamma)$.

Therefore, to decide if there is a satisfying assignment for the CNF-SAT instance φ, it suffices to query a 1.001-approximation of $d_F(P_1, Q_i)$ for all $1 \leq i \leq K$. We preprocess the trajectory P_1 in $O((M|A_1|)^\alpha) = O(M^\alpha 2^{\alpha N/(2\alpha+1)}) = O(M^\alpha 2^{N/2})$ time. We answer all K queries in time

$$\begin{aligned} O(\textstyle\sum_{i=1}^{K}(M|A_1|)^{1-\delta}(M|B_i|)^{1-\delta}) &= O(KM^2|A_1|^{1-\delta}|A_1|^{(1-\delta)\gamma}) \\ &= O(M^2|A_1|^{(1-\delta)+(1-\delta)\gamma+2\alpha-\gamma}) \\ &= O(M^2|A_1|^{(1-\delta)+2\alpha}) \\ &= O(M^2 2^{N(1+2\alpha-\delta)/(1+2\alpha)}) \\ &= O(M^2 2^{(1-\delta/(1+2\alpha))N}). \end{aligned}$$

Putting this together, we yield an algorithm for CNF-SAT with running time

$$O(M^\alpha 2^{N/2} + M^2 2^{(1-\frac{\delta}{1+2\alpha})N}),$$

where α and δ are constants, contradicting SETH′and SETH. □

An open problem is whether one can adapt the lower bound of Buchin et al. [42] to rule out approximation factors between 1.001 and 3. In particular, one would need to extend their construction to hold for a pair of trajectories with an imbalanced number of vertices.

Another open problem is whether one can extend the lower bound to range searching queries. Given a database of k trajectories with m vertices each and a query trajectory with n vertices, Baldus and Bringmann [18] conjecture that a $O((kmn)^{1-\delta})$ time algorithm for range searching would falsify SETH.

3.7.2 Map matching queries on geometric planar graphs

We return to the map matching problem. The main result of this section is that there is no data structure that can preprocess a geometric planar graph in polynomial time to answer map matching queries in truly subquadratic time. To build towards this result, we first show that Alt et al.'s [11] $O(pq \log p)$ time algorithm for Problem 1 is optimal up to lower-order factors, conditioned on unbalanced OVH.

Definition 29 (OVH)**.** *The OV problem is to decide whether the sets $A, B \subseteq \{0,1\}^d$ contain a pair of binary vectors $(a, b) \in A \times B$ so that a and b are orthogonal. Let $n = |A|$ and $m = |B|$. OVH states that there is no $\tilde{O}((nm)^{1-\delta})$ time algorithm for OV for any $\delta > 0$, where $\tilde{O}$ hides polynomial factors in d. This holds for any polynomial restrictions of n and m.*

If OVH fails, then so does SETH [164]. We use OVH to prove our lower bound for Problem 1. For constructing our graph and our trajectory, we use the notation $Q = \bigcirc_{i=1}^{q} a_i = a_1 \circ \ldots \circ a_q$ to denote the polygonal curve Q obtained by linearly interpolating between the vertices $a_1, \ldots, a_q$.

Lemma 30. *Let P be a geometric planar graph of complexity p and Q be a trajectory of complexity q. There is no 2.999-approximation with running time $O((pq)^{1-\delta})$ for computing $\min_\pi d_F(\pi, Q)$ for any $\delta > 0$, unless SETH fails. This holds for any polynomial restrictions of p and q.*

Proof. Suppose for the sake of contradiction that there exists a positive constant δ so that there is a 2.999-approximation with running time $O((pq)^{1-\delta})$ for Problem 1.

We are given an OV instance $A, B \subseteq \{0,1\}^d$. Let $p = |A|$ and $q = |B|$, where there may be any polynomial restriction of p and q. First, we will construct a graph P of complexity $O(dp)$ and a trajectory Q of complexity $O(dq)$. Then we will show that a 2.999-approximation of $\min_\pi d_F(\pi, Q)$ yields an $O((pq)^{1-\delta})$ time algorithm for OV.

Let h be a small constant, which we will choose later on in the proof. Inspired by Buchin et al. [42] and Bringmann et al. [29], we define the following polygonal curves. See Figure 3.16. It is straightforward to verify that $d_F(0_A, 0_B) = d_F(0_A, 1_B) = d_F(1_A, 0_B) = 1$, and $d_F(1_A, 1_B) = 3$.

$$\begin{aligned}
1_A &:= (0,0) \circ (12,0) \circ (12,h) \circ (6,h) \circ (6,2h) \circ (18,2h) \\
0_B &:= (0,0) \circ (13,0) \circ (13,h) \circ (5,h) \circ (5,2h) \circ (18,2h) \\
0_A &:= (0,0) \circ (14,0) \circ (14,h) \circ (4,h) \circ (4,2h) \circ (18,2h) \\
1_B &:= (0,0) \circ (15,0) \circ (15,h) \circ (3,h) \circ (3,2h) \circ (18,2h)
\end{aligned}$$

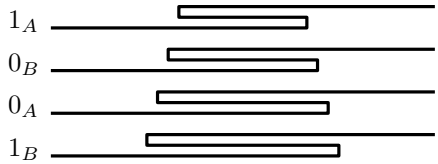


Figure 3.16: The polygonal curves 1_A, 0_B, 0_A and 1_B.

We use 0_A and 1_A to construct P. We start by constructing the curves R, S and T_i.

$$\begin{array}{rcl} R & := & \bigcirc_{k=1}^{d} R_k \quad \text{where} \quad R_k := 0_A + (18k, 0), \\ S & := & \bigcirc_{k=1}^{d} S_k \quad \text{where} \quad S_k := 0_A + (18k, 7ph), \\ T_i & := & \bigcirc_{j=1}^{d} T_{i,k} \quad \text{where} \quad T_{i,k} := A[i][k]_A + (18k, 3ih) \quad \text{for all} \quad 1 \le i \le p, \quad 1 \le k \le d \end{array}$$

where $A[i][k]$ is the k^{th} coordinate of the i^{th} vector in A, and $A[i][k]_A$ is either 0_A or 1_A depending on whether $A[i][k]$ is 0 or 1, and $+(x, y)$ translates the curve horizontally by x and vertically by y. See Figure 3.17.

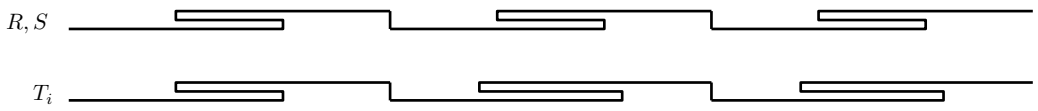


Figure 3.17: The curves R and S are obtained by concatenating translated versions of 0_A. The curves T_i are obtained by concatenating translated versions of 0_A and 1_A.

Next, we use R and S to construct curves U and V. Note that U and V each contain a loop.

$$\begin{array}{rcl} U & := & (0, -18) \circ (0, 0) \circ R \circ (36d, 3h) \circ (0, 3h) \circ (0, 0), \\ V & := & (0, 6ph) \circ S \circ (36d, 6ph) \circ (0, 6ph) \circ (0, 18), \end{array}$$

See Figure 3.18, left. Finally, we connect the curves T_i, U and V to obtain the graph P. For $1 \le i \le p$, we connect U to T_i with the edge $(0, 3h) \circ (18, 3ih)$. For $1 \le i \le p$, we connect T_i to V with the edge $(18d, 3ih + 2h) \circ (36d, 6ph)$. We take the union of T_i, U, V, and these $2p$ connections to obtain the graph P, completing its construction. See Figure 3.18, right. It is straightforward to verify that P is connected and planar, and $|P| = O(dp)$.

Now, we use 0_B and 1_B to construct Q.

$$\begin{array}{rcl} W_j & := & \bigcirc_{j=1}^{d} W_{j,k} \quad \text{where} \quad W_{j,k} := B[j][k]_B + (18k, 0) \quad \text{for all} \quad 1 \le j \le q, \quad 1 \le k \le d, \\ X & := & \bigcirc_{j=1}^{q} X_j \quad \text{where} \quad X_j := (0, 0) \circ W_j \circ (36d, 0) \circ (0, 3h) \quad \text{for all} \quad 1 \le j \le q, \\ Q & := & (0, -18) \circ X \circ (0, 18). \end{array}$$

Note that $B[j][k]$ is the k^{th} coordinate of the j^{th} vector in B, where $B[j][k]_B$ is either 0_B or 1_B depending on whether $B[j][k]$ is 0 or 1, and $+(x, y)$ translates the curve horizontally by x and vertically by y. This completes the construction of Q. See Figure 3.19. It is straightforward to verify that $|Q| = O(dq)$.

We will show that if our OV instance A, B is a YES-instance, then $\min_\pi d_F(\pi, Q) \le 1.001$. Suppose that $A[i]$ and $B[j]$ are orthogonal. We will construct a path $\pi \in P$ with

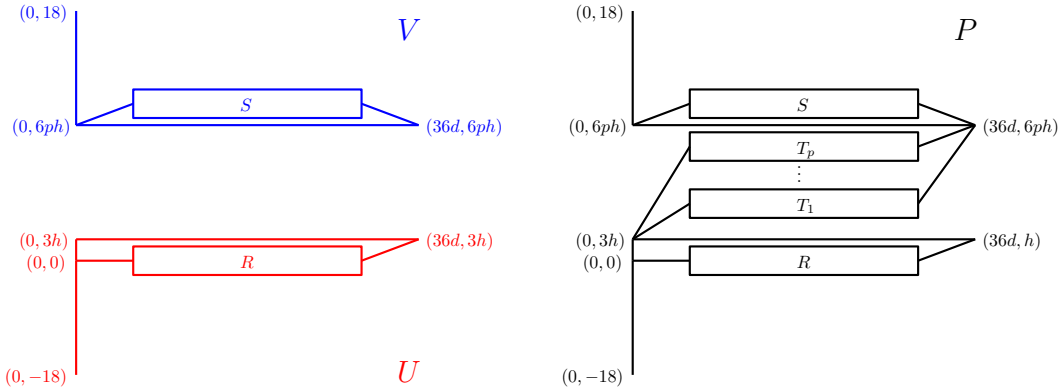


Figure 3.18: (Left) The lower curve U in red, the upper curve V in blue. (Right) The graph P obtained by connecting U, V and T_i.

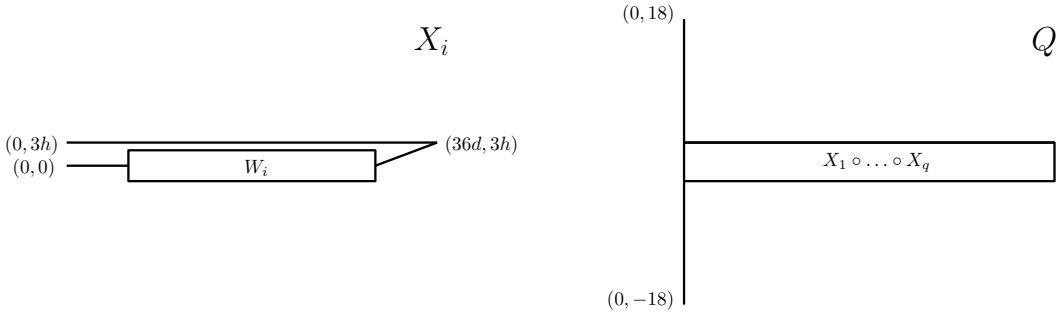


Figure 3.19: (Left) Curve X_i is obtained by concatenating $(0,0)$, W_i, $(36d, 3h)$ and $(0, 3h)$. (Right) Curve Q is obtained by concatenating $(0, -18)$, X_i for $1 \le i \le q$, and $(0, 18)$.

$d_F(\pi, Q) \le 1.001$, which we define as follows.

$$\begin{aligned} Y &:= \bigcirc_{\ell=1}^{j-1} Y_\ell \quad \text{where} \quad Y_\ell := (0,0) \circ R \circ (36d, 3h) \circ (0, 3h) \quad \text{for all} \quad 1 \le \ell \le j-1, \\ Z &:= \bigcirc_{\ell=j+1}^{q} Z_\ell \quad \text{where} \quad Z_\ell := (0, 6ph) \circ S \circ (36d, 6ph) \circ (0, 6ph) \quad \text{for all} \quad j+1 \le \ell \le q, \\ \pi &:= (0, -18) \circ Y \circ T_i \circ Z \circ (0, 18), \end{aligned}$$

It is straightforward to verify that π is a path of P. Next, we provide a matching of π and Q with Fréchet distance at most 1.001, assuming h is sufficiently small.

$$\begin{aligned} \pi \;=\; & (0,-18) \circ (0,0) \\ & \circ \bigcirc_{\ell=1}^{j-1} Y_\ell \\ & \circ T_i \\ & \circ \bigcirc_{\ell=j+1}^{q} Z_\ell \\ & \circ (0, 6ph) \circ (0, 18) \end{aligned} \qquad \begin{aligned} Q \;=\; & (0,-18) \circ (0,0) \\ & \circ \bigcirc_{\ell=1}^{j-1} X_\ell \\ & \circ X_j \\ & \circ \bigcirc_{\ell=j+1}^{q} X_\ell \\ & \circ (0, 3h) \circ (0, 18) \end{aligned}$$

It suffices to show that $d_F(Y_\ell, X_\ell) \le 1$, $d_F(T_i, X_j) \le 1$ and $d_F(Z_\ell, X_\ell) \le 1.001$, for a sufficiently small choice of h. This is equivalent to showing that $d_F(R, W_\ell) \le 1$, $d_F(T_i, W_j) \le 1$, and $d_F(S, W_\ell) \le 1.001$. We traverse these pairs of trajectories synchronously. For $1 \le k \le d$, we have $R_k = 0_A + (18k, 0)$, $W_{j,k} = B[j][k]_B + (18k, 0)$, $T_{i,k} = A[i][k]_A + (18k, 0)$ and

$S_k = 0_A + (18k, 7ph)$. Since $d_F(0_A, B[j][k]_B) \leq 1$, we have $d_F(R_k, W_{j,k}) \leq 1$. Putting this together for $1 \leq k \leq d$, we have $d_F(R, W_\ell) \leq 1$. Also, we have $d_F(A[i][k]_A, B[j][k]_B) \leq 1$ for all $1 \leq k \leq d$, since $A[i]$ and $B[k]$ are orthogonal. Therefore, $d_F(T_{i,k}, W_{j,k}) \leq 1$, and putting this together for $1 \leq k \leq d$, we have $d_F(T_i, W_j) \leq 1$. Finally, since $d_F(0_A, B[j][k]_B) \leq 1$, we have $d_F(S_k, W_{j,k}) \leq 1 + 7ph$. We can obtain that $1 + 7ph \leq 1.001$ by setting $h = 0.0001/p$. Putting this together for $1 \leq k \leq d$, we have $d_F(S, W_\ell) \leq 1.001$. To summarise, we have $d_F(Y_\ell, X_\ell) \leq 1$, $d_F(T_i, X_j) \leq 1$ and $d_F(Z_\ell, X_\ell) \leq 1.001$, so $d_F(\pi, Q) \leq 1.001$ as required.

We show that if our OV instance A, B is a NO-instance, then $\min_\pi d_F(\pi, Q) \geq 3$. Suppose for the sake of contradiction that A, B is a NO-instance but there exists $\pi \in P$ so that $d_F(\pi, Q) < 3$. First, note that π must start at $(0, -18)$ and end at $(0, 18)$ since no other vertices in P can match to the start and end points of Q. Note that $(0, -18) \in U$ and $(0, 18) \in V$, and that any path π from U to V must pass through one of the curves T_i. Without loss of generality, let T_i be a subcurve of π. Consider the point $(22, 3ih) \in T_i \subset \pi$. This point must match to some point in Q that is not on the edges $(0, -18) \circ (0, 0)$ or $(0, 3h) \circ (0, 18)$. Therefore, there exists some j so that $(22, 3ih) \in \pi$ matches to a point on X_j. The point $(22, 3ih)$ cannot match to any of the edges $(0, 0) \circ (18, 0)$, $(18(d+1), 2h) \circ (36d, 0)$, $(36d, 0) \circ (0, 3h)$ or $(0, 3h) \circ (0, 0)$ on X_j. Therefore, the point $(22, 3ih) \in T_i$ that matches to a point on W_j. As the path $W_j \subset Q$ is traversed, the path π must continue to traverse the path along T_i, since T_i is an isolated path that only connects to the rest of P at its endpoints. Therefore, T_i and W_j are traversed simultaneously. Specifically, the subcurves $T_{i,k} \subset T_i$ and $W_{j,k} \subset W_j$ are traversed simultaneously, since no points on $T_{i,k}$ can match to points on $W_{j,k'}$ for all $k \neq k'$. This implies $d_F(\pi, Q) \geq d_F(T_{i,k}, W_{j,k})$ for all $1 \leq k \leq d$.

Finally, we use the fact that A, B is a NO-instance to show that $d_F(T_{i,k}, W_{j,k}) = 3$ for some $1 \leq k \leq d$. Since A, B is a NO-instance, $A[i]$ and $B[j]$ are not orthogonal. Therefore, there exists a k so that $A[i][k] = B[j][k] = 1$. Therefore, $d_F(A[i][k]_A, B[j][k]_B) = 3$. Since $T_{i,k} = A[i][k]_A + (18k, 3ih)$ and $W_{j,k} = B[j][k]_B + (18k, 0)$, we have $d_F(A[i][k]_A, B[j][k]_B) = 3 + 3ih \geq 3$. Therefore, $d_F(\pi, Q) \geq d_F(T_{i,k}, W_{j,k}) > 3$, contradicting the fact that $d_F(\pi, Q) < 3$. Therefore, if our OV instance A, B is a NO-instance, then $\min_\pi d_F(\pi, Q) \geq 3$ as required.

To summarise, we can decide if A, B is a YES-instance or a NO-instance by deciding whether $\min_\pi d_F(\pi, Q) \leq 1.001$ or $\min_\pi d_F(\pi, Q) \geq 3$. Therefore, any 2.999-approximation with running time $O((pq)^{1-\delta})$ yields an algorithm for OV with running time $O(d^2(pq)^{1-\delta})$, where $p = |A|$ and $q = |B|$. This contradicts OVH and SETH. □

Finally, we combine the ideas in Lemma 28 and 30 to obtain the main theorem of the section. The theorem essentially states that for geometric planar graphs, preprocessing does not help for map matching. In particular, we show that even with polynomial preprocessing time on the graph, one cannot obtain a truly subquadratic query time for the map matching problem in Problem 2.

Theorem 8. *Given a geometric planar graph of complexity p, there is no data structure that can be constructed in* $\text{poly}(p)$ *time, that when given a query trajectory of complexity q, can answer* 2.999*-approximate map matching queries in $O((pq)^{1-\delta})$ query time for any $\delta > 0$, unless SETH fails. This holds for any polynomial restrictions of p and q.*

Proof. Suppose for the sake of contradiction that there exists positive constants α and δ so that one can construct a data structure in $O(n^\alpha)$ preprocessing time to answer 2.999-approximate map matching queries with a query time of $O((pq)^{1-\delta})$.

Suppose $q = \Theta(p^\gamma)$ for some $\gamma > 0$. We take two cases. In the first case, $\gamma \geq 2\alpha$. Given a geometric planar graph of complexity p and a trajectory of complexity q, we can preprocess

the graph in $O(p^\alpha)$ time, and query a 2.999-approximation of $\min_\pi d_F(\pi, Q)$ in $O((pq)^{1-\delta})$ time. But $O(p^\alpha) = O(q^{1/2})$, so the overall running time is $O((pq)^{1-\delta})$. This contradicts Lemma 30.

In the second case, $\gamma \leq 2\alpha$. We are given an OVH instance A, B where $|A| = n$ and $|B| = m$. Since OVH holds for any polynomial restrictions of n and m, we may assume that $m = \Theta(n^{2\alpha})$. Partition the set B into subsets $B_1, \ldots, B_K$ so that $|B_i| = \Theta(|A|^\gamma)$, for all $1 \leq i \leq K$, and $K = O(|A|^{2\alpha-\gamma})$. Note that $A \times B$ contains a pair of orthogonal vectors if and only if there exists $1 \leq i \leq K$ so that $A \times B_i$ contains a pair of orthogonal vectors. Given the OV instance A, B_i, we use Lemma 30 to construct a geometric planar graph P and a trajectories Q_i so that $p = |P| = O(d|A|) = O(dn)$ and $q = |Q_i| = O(d|B_i|) = O(dn^\gamma)$. Moreover, by Lemma 30, if (A, B_i) is a YES-instance, then $\min_\pi(\pi, Q_i) \leq 1.001$, whereas if (A, B_i) is a NO-instance, then $\min_\pi(\pi, Q_i) \geq 3$. Note that $q = \Theta(p^\gamma)$.

Therefore, to decide if A, B is a YES-instance or a NO-instance, it suffices to query a 2.999-approximation of $\min_\pi(\pi, Q_i)$ for all $1 \leq i \leq K$. Recall that $m = \Theta(n^{2\alpha})$. We preprocess the graph P in $O((dn)^\alpha) = O(d^\alpha m^{1/2})$ time. We answer all K queries in time

$$\begin{aligned} O(\textstyle\sum_{i=1}^{K}(dn)^{1-\delta}(dn^\gamma)^{1-\delta}) &= O(Kd^2n^{1-\delta+\gamma}) \\ &= O(d^2n^{(1-\delta+\gamma+2\alpha-\gamma)}) \\ &= O(d^2n^{2\alpha+1-\delta}) \\ &= O(d^2n^{(2\alpha+1)(1-\delta/(1+2\alpha))}) \\ &= O(d^2(mn)^{1-\delta/(1+2\alpha)}). \end{aligned}$$

Putting this together, we yield an algorithm for OVH with running time

$$O(d^\alpha m^{1/2} + d^2(mn)^{1-\frac{\delta}{1+2\alpha}}),$$

where α and δ are constants. This contradicts OVH under the polynomial restriction $m = \Theta(n^{2\alpha})$, thereby contradicting SETH. □

3.8 Conclusion

We showed that for c-packed graphs, one can construct a data structure of near-linear size, so that map matching queries can be answered in time near-linear in terms of the query complexity, and polylogarithmic in terms of the graph complexity. We showed that for geometric planar graphs, there is no data structure for answering map matching queries in truly subquadratic time, unless SETH fails.

Our map matching queries return the minimum Fréchet distance between the query trajectory and any path in an undirected graph. The data structure can be modified for directed graphs and for matched paths that start and end along edges of the graph. We can also modify the data structure to return the length of the minimum Fréchet distance path. More generally, one can modify the data structure to return $\sum_{e\in\pi} f(e)$ for any function f, where π is the path with approximate minimum Fréchet distance. One application of this is fare estimation for ride-sharing services.

Our data structures return the minimum Fréchet distance of the matched path. One can modify our data structure to retrieve the path that attains the minimum Fréchet distance, however, the space requirement would increase to quadratic. An open problem is whether one can obtain a map matching data structure that retrieves the matched path, and uses subquadratic space. Another open problem is whether one can make ε chooseable at query time, rather than at preprocessing time.

Yet another direction for future work is to improve the preprocessing, size, and query time of the data structure. Can one improve the preprocessing time to subquadratic? Can one reduce the dependencies on c, ε^{-1}, $\log q$ and $\log p$? For example, can one improve the query time by avoiding parametric search? Avoiding parametric search would also make the algorithm more likely to be implementable in practice.

Another practical consideration is verifying whether real-world road networks are indeed c-packed. Since these road networks contain upwards of a million edges [53], a faster implementation for computing the c-packedness value of a graph [100] would be required. If real-world road networks are not c-packed, an interesting direction for future work would be to consider other realistic input models, such as ϕ-low-density, which have small values of ϕ even on large road networks [54].

Finally, two open problems are proposed in Section 3.7. Can one modify the lower bound of Buchin et al. [42] to rule out approximation ratios between 1.001 and 3 for preprocessing a trajectory to answer Fréchet distance queries in truly subquadratic time? Can one extend the lower bounds to rule out efficient data structures for other Fréchet distance queries, for example, range searching queries?

Chapter 4

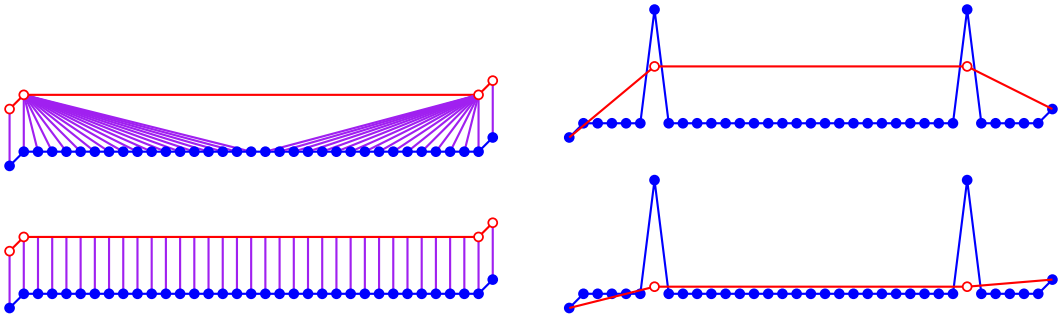
Computing Continuous Dynamic Time Warping of Time Series in Polynomial Time

4.1 Introduction

Time series data arises from many sources, such as financial markets [153], seismology [168], electrocardiography [20] and epidemiology [22]. Domain-specific questions can often be answered by analysing these time series. A common way of analysing time series is by finding similarities. Computing similarities is also a fundamental building block for other analyses, such as clustering, classification, or simplification. There are numerous similarity measures considered in literature [16, 58, 80, 116, 151, 155], many of which are application dependent.

Dynamic Time Warping (DTW) is arguably the most popular similarity measure for time series, and is widely used across multiple communities [6, 93, 129, 132, 140, 142, 152, 156, 160]. Under DTW, a minimum cost discrete alignment is computed between a pair of time series. A discrete alignment is a sequence of pairs of points, subject to the following four conditions: *(i)* the first pair is the first sample from both time series, *(ii)* the last pair is the last sample from both time series, *(iii)* each sample must appear in some pair in the alignment, and *(iv)* the alignment must be a monotonically increasing sequence for both time series. The cost of a discrete alignment, under DTW, is the sum of the distances between aligned points. A drawback of a similarity measure with a discrete alignment is that it is sensitive to the sampling rates of the time series. As such, DTW is a poor measure of similarity between a time series with a high sampling rate and a time series with a low sampling rate. For such cases, it is more appropriate to use a similarity measure with a continuous alignment. In Figure 4.1a, we provide a visual comparison of a discrete alignment versus a continuous alignment, for time series with vastly different sampling rates.

The Fréchet distance is a similarity measure that has gained popularity, especially in the theory community [12, 37, 69, 148]. To apply the Fréchet distance to a time series, we linearly interpolate between sampled points to obtain a continuous one-dimensional polygonal curve. Under the Fréchet distance, a minimum cost continuous alignment is computed between the pair of curves. A continuous alignment is a simultaneous traversal of the pair of curves that



(a) Top: The optimal alignment under a discrete similarity measure, e.g. DTW. Bottom: The optimal alignment under a continuous similarity measure.

(b) Top: The optimal simplification under a bottleneck measure, e.g., Fréchet distance. Bottom: The optimal simplification under a summation-based similarity measure.

Figure 4.1: Issues with discrete (left) and bottleneck (right) measures as opposed to continuous, summed measures.

satisfies the same four conditions as previously stated for DTW. The cost of a continuous alignment, under the Fréchet distance, is the maximum distance between a pair of points in the alignment. The Fréchet distance is a bottleneck measure in that it only measures the maximum distance between aligned points. As a result, the drawback of the Fréchet distance is that it is sensitive to outliers. For such cases, a summation-based similarity measure is significantly more robust. In Figure 4.1b, we illustrate a high complexity curve, and its low complexity "simplification" that is the most similar to the original curve, under either a bottleneck or summation similarity measure. The simplified curve under the Fréchet distance is sensitive to and drawn towards its outlier points.

Continuous Dynamic Time Warping (CDTW) is a recently proposed alternative that does not exhibit the aforementioned drawbacks. It obtains the best of both worlds by combining the continuous nature of the Fréchet distance with the summation of DTW. CDTW was first introduced by Buchin [43], where it was referred to as the average Fréchet distance. CDTW has also been referred to as the summed, or integral, Fréchet distance. CDTW is similar to the Fréchet distance in that a minimum cost continuous alignment is computed between the pair of curves. The cost of a continuous alignment, under CDTW, is the integral of the distances between pairs of points in the alignment. We provide a formal definition in Section 4.2. Other definitions were also given under the name CDTW [75, 143], see Section 4.1.1.

Compared to existing popular similarity measures, CDTW is robust to both the sampling rate of the time series and to its outliers. CDTW has been used in applications where this robustness is desirable. In Brakatsoulas et al. [25], the authors applied CDTW to map-matching of vehicular location data. The authors highlight two common errors in real-life vehicular data, that is, measurement errors and sampling errors. Measurement errors result in outliers whereas sampling errors cause discrepancies in sampling rates between input curves. Their experiments show an improvement in map-matching when using CDTW instead of the Fréchet distance. In a recent paper, Brankovic et al. [27] applied CDTW to clustering of bird migration data and handwritten character data. The authors used (k, ℓ)-center and medians clustering, where each of the k clusters has a (representative) center

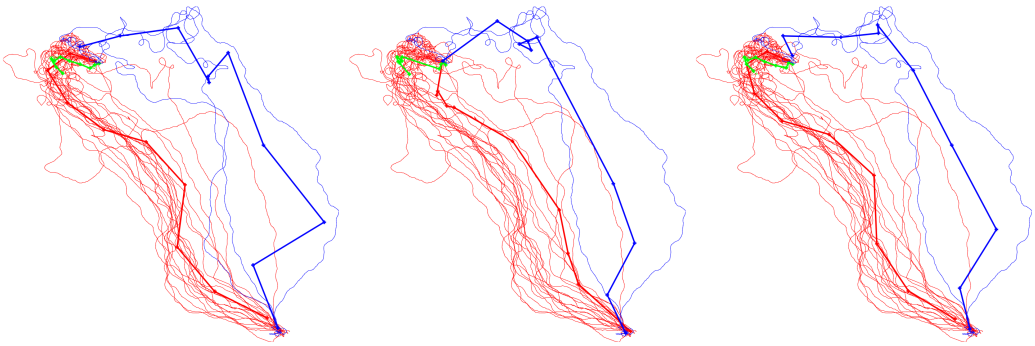


Figure 4.2: Clustering of the c17 pigeon's trajectories under the DTW (left), Fréchet (middle), and CDTW (right) distances. Figures were provided by the authors of [27].

curve of complexity at most ℓ. Low complexity center curves are used to avoid overfitting. Compared to DTW and the Fréchet distance, Brankovic et al. [27] demonstrated that clustering under CDTW produced centers that were more visually similar to the expected center curve. Under DTW, the clustering quality deteriorated for small values of ℓ, whereas under the Fréchet distance, the clustering quality deteriorated in the presence of outliers.

Brankovic et al.'s [27] clustering of a pigeon data set [125] is shown in Figure 4.2. The Fréchet distance is paired with the center objective, whereas DTW and CDTW are paired with the medians objective. Under DTW (left), the discretisation artifacts are visible. The blue center curve is jagged and visually dissimilar to its associated input curves. Under the Fréchet distance (middle), the shortcoming of the bottleneck measure and objective is visible. The red center curve fails to capture the shape of its associated input curves, in particular, it misses the top-left "hook" appearing in its associated curves. Under CDTW (right), the center curves are smooth and visually similar to their associated curves.

Despite its advantages, the shortcoming of CDTW is that there is no exact algorithm for computing it in polynomial time. Heuristics were used to compute CDTW in the map-matching [25] and clustering [27] experiments. Maheshwari et al. [124] provided a $(1+\varepsilon)$-approximation algorithm for CDTW in $\mathcal{O}(\zeta^4 n^4/\varepsilon^2)$ time, for curves of complexity n and spread ζ, where the spread is the ratio between the maximum and minimum interpoint distances. Existing heuristic and approximation methods [25, 27, 124] use a sampled grid on top of the dynamic program for CDTW, introducing an inherent error that depends on the fineness of the sampled grid, which is reflected in the dependency on ζ in [124].

In this work, we present the first exact algorithm for computing CDTW for one-dimensional curves. Our algorithm runs in time $\mathcal{O}(n^5)$ for a pair of one-dimensional curves, each with complexity at most n. Unlike previous approaches, we avoid using a sampled grid and instead devise a propagation method that solves the dynamic program for CDTW exactly. In our propagation method, the main difficulty lies in bounding the total complexity of our propagated functions. Showing that CDTW can be computed in polynomial time fosters hope for faster polynomial time algorithms, which would add CDTW to the list of practical similarity measures for curves.

4.1.1 Related work

Algorithms for computing popular similarity measures, such as DTW and the Fréchet distance, are well studied. Vintsyuk [156] proposed Dynamic Time Warping as a similarity measure for time series, and provided a simple dynamic programming algorithm for computing the DTW distance that runs in $\mathcal{O}(n^2)$ time, see also [21]. Gold and Sharir [93] improved the upper bound for computing DTW to $\mathcal{O}(n^2/\log\log n)$. For the Fréchet distance, Alt and Godau [12] proposed an $\mathcal{O}(n^2 \log n)$ time algorithm for computing the Fréchet distance between a pair of curves. Buchin et al. [37] improved the upper bound for computing the Fréchet distance to $\mathcal{O}(n^2\sqrt{\log n}(\log\log n)^{3/2})$. Assuming SETH, it has been shown that there is no strongly subquadratic time algorithm for computing the Fréchet distance or DTW [2, 28, 30, 33, 42].

Our definition of CDTW was originally proposed by Buchin [43], and has since been used in several experimental works [25, 27]. We give Buchin's [43] definition formally in Section 4.2. Other definitions under the name CDTW have also been considered. We briefly describe the main difference between these definitions and the one used in this chapter.

To the best of our knowledge, the first continuous version of DTW was by Serra and Berthod [143]. The same definition was later used by Munich and Perona [131]. Although a continuous curve is used in their definition, the cost of the matching is still a discrete summation of distances to sampled points. Our definition uses a continuous summation (i.e. integration) of distances between all points on the curves, and therefore, is more robust to discrepancies in sampling rate. Efrat et al. [75] proposed a continuous version of DTW that uses integration. However, their integral is defined in a significantly different way to ours. Their formulation minimises the change of the alignment and not the distance between aligned points. Thus, their measure is translational invariant and designed to compare the shapes of curves irrespective of their absolute positions in space.

4.2 Preliminaries

We use $[n]$ to denote the set $\{1, \ldots, n\}$. To continuously measure the similarity of time series, we linearly interpolate between sampled points to obtain a one-dimensional polygonal curve. A one-dimensional polygonal curve P of complexity n is given by a sequence of vertices, $p_1, \ldots, p_n \in \mathbb{R}$, connected in order by line segments. Furthermore, let $||\cdot||$ be the norm in the one-dimensional space $\mathbb{R}$. In higher dimensions, the Euclidean $\mathcal{L}_2$ norm is the most commonly used norm, but other norms such as $\mathcal{L}_1$ and $\mathcal{L}_\infty$ may be used.

Consider a pair of one-dimensional polygonal curves $P = p_1, \ldots, p_n$ and $Q = q_1, \ldots, q_m$. Let $\Delta(n, m)$ be the set of all sequences of pairs of integers $(x_1, y_1), \ldots, (x_k, y_k)$ satisfying $(x_1, y_1) = (1, 1)$, $(x_k, y_k) = (n, m)$ and $(x_{i+1}, y_{i+1}) \in \{(x_i+1, y_i), (x_i, y_i+1), (x_i+1, y_i+1)\}$. The DTW distance between P and Q is defined as

$$d_{DTW}(P, Q) = \min_{\alpha \in \Delta(n,m)} \sum_{(x,y)\in\alpha} ||p_x - q_y||.$$

The discrete Fréchet distance between P and Q is defined as

$$d_{dF}(P, Q) = \min_{\alpha \in \Delta(n,m)} \max_{(x,y)\in\alpha} ||p_x - q_y||.$$

Let p and q be the total arc lengths of P and Q respectively. Define the parametrised curve $\{P(z) : z \in [0, p]\}$ to be the one-dimensional curve P parametrised by its arc length.

In other words, $P(z)$ is a piecewise linear function so that the arc length of the subcurve from $P(0)$ to $P(z)$ is z. Define $\{Q(z) : z \in [0, q]\}$ analogously. Let $\Gamma(p)$ be the set of all continuous and non-decreasing functions $\alpha : [0, 1] \to [0, p]$ satisfying $\alpha(0) = 0$ and $\alpha(1) = p$. Let $\Gamma(p, q) = \Gamma(p) \times \Gamma(q)$. The continuous Fréchet distance between P and Q is defined as

$$d_F(P, Q) = \inf_{(\alpha,\beta)\in\Gamma(p,q)} \max_{z\in[0,1]} ||P(\alpha(z)) - Q(\beta(z))||,$$

The CDTW distance between P and Q is defined as

$$d_{CDTW}(P, Q) = \inf_{(\alpha,\beta)\in\Gamma(p,q)} \int_0^1 ||P(\alpha(z)) - Q(\beta(z))|| \cdot ||\alpha'(z) + \beta'(z)|| \cdot dz.$$

For the definition of CDTW, we additionally require that α and β are differentiable. The original intuition behind $d_{CDTW}(P, Q)$ is that it is a line integral in the parameter space, which we will define in Section 4.2.1. The term $||\alpha'(z) + \beta'(z)||$ implies that we are using the $\mathcal{L}_1$ metric in the parameter space, but other norms have also been considered [116, 124].

4.2.1 Parameter space under CDTW

The parameter space under CDTW is analogous to the free space diagram under the continuous Fréchet distance. Similar to previous work [43, 116, 124], we transform the problem of computing CDTW into the problem of computing a line integral in the parameter space.

Recall that the total arc lengths of P and Q are p and q respectively. The parameter space is defined to be the rectangular region $R = [0, p] \times [0, q]$ in $\mathbb{R}^2$. The region is imbued with a metric $|| \cdot ||_R$. The $\mathcal{L}_1$, $\mathcal{L}_2$ and $\mathcal{L}_\infty$ norms have all been considered, but $\mathcal{L}_1$ is the preferred metric as it is the easiest to work with [116, 124]. At every point $(x, y) \in R$ we define the height of the point to be $h(x, y) = ||P(x) - Q(y)||$.

Next, we provide the line integral formulation of d_{CDTW}, which is the original motivation behind its definition. To make our line integral easier to work with, we parametrise our line integral path γ in terms of its $\mathcal{L}_1$ arc length in R. The following lemma is a consequence of Section 6.2 in [43]. We provide a proof sketch of the result for the sake of self-containment.

Lemma 1.

$$d_{CDTW}(P, Q) = \inf_{\gamma\in\Psi(p,q)} \int_0^{p+q} h(\gamma(z)) \cdot dz,$$

where $\Psi(p, q)$ is the set of all functions $\gamma : [0, p + q] \to R$ satisfying $\gamma(0) = (0, 0)$, $\gamma(p + q) = (p, q)$, γ is differentiable and non-decreasing in both x- and y-coordinates, and $||\gamma'(z)||_R = 1$.

Proof. Recall that the definition of $d_{CDTW}(P, Q)$ is:

$$d_{CDTW}(P, Q) = \inf_{(\alpha,\beta)\in\Gamma(p,q)} \int_0^1 ||P(\alpha(z)) - Q(\beta(z))|| \cdot ||\alpha'(z) + \beta'(z)|| \cdot dz$$

Let $\gamma = (\alpha, \beta) \in \Gamma(p, q)$. Then $\gamma(0) = (\alpha(0), \beta(0)) = (0, 0)$ and $\gamma(1) = (\alpha(1), \beta(1)) = (p, q)$. By considering $\gamma(z)$ as a point in the the parameter space R, we observe that if we vary $z \in [0, 1]$, then $\gamma(z)$ is a curve starting at $(0, 0)$, ending at (p, q), and is non-decreasing in both x- and y-coordinates. Now, consider the integral of $h(\cdot)$ along the curve γ. The mathematical expansion of the line integral $\int_\gamma h(z) \cdot dz$ is:

$$\int_\gamma h(z) \cdot dz = \int_0^1 h(\gamma(z)) \cdot ||\gamma'(z)||_R \cdot dz.$$

The expanded integral closely resembles the formula for $d_{CDTW}(P, Q)$. The first term $h(\gamma(z))$ is equal to $||P(\alpha(z)) - Q(\beta(z))||$. The second term $||\gamma'(z)||_R$ is equal to $||\alpha'(z) + \beta'(z)||$, since $||\cdot||_R$ is the $\mathcal{L}_1$ norm and because α and β are non-decreasing. Hence,

$$d_{CDTW}(P, Q) = \inf_{\gamma \in \Gamma(p,q)} \int_0^1 h(\gamma(z)) \cdot ||\gamma'(z)||_R \cdot dz.$$

We now reparametrise $\gamma \subset R$ in terms of its $\mathcal{L}_1$ arc length in R. This is the "natural parametrisation" of the curve γ. We already know that γ starts at $(0, 0)$, ends at (p, q), and is differentiable and non-decreasing in x- and y-coordinates. We let $\Psi(p, q)$ be the set of all functions that satisfy these three conditions, in addition to a fourth condition, $||\gamma'(z)||_R = 1$. Then $\alpha'(z) + \beta'(z) = 1$, as $\alpha'(z), \beta'(z) > 0$. By integrating from 0 to z, we get $\alpha(z) + \beta(z) = z$. In particular, we have $\gamma(p + q) = (p, q)$, since the curve must end at (p, q). So $z \in [0, p + q]$ is the new domain of the reparametrised curve $\gamma \in \Psi(p, q)$. Putting this together, we obtain the stated lemma. □

4.2.2 Properties of the parameter space

Before providing the algorithm for minimising our line integral, we first provide some structural insights of our parameter space $R = [0, p] \times [0, q]$. Recall that $P : [0, p] \to \mathbb{R}$ maps points on the x-axis of R to points on the one-dimensional curve P, and analogously for Q and the y-axis. Hence, each point $(x, y) \in R$ is associated with a pair of points $P(x)$ and $Q(y)$, so that the height function $h(x, y) = ||P(x) - Q(y)||$ is simply the distance between the associated pair of points. Divide the x-axis of R into $n - 1$ segments that are associated with the $n - 1$ segments $p_1p_2, \ldots, p_{n-1}p_n$ of P. Divide the y-axis of R into $m - 1$ segments analogously. In this way, we divide R into $(n - 1)(m - 1)$ cells, which we label as follows:

Definition 2 (cell)**.** *Cell (i, j) is the region of the parameter space associated with segment p_ip_{i+1} on the x-axis, and q_jq_{j+1} on the y-axis, where $i \in [n - 1]$ and $j \in [m - 1]$.*

For points (x, y) restricted to a single cell (i, j), the functions $P(x)$ and $Q(y)$ are linear. Hence, $P(x) - Q(y)$ is also linear, so $h(x, y) = ||P(x) - Q(y)||$ is a piecewise linear surface with at most two pieces. If $h(x, y)$ consists of two linear surface pieces, the boundary of these two pieces is along a segment where $h(x, y) = 0$. Since we are working with one-dimensional curves, we have two cases for the relative directions of the vectors $\overrightarrow{p_ip_{i+1}}$ and $\overrightarrow{q_jq_{j+1}}$. If the vectors are in the same direction, since $\overrightarrow{p_ip_{i+1}}$ and $\overrightarrow{q_jq_{j+1}}$ are both parametrised by their arc lengths, they must be travelling in the same direction and at the same rate. Therefore, the line satisfying $h(x, y) = 0$ has slope 1 in R. Using a similar argument, if $\overrightarrow{p_ip_{i+1}}$ and $\overrightarrow{q_jq_{j+1}}$ are in opposite direction, then the line satisfying $h(x, y) = 0$ has slope -1 in R.

The line with zero height and slope 1 will play an important role in our algorithm. We call these lines valleys. If a path γ travels along a valley, the line integral accumulates zero cost as long as it remains on the valley, since the valley has zero height.

Definition 3 (valley)**.** *In a cell, the set of points (x, y) satisfying $h(x, y) = 0$ forms a line, moreover, the line has slope 1 or -1. We call this line a valley.*

4.3 Algorithm

Our approach is a dynamic programming algorithm over the cells in the parameter space, which we defined in Section 4.2.1. To the best of our knowledge, all the existing approximation algorithms and heuristics [25, 27, 116] use a dynamic programming approach, or simply reduce the problem to a shortest path computation [124]. Next, we highlight the key difference between our approach and previous approaches.

In previous algorithms, sampling is used along cell boundaries to obtain a discrete set of grid points. Then, the optimal path between the discrete set of grid points is computed. The shortcoming of previous approaches is that an inherent error is introduced by the grid points, where the error depends on the fineness of the grid that is used.

In our algorithm, we consider all points along cell boundaries, not just a discrete subset. However, this introduces a challenge whereby we need to compute optimal paths between continuous segments of points. To overcome this obstacle, we devise a new method of propagating continuous functions across a cell. The main difficulty in analysing the running time of our algorithm lies in bounding the total complexity of the propagated continuous functions, across all cells in the dynamic program.

Our improvement over previous approaches is in many ways similar to previous algorithms for the weighted region problem [128], and the partial curve matching problem [38]. In all three problems, a line integral is minimised over a given terrain, and an optimal path is computed instead of relying on a sampled grid. However, our problem differs from that of [128] and [38] in two important ways. First, in both [128] and [38], the terrain is a piecewise constant function, whereas in our problem, the terrain is a piecewise linear function. Second, our main difficulty is bounding the complexity of the propagated functions. In [128], a different technique is used that does not propagate functions. In [38], the propagated functions are concave, piecewise linear and their complexities remain relatively low. In our algorithm, the propagated functions are piecewise quadratic and their complexities increase at a much higher, albeit bounded, rate.

The remainder of our chapter is structured as follows. In Section 4.3.1, we set up our dynamic program. In Section 4.3.2, we solve its base case. In Section 4.3.3 we solve the propagation step. In Section 4.3.4, we analyse the algorithm's running time. In Section 4.3.5, we fill in missing bounds from the running time analysis. We consider our main technical contribution to be the running time analysis in Sections 4.3.4 and 4.3.5, and their proofs in Section 4.4.

4.3.1 Dynamic program

Our dynamic program is performed with respect to the following cost function.

Definition 4 (cost function)**.** *Let $(x, y) \in R$, we define*

$$cost(x, y) = \inf_{\gamma \in \Psi(x,y)} \int_0^{x+y} h(\gamma(z)) \cdot dz.$$

Recall from Lemma 1 that $d_{CDTW}(P, Q) = \inf_{\gamma \in \Psi(p,q)} \int_0^{p+q} h(\gamma(z)) \cdot dz$, which implies that $cost(p, q) = d_{CDTW}(P, Q)$. Another way of interpreting Definition 4 is that $cost(x, y)$ is equal to $d_{CDTW}(P_x, Q_y)$, where P_x is the subcurve from $P(0)$ to $P(x)$, and Q_y is the subcurve from $Q(0)$ to $Q(y)$.

Recall from Section 4.2.2 that the parameter space is divided into $(n-1)(m-1)$ cells. Our dynamic program solves cells one at a time, starting from the bottom left cell and working towards the top right cell. A cell is considered solved if we have computed the cost of every point on the boundary of the cell. Once we solve the top right cell of R, we obtain the cost of the top right corner of R, which is $cost(p,q) = d_{CDTW}(P,Q)$, and we are done.

In the base case, we compute the cost of all points lying on the lines $x = 0$ and $y = 0$. Note that if $x = 0$ or $y = 0$, then the function $cost(x,y)$ is simply a function in terms of y or x respectively. In general, the function along any cell boundary — top, bottom, left or right — is a univariate function in terms of either x or y. We call these boundary cost functions.

Definition 5 (boundary cost function)**.** *A boundary cost function is $cost(x,y)$, but restricted to a top, bottom, left or right boundary of a cell. If it is restricted to a top or bottom (resp. left or right) boundary, the boundary cost function is univariate in terms of x (resp. y).*

In the propagation step, we use induction to solve the cell (i,j) for all $1 \leq i \leq n-1$ and $1 \leq j \leq m-1$. We assume the base case. We also assume as an inductive hypothesis that, if $i \geq 2$, then the cell $(i-1,j)$ is already solved, and if $j \geq 2$, then the cell $(i,j-1)$ is already solved. Our assumptions ensure that we receive as input the boundary cost function along the bottom and left boundaries of the cell (i,j). In other words, we use the boundary cost functions along the input boundaries to compute the boundary cost functions along the output boundaries.

Definition 6 (input/output boundary)**.** *The input boundaries of a cell are its bottom and left boundaries. The output boundaries of a cell are its top and right boundaries.*

We provide details of the base case in Section 4.3.2, and the propagation step in Section 4.3.3.

4.3.2 Base case

The base case is to compute the cost of all points along the x-axis. The y-axis can be handled analogously. Recall that $cost(x,y) = \inf_{\gamma \in \Psi(x,y)} \int_0^{x+y} h(\gamma(z)) \cdot dz$. Therefore, for points $(x,0)$ on the x-axis, we have $cost(x,0) = \inf_{\gamma \in \Psi(x,0)} \int_0^{x} h(\gamma(z)) \cdot dz$. Since $\gamma(z)$ is non-decreasing in x- and y-coordinates, and $||\gamma'(z)|| = 1$, we must have that $\gamma'(z) = (1,0)$. By integrating from 0 to z, we get $\gamma(z) = (z,0)$, which implies that $cost(x,0) = \int_0^x h(z,0) \cdot dz$.

Consider, for $1 \leq i \leq n-1$, the bottom boundary of the cell $(i,1)$. The height function $h(z)$ is a piecewise linear function with at most two pieces, so its integral $cost(x,0) = \int_0^x h(z,0) \cdot dz$ is a continuous piecewise quadratic function with at most two pieces. Similarly, since the height function along $x = 0$ is a piecewise linear function with at most $2(n-1)$ pieces, the boundary cost function along $x = 0$ is a continuous piecewise quadratic function with at most $2(n-1)$ pieces. For boundaries not necessarily on the x- or y-axis, we claim that the boundary cost function is still a continuous piecewise-quadratic function.

Lemma 7. *The boundary cost function is a continuous piecewise-quadratic function.*

We defer the proof of Lemma 7 to Section 4.4. Although the boundary cost function has at most two pieces for cell boundaries on the x- or y-axis, in the general case it may have more than two pieces. As previously stated, the main difficulty in bounding our running time analysis in Section 4.3.4 is to bound complexities of the boundary cost functions.

4.3.3 Propagation step

First, we define optimal paths in the parameter space. We use optimal paths to propagate the boundary cost functions across cells in the parameter space. Note that the second part of Definition 8 is a technical detail to ensure the uniqueness of optimal paths. Intuitively, the optimal path from s to t is the path minimising the path integral, and if there are multiple such paths, the optimal path is the one with maximum y-coordinate.

Definition 8 (optimal path)**.** *Given $t = (x_t, y_t) \in R$, its optimal path is a path $\gamma \in \Psi(x_t, y_t)$ minimising the integral $\int_0^{x_t+y_t} h(\gamma(z)) \cdot dz$. If there are multiple such curves that minimise the integral, the optimal path is the one with maximum y-coordinate (or formally, the one with maximum integral of its y-coordinates).*

Suppose t is on the output boundary of the cell (i, j). Consider the optimal path γ that starts at $(0, 0)$ and ends at t. Let s be the first point where γ enters the cell (i, j). We consider the subpath from s to t, which is entirely contained in the cell (i, j). In the next lemma, we show that the shape of the subpath from s to t is restricted, in particular, there are only three types of paths that we need to consider. A similar proof can be found in Lemma 4 of Maheswari et al. [124]. Nonetheless, due to slight differences, we provide a full proof. Specifically, the differences are that we consider one-dimensional curves, and use the $\mathcal{L}_1$ norm in parameter space to obtain a significantly stronger statement for type (A) paths.

Lemma 9. *Let t be a point on the output boundary of a cell. Let s be the first point where the optimal path to t enters the cell. There are only three types of paths from s to t:*

(A) The segments of the cell are in opposite directions. Then all paths between s and t have the same cost.

(B) The segments of the cell are in the same direction and the optimal path travels towards the valley, then along the valley, then away from the valley.

(C) The segments of the cell are in the same direction and the optimal path travels towards the valley, then away from the valley.

For an illustration of these three types of paths, see Figure 4.3.

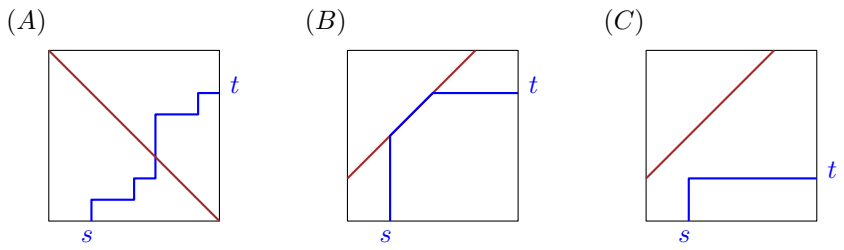


Figure 4.3: The three types of optimal paths through a cell.

Proof. We begin by summarising the main steps of the proof. Define γ_1 to be an optimal path to s, followed by either a type (A), (B) or (C) path from s to t. If the segments are in opposite directions, we use a type (A) path, whereas if the segments are in the same direction, we use either a type (B) or type (C) path. Define γ_2 to be an optimal path to s,

followed by any path from s to t. The main step is to show that $h(\gamma_1(z)) \leq h(\gamma_2(z))$, as this would imply that γ_2 is an optimal path from s to t. In fact, if the segments are in the opposite directions, we get that $h(\gamma_1(z)) = h(\gamma_2(z))$, implying that all type (A) paths from s to t have the same cost. This completes the summary of the main steps of the proof.

We start our proof by considering the case where, for this cell, the curves are in opposite directions. Let $t = (x_t, y_t)$ and let $\gamma_1 \in \Psi(x_t, y_t)$ be an optimal path to t. Let $s = (x_s, y_s)$, and let $\gamma_2 \in \Psi(x_t, y_t)$ be the concatenation of an optimal path to s and any path from s to t.

By definition, $\gamma_1(x_t+y_t) = \gamma_2(x_t+y_t) = (x_t, y_t)$, and $\gamma_1(x_s+y_s) = \gamma_2(x_s+y_s) = (x_s, y_s)$. It suffices to show that $\int_{x_s+y_s}^{x_t+y_t} h(\gamma_1(z)) \cdot dz = \int_{x_s+y_s}^{x_t+y_t} h(\gamma_2(z)) \cdot dz$, as this would imply that any path from s to t has the same cost. We will show a slightly stronger statement, that for all $x_s + y_s \leq z \leq x_t + y_t$, we have $h(\gamma_1(z)) = h(\gamma_2(z))$.

Let $\gamma_1(z) = (\alpha_1(z), \beta_1(z))$ be the x- and y-coordinates of $\gamma_1(z)$. Since $\gamma_1(z) \in \Psi(x_t, y_t)$, we have $||\gamma_1'(z)||_R = 1$ and $\alpha_1'(z), \beta_1'(z) > 0$. Therefore, $\alpha_1'(z) + \beta_1'(z) = 1$. By integrating from 0 to z, we get $\alpha_1(z) + \beta_1(z) = z$. Let $\gamma_2(z) = (\alpha_2(z), \beta_2(z))$. Then similarly, $\alpha_2(z) + \beta_2(z) = z$.

Without loss of generality, assume P is in the positive direction, and Q is in the negative direction. Recall that P is parametrised by arc length, so the arc length of the subcurve from $P(0)$ to $P(z)$ is z. Hence, for $x_s + y_s \leq z \leq x_t + y_t$, and for $i \in \{1, 2\}$, since P is in the positive direction, we have $P(\alpha_i(z)) = P(x_s) + \alpha_i(z) - x_s$. Similarly, since Q is in the negative direction, $Q(\beta_i(z)) = Q(y_s) - \beta_i(z) + y_s$. Putting this together,

$$\begin{aligned} h(\gamma_1(z)) &= ||P(\alpha_1(z)) - Q(\beta_1(z))|| \\ &= ||P(x_s) + \alpha_1(z) - x_s - Q(y_s) + \beta_1(z) - y_s|| \\ &= ||P(x_s) + \alpha_2(z) - x_s - Q(y_s) + \beta_2(z) - y_s|| \\ &= ||P(\alpha_2(z)) - Q(\beta_2(z))|| \\ &= h(\gamma_2(z)). \end{aligned}$$

This concludes the proof in the first case.

Second, we consider the case where, for this cell, the curves are in the same direction. For this case, a similar proof is given in Lemma 4 of Maheshwari et al. [124]. Nonetheless, we provide a proof for the sake of completeness. Similarly to the first case, let $t = (x_t, y_t)$ and $s = (x_s, y_s)$. Let γ_1 be the concatenation of an optimal path to s, and then either a type (B) or type (C) path from s to t. Recall from the statement of our lemma that a type (B) path is an axis-parallel path from s towards the valley, then a path along the valley, then an axis-parallel path away from the valley to t. A type (C) path is an axis-parallel path from s towards the valley, then an axis-parallel path away from the valley to t. Let $\gamma_2 \in \Psi(x_t, y_t)$ be the concatenation of an optimal path to s, then any path from s to t.

By definition, $\gamma_1(x_t+y_t) = \gamma_2(x_t+y_t) = (x_t, y_t)$, and $\gamma_1(x_s+y_s) = \gamma_2(x_s+y_s) = (x_s, y_s)$. We are required to show that $\int_{x_s+y_s}^{x_t+y_t} h(\gamma_1(z)) \cdot dz \leq \int_{x_s+y_s}^{x_t+y_t} h(\gamma_2(z)) \cdot dz$, as this would imply that γ_1 is an optimal path from s to t. We will show a slightly stronger statement, that for all $x_s + y_s \leq z \leq x_t + y_t$, we have $h(\gamma_1(z)) \leq h(\gamma_2(z))$.

For $i \in \{1, 2\}$, let $\gamma_i(z) = (\alpha_i(z), \beta_i(z))$. For the same reasons as in the first case, $\alpha_i(z) + \beta_i(z) = z$. Without loss of generality, assume both P and Q are in the positive direction. For the same reasons as in the first case, for $i \in \{1, 2\}$, we have $P(\alpha_i(z)) = P(x_s) + \alpha_i(z) - x_s$ and $Q(\beta_i(z)) = Q(y_s) + \beta_i(z) - y_s$.

For type (B) paths, if $\gamma_1(z)$ is along the valley, then $h(\gamma_1(z)) = 0$ so clearly $h(\gamma_1(z)) \leq h(\gamma_2(z))$. For both type (B) and type (C) paths, the only remaining case is where $\gamma_1(z)$ is

along an axis parallel path from s towards the valley, since other the case where $\gamma_1(z)$ is along an axis parallel path away from the valley to t can be handled analogously.

Without loss of generality, assume s is on the bottom boundary, and $\gamma_1(z)$ is along a vertical path from s to the valley. Then $\alpha_1(z) = x_s$ and $\beta_1(z) = z - \alpha_1(z) = z - x_s$. Since $\gamma_2(z)$ is monotonically increasing in the x-coordinate, we have $\alpha_2(z) \geq x_s$ and $\beta_2(z) = z - \alpha_2(z) \leq z - x_s$. Without loss of generality, assume $P(x_s) - Q(y_s) \geq 0$. Since $\gamma_1(z)$ is on the vertical path from s to the valley, but does not cross over the valley, we have $P(\alpha_1(z)) - Q(\beta_1(z)) \geq 0$. Putting this together,

$$\begin{aligned} h(\gamma_1(z)) &= P(\alpha_1(z)) - Q(\beta_1(z)) \\ &= P(x_s) + \alpha_1(z) - x_s - Q(y_s) - \beta_1(z) + y_s \\ &= P(x_s) + x_s - x_s - Q(y_s) - z + x_s + y_s \\ &\leq P(x_s) + \alpha_2(z) - x_s - Q(y_s) - \beta_2(z) - y_s \\ &= P(\alpha_2(z)) - Q(\beta_2(z)) \\ &= h(\gamma_2(z)), \end{aligned}$$

where the second last equality uses that $P(\alpha_2(z)) - Q(\beta_2(z)) \geq P(\alpha_1(z)) - Q(\beta_1(z)) \geq 0$. This concludes the proof in the second case, and we are done. □

We leverage Lemma 9 to propagate the boundary cost function from the input boundaries to the output boundaries of a cell. We provide an outline of our propagation procedure in one of the three cases, that is, for type (B) paths. These paths are the most interesting to analyse, and looking at this special case provides us with some intuition for the other cases. For type (B) paths, we compute the cost function along the output boundary in three consecutive steps. We first list the steps, then we describe the steps in detail.

1. We compute the cost function along the valley in a restricted sense.
2. We compute the cost function along the valley in general.
3. We compute the cost function along the output boundary.

In the first step, we restrict our attention only to paths that travel from the input boundary towards the valley. This is the first segment in the type (B) path as defined in Lemma 9. We call this first segment a type $(B1)$ path, see Figure 4.4. Define the type $(B1)$ cost function to be the cost function along the valley if we can only use type $(B1)$ paths from the input boundary to the valley. The type $(B1)$ cost function is simply the cost function along the bottom or left boundary plus the integral of the height function along the type $(B1)$ path. The height function along the type $(B1)$ path is a linear function, so the integral is a quadratic function. To obtain the type $(B1)$ cost function, we add the quadratic function for the type $(B1)$ path to the cost function along an input boundary. We combine the type $(B1)$ cost functions along the bottom and the left boundaries by taking their lower envelope.

In the second step, we compute the cost function along the valley in general. It suffices to consider paths that travel from the input boundary towards the valley, and then travel along the valley. This path is the first two segments in a type (B) path as defined in Lemma 9. We call these first two segments a type $(B2)$ path, see Figure 4.4. Since the height function is zero along the valley, if we can reach a valley point with a particular cost with a type $(B1)$ path, then we can reach all points on the valley above and to the right of it with a type $(B2)$ path with the same cost. Therefore, the type $(B2)$ cost function is

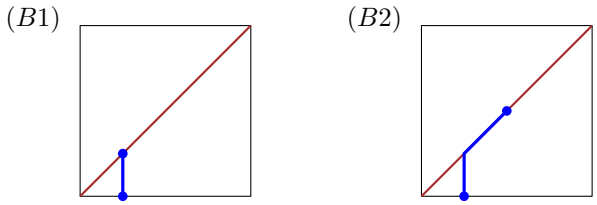


Figure 4.4: The type $(B1)$ and type $(B2)$ paths from the bottom boundary to the valley.

the cumulative minimum of the type $(B1)$ cost function, see Figure 4.5. Note that the type $(B2)$ cost function may have more quadratic pieces than the type $(B1)$ cost function. For example, in Figure 4.5, the type $(B2)$ cost function has twice as many quadratic pieces as the type $(B1)$ cost function, since each quadratic piece in the type $(B1)$ cost function splits into two quadratic pieces in the type $(B2)$ cost function — the original quadratic piece plus an additional horizontal piece.

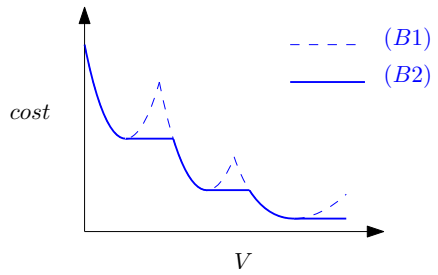


Figure 4.5: The type $(B2)$ cost function plotted over its position along the valley V. The type $(B2)$ cost function is the cumulative minimum of the type $(B1)$ cost function.

In the third step, we compute the cost function along the output boundary, given the type $(B2)$ cost function along the valley. A type (B) path is a type $(B2)$ path appended with a horizontal or vertical path from the valley to the boundary. The height function of the appended path is a linear function, so its integral is a quadratic function. We add this quadratic function to the type $(B2)$ function along the valley to obtain the output function. This completes the description of the propagation step in the type (B) paths case.

Using a similar approach, we can compute the cost function along the output boundary in the type (A) and type (C) paths as well. The propagation procedure differs slightly for each of the three path types, for details see Section 4.4. Recall that due to the second step of the type (B) propagation, each quadratic piece along the input boundary may propagate to up to two pieces along the output boundary. In general, we claim that each quadratic piece along the input boundary propagates to at most a constant number of pieces along the output boundary. Moreover, given a single input quadratic piece, this constant number of output quadratic pieces can be computed in constant time.

Lemma 10. *Each quadratic piece in the input boundary cost function propagates to at most a constant number of pieces along the output boundary. Propagating a quadratic piece takes constant time.*

We defer the proof of Lemma 10 to Section 4.4. We can now state our propagation step in general. Divide the input boundaries into segments, so that for each segments, the cost

function along that segments is a single quadratic piece. Apply Lemma 10 to a segments to compute in constant time a piecewise quadratic cost function along the output boundary. Apply this process to all segments to obtain a set of piecewise quadratic cost functions along the output boundary. Combine these cost functions by taking their lower envelope. Return this lower envelope as the boundary cost function along the output boundary. This completes the statement of our propagation step. Its correctness follows from construction.

4.3.4 Running time analysis

We start the section with a useful lemma. Essentially the same result is stated without proof as Observation 3.3 in [38]. For the sake of completeness, we provide a proof.

Lemma 11. *Let γ_1, γ_2 be two optimal paths. These paths cannot cross, i.e., there are no z_1, z_2 such that $\gamma_1(z_1)$ is below $\gamma_2(z_1)$ and $\gamma_1(z_2)$ is above $\gamma_2(z_2)$.*

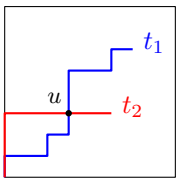


Figure 4.6: A pair of optimal paths that cross at a point u.

Proof. Suppose for sake of contradiction that there exists a pair of optimal paths, γ_1 to t_1 and γ_2 to t_2, that cross over at a point u, see Figure 4.6. Moreover, assume that u is the first such crossover point, i.e., u is the last point such that γ_1 and γ_2 are non-crossing until u. Without loss of generality, before the intersection point u, the path γ_1 is below γ_2, and after the intersection point u, the path γ_1 is above γ_2. Since γ_1 is an optimal path to t_1, the portion of γ_1 up to u must be the optimal path to u. Similarly, the portion of γ_2 up to u must be the optimal path to u. Since optimal paths are unique, we obtain that γ_1 and γ_2 are identical up to the first crossover point u, contradicting the fact that γ_1 and γ_2 cross. □

Define N to be the total number of quadratic pieces in the boundary cost functions over all boundaries of all cells. We will show that the running time of our algorithm is $\mathcal{O}(N)$.

Lemma 12. *The running time of our dynamic programming algorithm is $\mathcal{O}(N)$.*

Proof. The running time of the dynamic program is dominated by the propagation step. Let $I_{i,j}$ denote the input boundaries of the cell (i, j). Let $|I_{i,j}|$ denote the number of quadratic functions in the input boundary cost function. By Lemma 10, each piece only propagates to a constant number of new pieces along the output boundary, and these pieces can be computed in constant time. The final piecewise quadratic function is the lower envelope of all the new pieces, of which there are $\mathcal{O}(|I_{i,j}|)$ many.

We use Lemma 11 to speed up the computation of the lower envelope, so that this step takes only $\mathcal{O}(|I_{i,j}|)$ time. Since optimal paths do not cross, it implies that the new pieces along the output boundary appear in the same order as their input pieces. We perform the propagation in order of the input pieces. We maintain the lower envelope of the new pieces

in a stack. For each newly propagated piece, we remove the suffix that is dominated by the new piece and then add the new piece to the stack. Since each quadratic piece can be added to the stack at most once, and removed from the stack at most once, the entire operation takes $\mathcal{O}(|I_{i,j}|)$ time. Summing over all cells, we obtain an overall running time of $\mathcal{O}(N)$. □

Note that Lemma 12 does not yet guarantee that our algorithm runs in polynomial time as we additionally need to bound N by a polynomial. Lemma 10 is of limited help. The lemma states that each piece on the input boundary propagates to at most a constant number of pieces on the output boundary. Recall that in Section 4.3.3, we illustrated a type (B) path that resulted in an output boundary having twice as many quadratic pieces as its input boundary. The doubling occurred in the second step of the propagation of type (B) paths, see Figure 4.5. If this doubling behaviour were to occur for all our cells in our dynamic program, we would get up to $N = \Omega(2^{n+m})$ quadratic pieces in the worst case, where n and m are the complexities of the polygonal curves P and Q. To obtain a polynomial running time, we show that although this doubling behaviour may occur, it does not occur *too often*.

4.3.5 Bounding the cost function's complexity

Our bound comes in two parts. First, we subdivide the boundaries in the parameter space into subsegments and show, in Lemma 13, that there are $\mathcal{O}((n+m)^3)$ subsegments in total. Second, in Lemma 15, we show that each subsegment has at most $\mathcal{O}((n+m)^2)$ quadratic pieces. Putting this together in Theorem 16 gives $N = \mathcal{O}((n+m)^5)$.

We first define the $\mathcal{O}((n+m)^3)$ subsegments. The intuition behind the subsegments is that for any two points on the subsegment, the optimal path to either of those two points is structurally similar. We can deform one of the optimal paths to the other without passing through any cell corner, or any points where a valley meets a boundary.

Formally, define A_k to be the union of the input boundaries of the cells (i,j) such that $i+j=k$. Alternatively, A_k is the union of the output boundaries of the cells (i,j) such that $i+j+1=k$. Next, construct the partition $A_k \coloneqq \{A_{k,1}, A_{k,2}, \ldots, A_{k,L}\}$ of A_k into subsegments. Define the subsegment $A_{k,\ell}$ to be the segment between the ℓ^{th} and $(\ell+1)^{th}$ *critical point* along A_k. We define a critical point to be either *(i)* a cell corner, *(ii)* a point where the valley meets the boundary, or *(iii)* a point where the optimal path switches from passing through a subsegment $A_{k-1,\ell'}$ to a different subsegment $A_{k-1,\ell''}$.

Let $|A_k|$ denote the number of piecewise quadratic cost functions $A_{k,\ell}$ along A_k. Let $|A_{k,\ell}|$ denote the number of pieces in the piecewise quadratic cost function along the subsegment $A_{k,\ell}$. Thus, we can rewrite the total number of quadratic functions N as:

$$N = \sum_{k=2}^{n+m-1} \sum_{\ell=1}^{|A_k|} |A_{k,\ell}|.$$

We first show that the number of subsegments $|A_k|$ is bounded by $\mathcal{O}(k^2)$ and then proceed to show that $|A_{k,\ell}|$ is bounded by $\mathcal{O}(k^2)$ for all k, ℓ.

Lemma 13. *For any $k \in [n+m]$, we have $|A_k| \leq 2k^2$.*

Proof. We prove the lemma by induction. Since the cell $(1,1)$ has at most one valley, and since the input boundary A_2 has one cell corner, we have $|A_2| \leq 3$. For the inductive step, note that there are at most $2k$ cell corners on A_k, and there are at most k points where a

valley meets a boundary on A_k. By the inductive hypothesis, there are at most $2(k-1)^2$ subsegments on A_{k-1}. And as optimal paths do not cross by Lemma 11, each subsegment of A_{k-1} contributes at most once to the optimal path switching from one subsegment to a different one on A_k. Thus, for $k \geq 3$, we obtain $|A_k| \leq 2(k-1)^2 + 2k + k + 1 = 2k^2 - k + 3 \leq 2k^2$. □

Next, we show that $|A_{k,\ell}|$ is bounded by $\mathcal{O}(k^2)$ for all k, ℓ. We proceed by induction. Recall that, due to the third type of critical point, all optimal paths to $A_{k,\ell}$ pass through the same subsegment of A_{k-1}, namely $A_{k-1,\ell'}$ for some ℓ'. Our approach is to assume the inductive hypothesis for $|A_{k-1,\ell'}|$, and bound $|A_{k,\ell}|$ relative to $|A_{k-1,\ell'}|$. We already have a bound of this form, specifically, Lemma 10 implies that $|A_{k,\ell}| \leq c \cdot |A_{k-1,\ell'}|$, for some constant $c > 1$. Unfortunately, this bound does not rule out an exponential growth in the cost function complexity. We instead prove the following improved bound:

Lemma 14. *Let $|A_{k,\ell}|$ be a subsegment on A_k, and suppose all optimal paths to $|A_{k,\ell}|$ pass through subsegment $|A_{k-1,\ell'}|$ on A_{k-1}. Then*

$$\begin{aligned} |A_{k,\ell}| &\leq |A_{k-1,\ell'}| + D(A_{k-1,\ell'}) + 1, \\ D(A_{k,\ell}) &\leq D(A_{k-1,\ell'}) + 1, \end{aligned}$$

where $D(\cdot)$ counts, for a given subsegment, the number of distinct pairs (a, b) over all quadratics $ax^2 + bx + c$ in the boundary cost function for that subsegment.

We defer the proof of Lemma 14 to Section 4.4. The lemma obtains a polynomial bound on the growth of the number of quadratic pieces by showing, along the way, a polynomial bound on the growth of the number of distinct (a, b) pairs over the quadratics $ax^2 + bx + c$.

As we consider this lemma to be one of the main technical contributions of the chapter, we will briefly outline its intuition. It is helpful for us to revisit the doubling behaviour of type (B) paths. Recall that in our example in Figure 4.5, we may have $|A_{k,\ell}| = 2|A_{k-1,\ell'}|$. This doubling behaviour does not contradict Lemma 14, so long as all quadratic functions along $A_{k-1,\ell'}$ have distinct (a, b) pairs. In fact, for $|A_{k,\ell}| = 2|A_{k-1,\ell'}|$ to occur, each quadratic function in $|A_{k-1,\ell'}|$ must create a new horizontal piece in the cumulative minimum step. But for any two quadratic functions with the same (a, b) pair, only one of them can to create a new horizontal piece, since the horizontal piece starts at the x-coordinate $-\frac{b}{2a}$. Therefore, we must have had that all quadratic functions along $A_{k-1,\ell'}$ have distinct (a, b) pairs. In Section 4.4, we generalise this argument and prove $|A_{k,\ell}| \leq |A_{k-1,\ell'}| + D(A_{k-1,\ell'}) + 1$.

We perform a similar analysis in the special case of type (B) paths to give the intuition behind $D(A_{k,\ell}) \leq D(A_{k-1,\ell'}) + 1$. For type (B) paths, the number of distinct (a, b) pairs changes only in the cumulative minimum step. All pieces along $A_{k,\ell}$ can either be mapped to a piece along $A_{k-1,\ell'}$, or it is a new horizontal piece. However, all new horizontal pieces have an (a, b) pair of $(0, 0)$, so the number of distinct (a, b) pairs increases by only one. For the full proof of Lemma 14 for all three path types, refer to Section 4.4.

With Lemma 14 in mind, we can now prove a bound on $|A_{k,\ell}|$ by induction.

Lemma 15. *For any $k \in [n + m]$ and $A_{k,\ell} \in A_k$ we have $|A_{k,\ell}| \leq k^2$.*

Proof. Note that in the base case $D(A_{2,\ell}) \leq 2$ and $|A_{2,\ell}| \leq 4$ for any $A_{2,\ell} \in A_2$. By Lemma 14, we get $D(A_{k,\ell}) \leq D(A_{k-1,\ell'}) + 1$, for some subsegment $A_{k-1,\ell'}$ on A_{k-1}. By a simple induction, we get $D(A_{k,\ell}) \leq k$ for any $k \in [n + m]$. Similarly, assuming $|A_{k-1,\ell}| \leq (k-1)^2$ for any $A_{k-1,\ell} \in A_{k-1}$, we use Lemma 14 to inductively obtain $|A_{k,\ell}| \leq |A_{k-1,\ell'}| + D(A_{k-1,\ell'}) + 1 \leq |A_{k-1,\ell'}| + k \leq (k-1)^2 + k - 1 + 1 \leq k^2$ for any $A_{k,\ell} \in A_k$. □

Using our lemmas, we can finally bound N, and thereby the overall running time.

Theorem 16. *The Continuous Dynamic Time Warping distance between two 1-dimensional polygonal curves of length n and m, respectively, can be computed in time $\mathcal{O}((n+m)^5)$.*

Proof. Using Lemmas 13 and 15, we have

$$N = \sum_{k=2}^{n+m-1} \sum_{\ell=1}^{|A_k|} |A_{k,\ell}| \leq \sum_{k=2}^{n+m} \sum_{\ell=1}^{|A_k|} k^2 \leq \sum_{k=2}^{n+m} 2k^4 \leq 2(n+m)^5.$$

Thus, the overall running time of our algorithm is $\mathcal{O}((n+m)^5)$, by Lemma 12. □

4.4 Proofs of Lemmas 7, 10 and 14

In this section, we provide proofs for the following three lemmas.

Lemma 7. *The boundary cost function is a continuous piecewise-quadratic function.*

Lemma 10. *Each quadratic piece in the input boundary cost function propagates to at most a constant number of pieces along the output boundary. Propagating a quadratic piece takes constant time.*

Lemma 14. *Let $|A_{k,\ell}|$ be a subsegment on A_k, and suppose all optimal paths to $|A_{k,\ell}|$ pass through subsegment $|A_{k-1,\ell'}|$ on A_{k-1}. Then*

$$\begin{aligned} |A_{k,\ell}| &\leq |A_{k-1,\ell'}| + D(A_{k-1,\ell'}) + 1, \\ D(A_{k,\ell}) &\leq D(A_{k-1,\ell'}) + 1, \end{aligned}$$

where $D(\cdot)$ counts, for a given subsegment, the number of distinct pairs (a, b) over all quadratics $ax^2 + bx + c$ in the boundary cost function for that subsegment.

Our approach is to use induction to prove these three lemmas. Our inductive approach is to prove the three lemmas simultaneously, in that the inductive hypothesis for one lemma is required to prove the other lemmas. In fact, for this inductive approach to work, we require a stronger version of Lemma 7, which we state as Lemma 17.

Lemma 17. *The cost along a subsegment $A_{k,\ell}$ is a continuous piecewise quadratic function. Moreover, for every point t on the piecewise quadratic function, the left derivative at t is greater than or equal to the right derivative at t.*

The remainder of this chapter is dedicated to prove these lemmas required for the correctness of our algorithm and its running time. In Section 4.3.2, we proved Lemma 17 for the base case where $k = 2$. Lemmas 10 and 14 are clearly true for $k = 2$. For the inductive case we require a case analysis. Recall Lemma 9.

Lemma 9. *Let t be a point on the output boundary of a cell. Let s be the first point where the optimal path to t enters the cell. There are only three types of paths from s to t:*

(A) The segments of the cell are in opposite directions. Then all paths between s and t have the same cost.

(B) The segments of the cell are in the same direction and the optimal path travels towards the valley, then along the valley, then away from the valley.

(C) The segments of the cell are in the same direction and the optimal path travels towards the valley, then away from the valley.

For an illustration of these three types of paths, see Figure 4.3.

Using Lemma 9, we divide our inductive step into separate case analyses for each of the three types of optimal paths:

	Type (A)	Type (B)	Type (C)
Lemma 17	Lemma 21	Lemma 28	Lemma 31
Lemma 10	Lemma 22	Lemma 29	Lemma 32
Lemma 14	Lemma 23	Lemma 30	Lemma 33

For proving our lemmas, it will be useful to us to introduce the following notation.

Definition 18 (parent)**.** *If all optimal paths of $A_{k,\ell}$ pass through $A_{k-1,\ell'}$, then the subsegment $A_{k-1,\ell'}$ is called the parent of $A_{k,\ell}$.*

4.4.1 Type *(A)* paths

In this section we show that Lemmas 10, 14, and 17 hold for the output boundary cost functions of type (A) paths. We assume the inductive hypothesis, that Lemmas 10, 14, and 17 hold for the input boundary cost function. First we state two observations that help simplify our proofs.

Observation 19. *Let $A_{k-1,\ell'}$ be the parent of $A_{k,\ell}$. Suppose there exists a type (A) path from $A_{k-1,\ell'}$ to $A_{k,\ell}$. Then all paths from $A_{k-1,\ell'}$ to $A_{k,\ell}$ are type (A) paths.*

Proof. Type (A) paths only exist between segments that are in opposite directions. Type (B) or (C) paths only exist between segments that are in the same direction. If not all paths were type (A) paths, we must have switched directions, which means there is a cell corner, which is a critical point. But in a subsegment there are no critical points. □

Observation 20. *All optimal paths in a type (A) cell can be replaced by a single horizontal segment or a single vertical segment, possibly changing the starting point, but without changing the cost at the end point.*

Proof. Consider any optimal path through a type (A) cell and without loss of generality assume that it starts on the bottom boundary. By Lemma 9 we can replace a type (A) path by any other path without changing the cost. In particular, we can replace it by the path that first goes horizontally along the bottom boundary and then vertically until the end point. However, the cost of this path is at least as high as the exclusively vertical path that starts at the point where we leave the bottom boundary. Thus, we can only consider paths that consist of a single horizontal or vertical segment, without changing the cost on the outputs of the cell. □

We now show Lemma 17 for type (A) paths. Note that by Observation 19 we already know that either all paths are of type (A) or none are. Hence, we only have to consider the former case.

Lemma 21. *Let $A_{k-1,\ell'}$ be the parent of $A_{k,\ell}$. Suppose all paths from $A_{k-1,\ell'}$ to $A_{k,\ell}$ are type (A) paths. Then the cost along $A_{k,\ell}$ is a continuous piecewise quadratic function. Moreover, for every point t on the piecewise quadratic function, the left derivative at t is greater than or equal to the right derivative at t.*

Proof. The cost along $A_{k,\ell}$ is equal to the cost along $A_{k-1,\ell'}$ plus the cost of an optimal type (A) path. By Observation 20, all optimal paths that we have to consider are either a single vertical or horizontal segment. The cost of a vertical/horizontal path inside a subsegment is a quadratic with respect to its x-coordinate/y-coordinate. This quadratic cost may change at a critical point, but since there are no critical points inside a subsegment, the cost is simply a quadratic. Summing a function with a quadratic preserves the fact that the function is a piecewise quadratic function. Summing a function with a quadratic also preserves the property that for every point t on the piecewise quadratic function, the left derivative at t is greater than or equal to the right derivative at t. □

We now show Lemma 10 for type (A) paths. Again, by Observation 19, we can assume that all paths are of type (A).

Lemma 22. *Let $A_{k-1,\ell'}$ be the parent of $A_{k,\ell}$. Suppose all paths starting from $A_{k-1,\ell'}$ are type (A) paths. If we propagate a single quadratic piece of $A_{k-1,\ell'}$ to the next level A_k, it is a piecewise quadratic function with at most a constant number of pieces. Moreover, this propagation step takes only constant time.*

Proof. First we consider propagating $A_{k-1,\ell'}$ to the opposite side. By Observation 20, we can assume that this happens along a vertical or horizontal segment. These paths have piecewise quadratic cost with at most two pieces. Computing the cost from $A_{k-1,\ell'}$ to the opposite side takes constant time. In the case that $A_{k-1,\ell'}$ contains the top left corner of its cell, we need to propagate its cost along the top boundary. This cost has at most two pieces and takes constant time to compute. In the case that $A_{k-1,\ell'}$ contains the bottom right corner of its cell, we need to propagate its cost along the right boundary. This cost has at most two pieces and takes constant time to compute. Hence, the cost along the next level A_k has a constant number of pieces and the propagation can be computed in constant time. □

Finally, we show Lemma 14 for type (A) paths.

Lemma 23. *Let $A_{k-1,\ell'}$ be the parent of $A_{k,\ell}$. Suppose all paths from $A_{k-1,\ell'}$ to $A_{k,\ell}$ are type (A) paths, then*

$$D(A_{k,\ell}) \leq D(A_{k-1,\ell'}),$$

$$|A_{k,\ell}| \leq |A_{k-1,\ell'}|.$$

Proof. Without loss of generality $A_{k-1,\ell'}$ is on the bottom boundary. If $A_{k,\ell}$ is on the top boundary, then its cost is the same as the one on $A_{k-1,\ell'}$ plus a quadratic, and we have $D(A_{k,\ell}) \leq D(A_{k-1,\ell'})$ and $|A_{k,\ell}| \leq |A_{k-1,\ell'}|$. If $A_{k,\ell}$ is on the right boundary, then all vertical paths pass through the bottom right corner and thus there is a single path with minimum cost. This implies $D(A_{k,\ell}) = |A_{k,\ell}| = 1$. □

4.4.2 Separating type *(B)* and *(C)* paths

Similarly to Section 4.4.1, it will be useful to show Lemmas 10, 14, and 17, for output boundary cost functions of only type (B) or type (C) paths. It will be useful to be able to consider two types of paths separately. In order to do this, we must show that we can further divide our subsegments so that, for each subsegment on the output boundary, all optimal paths to the subsegments are type (B) paths or that all optimal paths are type (C) paths.

Lemma 24. *Suppose we add a critical point in $A_{k,\ell}$ where the optimal path from $A_{k-1,\ell'}$ to $A_{k,\ell}$ changes between a type (B) path and a type (C) path. Then at most two critical points are added to $A_{k,\ell}$.*

Proof. Suppose there were at least three such critical points. Then the path types must alternate, between B, C, B, C. In particular, there must exist, along the output boundary, paths of type B, C, B. The two outer type (B) paths must visit the valley. However, the inner type (C) path is sandwiched between the type (B) paths, since by Lemma 11, no optimal paths may cross. Therefore, the type (C) path must also visit the valley, which contradicts the definition of the type (C) path. □

4.4.3 Type *(B)* paths

As a consequence of the additional critical points in Lemma 24, we obtain the following fact.

Observation 25. *Let $A_{k-1,\ell'}$ be the parent of $A_{k,\ell}$. Suppose there exists a type (B) path from $A_{k-1,\ell'}$ to $A_{k,\ell}$. Then all paths from $A_{k-1,\ell'}$ to $A_{k,\ell}$ are type (B) paths.*

Recall that V is the valley of our cell in the parameter space, if one exists.

Definition 26. *Let the type $(B1)$ function be the cost function along V if we only allow an axis parallel path from $A_{k-1,\ell'}$ to V. Let the type $(B2)$ function be the cost function along V if we allow an axis parallel path from $A_{k-1,\ell'}$ to V followed by a path of slope 1 along V.*

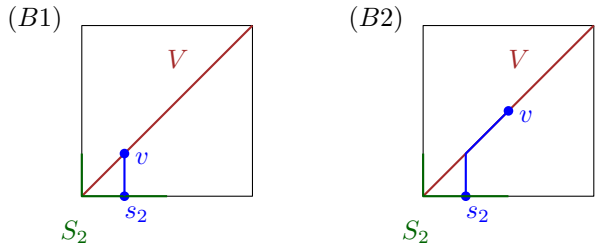


Figure 4.7: The type $(B1)$ and type $(B2)$ optimal path from $s_2 \in S_2$ to $v \in V$.

We prove an observation, that the type $(B2)$ cost function is indeed the cumulative minimum of the type $(B1)$ cost function.

Observation 27. *The type $(B2)$ cost function is the cumulative minimum function of the type $(B1)$ cost function.*

Proof. We observe that a type $(B2)$ paths is a type $(B1)$ path appended with a path along the valley. Since the height function is zero along the valley, if we can reach a valley point for a particular cost with a type $(B1)$ path, then we can reach all points on the valley above and to the right of it with a type $(B2)$ path. So the type $(B2)$ cost function is the cumulative minimum of the type $(B1)$ cost function, see Figure 4.5. □

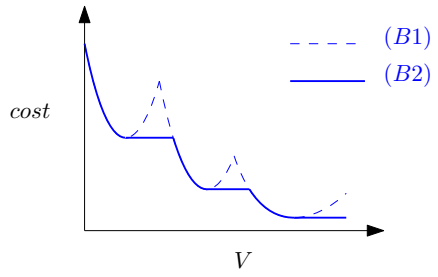


Figure 4.8: The type $(B2)$ cost function.

Finally, we can show the main lemmas of this section. Lemmas 28, 29 and 30 are the special cases of Lemmas 17, 10 and 14 in the type (B) paths case.

Lemma 28. *Let $A_{k-1,\ell'}$ be the parent of $A_{k,\ell}$. Suppose all paths from $A_{k-1,\ell'}$ to $A_{k,\ell}$ are type (B) paths. Then the cost along $A_{k,\ell}$ is a continuous piecewise quadratic function. Moreover, for every point t on the piecewise quadratic function, the left derivative at t is greater than or equal to the right derivative at t.*

Proof. Three operations are performed to get from the cost function along $A_{k-1,\ell'}$ to the cost function along $A_{k,\ell}$. From $A_{k-1,\ell'}$ to the type $(B1)$ cost function we add a quadratic function. From type $(B1)$ to type $(B2)$, we take the cumulative minimum, as shown in Figure 4.7. From the type $(B2)$ cost function to $A_{k,\ell}$, we add a quadratic function. All three operations preserve the fact that the cost function is a piecewise quadratic function. All three operations preserve the property that, for every point t on the piecewise quadratic function, the left derivative at t is greater than or equal to the right derivative at t. □

Lemma 29. *Let $A_{k-1,\ell'}$ be the parent of $A_{k,\ell}$. Suppose all paths from $A_{k-1,\ell'}$ are type (B) paths. If we propagate a single quadratic piece of $A_{k-1,\ell'}$ to the next level A_k, it is a piecewise quadratic function with at most a constant number of pieces. Moreover, this propagation step takes only constant time.*

Proof. Computing the cost function along V consists of adding a quadratic function to the cost function along $A_{k-1,\ell}$, and taking the cumulative minimum. This cost function has at most two pieces and can be computed in constant time. Adding a quadratic function to the cost function along V to yield the cost function along A_k takes constant time. □

Lemma 30. *Let $A_{k-1,\ell'}$ be the parent of $A_{k,\ell}$. Suppose all paths from $A_{k-1,\ell'}$ to $A_{k,\ell}$ are type (B) paths. Then*

$$D(A_{k,\ell}) \leq D(A_{k-1,\ell'}) + 1,$$

$$|A_{k,\ell}| \leq |A_{k-1,\ell'}| + D(A_{k-1,\ell'}) + 1.$$

Proof. Three operations are performed to get from the cost function along $A_{k-1,\ell'}$ to the cost function along $A_{k,\ell}$. The first and third are to add a quadratic function, whereas the second is to take the cumulative minimum.

The first and third operations do not change the number of distinct pairs (a, b). The second operation only adds horizontal functions, so the number of distinct pairs (a, b) increases by at most once. Putting this together yields $D(A_{k,\ell}) \leq D(A_{k-1,\ell'}) + 1$.

The first and third operations do not change the number of pieces in the piecewise quadratic function. The second operation adds a horizontal function at each of the local minima. The local minima of a piecewise quadratic function can occur at the endpoints of the function, at points where both the left and right derivatives are zero, or at points where the left derivative is strictly less than the right derivative. There are at most two endpoints, and we can only add a horizontal function to one of them. For each distinct pair (a, b), there is at most one local minima where both the left and right derivatives are zero, since this local minima only occurs at the x-coordinate given by $x = -\frac{b}{2a}$. By Lemma 28, we do not have any points where the left derivative is strictly less than the right derivative. Putting this together, we add at most $D(A_{k-1,\ell'}) + 1$ horizontal functions at the local minima of the piecewise quadratic function. This gives $|A_{k,\ell}| \leq |A_{k-1,\ell'}| + D(A_{k-1,\ell'}) + 1$, as required. □

4.4.4 Type *(C)* paths

Recall that we require the following three lemmas for type (C) paths.

Lemma 31. *Let $A_{k-1,\ell'}$ be the parent of $A_{k,\ell}$. Suppose all paths from $A_{k-1,\ell'}$ to $A_{k,\ell}$ are type (C) paths. Then the cost along $A_{k,\ell}$ is a continuous piecewise quadratic function. Moreover, for every point t on the piecewise quadratic function, the left derivative at t is greater than or equal to the right derivative at t.*

Lemma 32. *Let $A_{k-1,\ell'}$ be the parent of $A_{k,\ell}$. Suppose all paths from $A_{k-1,\ell'}$ are type (C) paths. If we propagate a single quadratic piece of $A_{k-1,\ell'}$ to the next level A_k, it is a piecewise quadratic function with at most a constant number of pieces. Moreover, this propagation step takes only constant time.*

Lemma 33. *Let $A_{k-1,\ell'}$ be the parent of $A_{k,\ell}$. Suppose all paths from $A_{k-1,\ell'}$ to $A_{k,\ell}$ are type (C) paths. Then*

$$D(A_{k,\ell}) \leq D(A_{k-1,\ell'}),$$
$$|A_{k,\ell}| \leq |A_{k-1,\ell'}|.$$

We prove these lemmas again by dividing into subcases. Our subcases are defined as follows.

Definition 34. *Let type $(C1)$, $(C2)$ and $(C3)$ paths be type (C) paths where the starting point and the ending point are on the left and right, bottom and right, or bottom and top boundaries respectively.*

To show that we can indeed divide our analysis into these three subcases, we make the following observation.

Observation 35. *Let $A_{k-1,\ell'}$ be the parent of $A_{k,\ell}$. Suppose there exists a type $(C1)$, or $(C2)$, or $(C3)$ path from $A_{k-1,\ell'}$ to $A_{k,\ell}$. Then all paths from $A_{k-1,\ell'}$ to $A_{k,\ell}$ are type $(C1)$, or $(C2)$, or $(C3)$, respectively.*

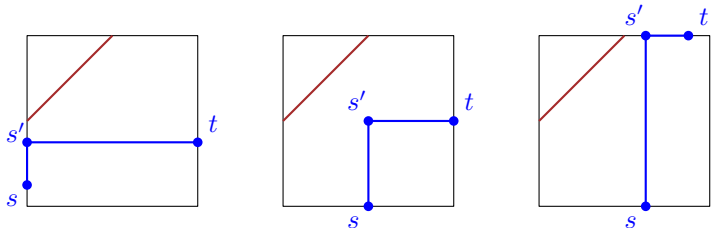


Figure 4.9: The type $(C1)$, $(C2)$ and $(C3)$ paths.

Proof. Suppose there were paths of different types between $A_{k-1,\ell'}$ to $A_{k,\ell}$. Then either their starting points are on different boundaries, or their ending points are on different boundaries. Therefore, there is either a critical point on $A_{k-1,\ell'}$ or on $A_{k,\ell}$. But this is not possible, so there are no paths of different types between $A_{k-1,\ell'}$ to $A_{k,\ell}$. □

We require separate case analyses for type $(C1)$, $(C2)$ and $(C3)$ paths. See Figure 4.10. We prove our claims for type $(C1)$ paths in Section 4.4.5, type $(C2)$ paths in Section 4.4.6, and type $(C3)$ paths in Section 4.4.7.

	Type $(C1)$	Type $(C2)$	Type $(C3)$
Lemma 31	Lemma 36	Lemma 42	Lemma 47
Lemma 32	Lemma 37	Lemma 43	Lemma 48
Lemma 33	Lemma 38	Lemma 44	Lemma 49

Figure 4.10: The proofs of Claims 31, 32 and 33 divided into three subcases.

4.4.5 Type (C1) paths

The type $(C1)$ path is first a vertical path from s to s', then a horizontal path from s' to t. However, we notice that since s' is on the left boundary, we can simplify the path to the horizontal part from s' to t, since we know the optimal cost at s' by our inductive hypothesis. The cost along the horizontal path from s' to t is a quadratic, so the cost function along $A_{k,\ell}$ is simply the cost function along $A_{k-1,\ell'}$ plus a quadratic. Using the same arguments as in Section 4.4.1, we yield the following three lemmas.

Lemma 36. *Let $A_{k-1,\ell'}$ be the parent of $A_{k,\ell}$. Suppose all paths from $A_{k-1,\ell'}$ to $A_{k,\ell}$ are type $(C1)$ paths. Then the cost along $A_{k,\ell}$ is a continuous piecewise quadratic function. Moreover, for every point t on the piecewise quadratic function, the left derivative at t is greater than or equal to the right derivative at t.*

Lemma 37. *Let $A_{k-1,\ell'}$ be the parent of $A_{k,\ell}$. Suppose all paths from $A_{k-1,\ell'}$ are type $(C1)$ paths. If we propagate a single quadratic piece of $A_{k-1,\ell'}$ to the next level A_k, it is a piecewise quadratic function with at most a constant number of pieces. Moreover, this propagation step takes only constant time.*

Lemma 38. *Let $A_{k-1,\ell'}$ be the parent of $A_{k,\ell}$. Suppose all paths from $A_{k-1,\ell'}$ to $A_{k,\ell}$ are type $(C1)$ paths. Then*

$$D(A_{k,\ell}) \leq D(A_{k-1,\ell'}),$$
$$|A_{k,\ell}| \leq |A_{k-1,\ell'}|.$$

4.4.6 Type (C2) paths

Definition 39. *Let $A_{k-1,\ell'}$ be the parent of $A_{k,\ell}$. Suppose all paths from $A_{k-1,\ell'}$ to $A_{k,\ell}$ are type $(C2)$ paths. Let the type $(C2)$ path be a vertical path from s to s' and a horizontal path from s' to t. Define $pathcost(s,t)$ to be the cost at s plus the cost along the type $(C2)$ path. In other words,*

$$pathcost(s,t) = cost(s) + \int_s^{s'} h(x) \cdot dx + \int_{s'}^{t} h(x) \cdot dx.$$

The cost at t is the minimum path cost over all type $(C2)$ paths ending at t. In other words, $cost(t) = \min_s pathcost(s,t)$.

Observation 40. *Suppose s is a point on $A_{k-1,\ell'}$ where the left derivative of $cost(s)$ at s does not match the right derivative of $cost(s)$ at s. Then $cost(t) \neq pathcost(s,t)$ for all t.*

Proof. We assume by inductive hypothesis that Lemma 17 is true along $A_{k-1,\ell'}$. In particular, we assume the cost function along $A_{k-1,\ell'}$ cannot obtain a local minimum at a point where its left derivative does not equal its right derivative. Recall that $pathcost(s,t) = cost(s) + \int_s^{s'} h(x) \cdot dx + \int_{s'}^{t} h(x)$. For fixed t, the second and third terms are (single piece) quadratic functions in terms of s. Therefore, for fixed t, we add two single piece quadratic functions to $cost(s)$ to obtain $pathcost(s,t)$. Therefore, $pathcost(s,t)$ cannot have a local minimum at s, since the left derivative does not match the right derivative at s in $cost(s)$, and adding the two quadratic functions to $cost(s)$ does not affect this property. Since $pathcost(s,t)$ does not have a local minimum at s, it cannot have a global minimum at s either, so $cost(t) \neq pathcost(s,t)$, as required. □

Observation 41. *Let $A_{k-1,\ell'}$ be the parent of $A_{k,\ell}$. Suppose that s is on $A_{k-1,\ell'}$ and t is on $A_{k,\ell}$ so that $cost(t) = pathcost(s,t)$. Then $\frac{\partial}{\partial s} pathcost(s,t) = 0$. Moreover, the set of points in a cell for which this occurs is a set of segments, and there are at most $|A_{k-1,\ell'}|$ such segments.*

Proof. By the contrapositive of Observation 40, we have that the point s is a point on $A_{k-1,\ell'}$ such that its the left derivative of $cost(s)$ at s matches the right derivative of $cost(s)$ at s. So the derivative of $cost(s)$ is well defined at s. Therefore, $\frac{\partial}{\partial s} pathcost(s,t)$ is well defined at s, since $pathcost(s,t) = cost(s) + \int_s^{s'} h(x) \cdot dx + \int_{s'}^{t} h(x)$, and the second and third terms are quadratic functions in terms of s if t is fixed. Since $\frac{\partial}{\partial s} pathcost(s,t)$ is well defined at s, and $cost(t) = pathcost(s,t)$ so $pathcost(s,t)$ is minimised as s, we have that $\frac{\partial}{\partial s} pathcost(s,t) = 0$. Since $pathcost(s,t)$ is a bivariate quadratic function with $|A_{k-1,\ell'}|$ pieces, the partial derivative equation $\frac{\partial}{\partial s} pathcost(s,t) = 0$ defines a bivariate linear function with $|A_{k-1,\ell'}|$ pieces. This bivariate linear function is a set of $|A_{k-1,\ell'}|$ segments, as required. □

Lemma 42. *Let $A_{k-1,\ell'}$ be the parent of $A_{k,\ell}$. Suppose all paths from $A_{k-1,\ell'}$ to $A_{k,\ell}$ are type $(C2)$ paths. Then the cost along $A_{k,\ell}$ is a continuous piecewise quadratic function. Moreover, for every point t on the piecewise quadratic function, the left derivative at t is greater than or equal to the right derivative at t.*

Proof. For all t on $A_{k,\ell}$, we define $\sigma(t)$ as the point on $A_{k-1,\ell'}$ such that $cost(t) = pc(t)$ where $pc(t) := pathcost(\sigma(t), t) = cost(\sigma(t)) + \int_{\sigma(t)}^{\sigma'(t)} h(r)dr + \int_{\sigma'(t)}^{t} h(r)dr$. By induction, we get that *cost* is a piecewise quadratic function on $A_{k-1,\ell'}$, and by Observation 41, we get

that σ is a piecewise linear function. Therefore, pc is a piecewise quadratic function and so is $cost$ on $A_{k,l}$.

Next, we show that for every point t_0 on $A_{k,\ell}$, pc is continuous in t_0 (which isn't necessarily true for σ) and the left derivative of pc at t_0 is greater than or equal to the right derivative of pc at t_0. We write $\partial^- pc(t_0) \geq \partial^+ pc(t_0)$. Consider the points directly to the left of t_0 such that a single linear piece of σ contains all of them in its domain. Now let $\overrightarrow{\sigma}$ be the linear function corresponding to this piece with its domain extended to the right of t_0, and let $\overrightarrow{pc}$ be the function $\overrightarrow{pc}(t) := pathcost(\overrightarrow{\sigma}(t), t) = cost(\overrightarrow{\sigma}(t)) + \int_{\overrightarrow{\sigma}(t)}^{\overrightarrow{\sigma}'(t)} h(r)dr + \int_{\overrightarrow{\sigma}'(t)}^{t} h(r)dr$. We analogously define $\overleftarrow{\sigma}$ and $\overleftarrow{pc}$ by considering points directly to the right of t_0 and extending the piece's domain to the left. Note that $\overrightarrow{\sigma} \neq \overleftarrow{\sigma}$ only if σ changes from one linear piece to another at t_0.

Both $\overrightarrow{pc}$ and $\overleftarrow{pc}$ are continuous, since $\overrightarrow{\sigma}$ and $\overleftarrow{\sigma}$ are linear while $cost$ is continuous on $A_{k-1,l'}$ by induction. For all $t < t_0$ and close to t_0, we have $\overrightarrow{pc}(t) = pc(t) = cost(t) = \min_s pathcost(s, t)$ by definition and thus $\overrightarrow{pc}(t) \leq \overleftarrow{pc}(t)$. Similarly, for all $t > t_0$ close to t_0, we have $\overrightarrow{pc}(t) \geq \overleftarrow{pc}(t)$. Together, this implies $\overrightarrow{pc}(t_0) = \overleftarrow{pc}(t_0)$, which in turn yields that pc is continuous in t_0 with $pc(t_0) = \overrightarrow{pc}(t_0) = \overleftarrow{pc}(t_0)$.

The continuity of $\overrightarrow{pc}$, $\overleftarrow{pc}$ and pc at t_0 also means that they all have a left and a right derivative at t_0. Assume for the sake of contradiction that $\partial^- \overrightarrow{pc}(t_0) \neq \partial^+ \overrightarrow{pc}(t_0)$. This would require $\partial^- cost(\overrightarrow{\sigma}(t_0)) \neq \partial^+ cost(\overrightarrow{\sigma}(t_0))$, because the second and third terms of $\overrightarrow{pc}(t)$ are quadratic functions of t without break points. Due to the linearity of $\overrightarrow{\sigma}$, $\partial^- cost(\overrightarrow{\sigma}(t_0)) \neq \partial^+ cost(\overrightarrow{\sigma}(t_0))$ only occurs when $\overrightarrow{\sigma}(t_0)$ is a break point of $cost$, which is impossible by $\overrightarrow{pc}(t_0) = pc(t_0) = cost(t_0)$ and the contrapositive of Observation 40. This yields a contradiction, so $\partial^- \overrightarrow{pc}(t_0) = \partial^+ \overrightarrow{pc}(t_0)$.

Finally, we have $\partial^- \overrightarrow{pc}(t_0) = \partial^- pc(t_0)$ and $\partial^+ \overrightarrow{pc}(t_0) \geq \partial^+ pc(t_0)$, since $\overrightarrow{pc}(t) = pc(t)$ for all $t \leq t_0$ and $\overrightarrow{pc}(t) \geq pc(t)$ for all $t > t_0$ close to t_0. Putting everything together yields

$$\partial^- cost(t_0) = \partial^- pc(t_0) = \partial^- \overrightarrow{pc}(t_0) = \partial^+ \overrightarrow{pc}(t_0) \geq \partial^+ pc(t_0) = \partial^+ cost(t_0)$$

as required. □

Lemma 43. *Let $A_{k-1,\ell'}$ be the parent of $A_{k,\ell}$. Suppose all paths from $A_{k-1,\ell'}$ are type $(C2)$ paths. If we propagate a single quadratic piece of $A_{k-1,\ell'}$ to the next level A_k, the cost is a quadratic function. Moreover, this propagation step takes only constant time.*

Proof. Let s be a point on a single quadratic piece on $A_{k-1,\ell'}$. In constant time, we can compute the bivariate quadratic function $pathcost(s, t)$. In constant time, we can compute the segment where $\frac{\partial}{\partial s} pathcost(s, t) = 0$. For all (s, t) on this segment, we write s as a linear function of t. We know by Observation 41 that $cost(t) = pathcost(s, t)$ along this segment. In constant time, we can substitute the linear function of s in terms of t into $pathcost(s, t)$ to obtain $cost(t)$, i.e. the propagated cost along the next level A_k. □

Lemma 44. *Let $A_{k-1,\ell'}$ be the parent of $A_{k,\ell}$. Suppose all paths from $A_{k-1,\ell'}$ to $A_{k,\ell}$ are type $(C2)$ paths. Then*

$$D(A_{k,\ell}) \leq D(A_{k-1,\ell'}),$$

$$|A_{k,\ell}| \leq |A_{k-1,\ell'}|.$$

Proof. First, we show $|A_{k,\ell}| \leq |A_{k-1,\ell'}|$. For all t on $A_{k,\ell}$, we define $s(t)$ as the point on $A_{k-1,\ell'}$ such that that $cost(t) = pathcost(s(t), t)$. By Observation 41, we get that $s(t)$ is a piecewise linear function in terms of t, and $s(t)$ has at most $|A_{k-1,\ell'}|$ linear pieces.

Substituting $s(t)$ into $pathcost(s(t), t)$, we get that $cost(t)$ is a piecewise quadratic function, which is the lower envelope of $|A_{k-1,\ell'}|$ quadratic pieces. By Lemma 11, the lower envelope of $|A_{k-1,\ell'}|$ pieces appear in the same order as the input pieces they were propagated from. Therefore, the lower envelope $cost(t)$ has at most $|A_{k-1,\ell'}|$ quadratic pieces, as required.

Next, we show $D(A_{k,\ell}) \leq D(A_{k-1,\ell'})$. Suppose that $f_1(x) = ax^2 + bx + c_1$, and $f_2(x) = ax^2 + bx + c_2$. We apply the propagation step in Lemma 43 to $f_1(x)$ and $f_2(x)$. Let $pathcost_1(s, t)$ be the bivariate quadratic function for s in the domain of $f_1(x)$, and similarly $pathcost_2(s, t)$ be the bivariate quadratic function for s in the domain of $f_2(x)$. Since $f_1(x)$ and $f_2(x)$ differ by at most a constant, we get that $pathcost_1(s, t)$ and $pathcost_2(s, t)$ differ by at most a constant. The derivative $\frac{\partial}{\partial s} pathcost(s, t)$ does not depend on the constant terms, so s and t have the same linear relationship for s in the domain of $f_1(x)$ and $f_2(x)$. Substituting the linear function $s(t)$ into $pathcost_1(s(t), t)$ and $pathcost_2(s(t), t)$, we obtain $cost_1(t)$ and $cost_2(t)$, that differ at most by an additive constant. Hence, any two quadratic functions in $cost(s)$ that share the same (a, b) terms propagate to a pair of quadratic functions in $cost(t)$ that share the same (a, b) terms. Hence, every one of the $D(A_{k-1,\ell'})$ many distinct (a, b) pairs in $cost(s)$ map to at most one of the $D(A_{k,\ell})$ many distinct (a, b) pairs in $cost(t)$. Therefore, $D(A_{k,\ell}) \leq D(A_{k-1,\ell'})$, as required. □

4.4.7 Type (C3) paths

Definition 45. *Let $A_{k-1,\ell'}$ be the parent of $A_{k,\ell}$. Let the type $(C3.1)$ function be the cost function along $A_{k,\ell}$ if we only allow vertical type $(C3)$ paths from $A_{k-1,\ell'}$ to $A_{k,\ell}$. Let the type $(C3.2)$ function be the cost function along $A_{k,\ell}$ if we allow arbitrary type $(C3)$ paths (vertical followed by a horizontal path) from $A_{k-1,\ell'}$ to $A_{k,\ell}$. See Figure 4.11.*

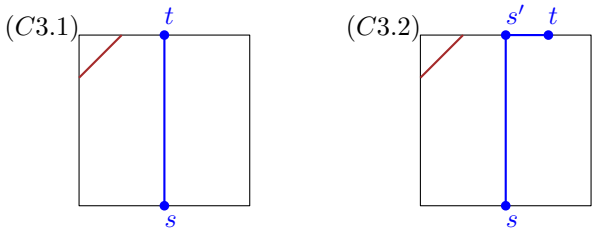


Figure 4.11: The type $(C3.1)$ and type $(C3.2)$ paths.

Similar to type $(B1)$ and type $(B2)$ functions in Section 4.4.3, our approach will be to first compute the type $(C3.1)$ and then modify it to obtain the type $(C3.2)$ function. Recall that for a type $(B1)$ function $f(t)$, the type $(B2)$ function is the cumulative minimum of $f(t)$. Recall that the cumulative minimum has horizontal extensions at each of the local minima of $f(t)$. For type $(C3)$ paths, we apply similar extensions, but our new extensions will be non-horizontal, but instead follow the shape of a quadratic function $g(t)$. See Figure 4.12.

Observation 46. *Let $cost_{3.1}(t)$ be a type $(C3.1)$ function, and let $g(t)$ be the integral of the height function $h(t)$ along the top boundary. Then the type $(C3.2)$ function, $cost_{3.2}(t)$, is given by*

$$cost_{3.2}(t) = \min_{s' \leq t} \big(cost_{3.1}(s') - g(s')\big) + g(t)$$

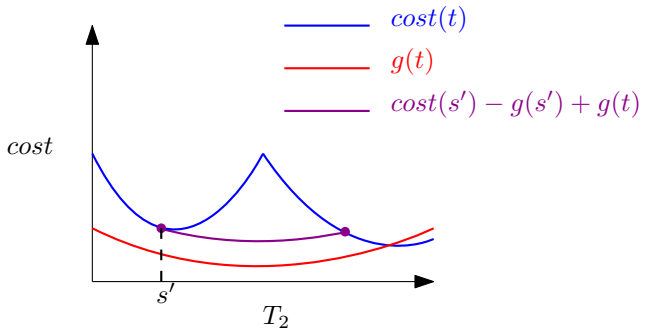


Figure 4.12: A non-horizontal extension (purple) of a function (blue) that follows the shape of a quadratic (red).

Proof. Recall that a type $(C3.2)$ path is a path from s (bottom) to s' (top) to t (top). The type $(C3.2)$ cost function is the minimum cost over all such type $(C3.2)$ paths. Therefore,

$$\begin{aligned} cost_{3.2}(t) &= \min_{s' \le t} \left(cost(s) + \int_s^{s'} h(t)dt + \int_{s'}^{t} h(t)dt \right) \\ &= \min_{s' \le t} \left(cost_{3.1}(s') + \int_{s'}^{t} h(t)dt \right) \\ &= \min_{s' \le t} \left(cost_{3.1}(s') - g(s') + g(t) \right) \\ &= \min_{s' \le t} \left(cost_{3.1}(s') - g(s') \right) + g(t), \end{aligned}$$

as required. □

Now, we can compute $cost_{3.2}(t)$ from $cost(s)$ in the following way. First, we compute the quadratic function $\int_s^{s'} h(t)dt$ and add it to $cost(s)$ to obtain $cost_{3.1}(s')$. Next, we compute $g(s')$, the integral of the height function along the top boundary, and subtract it from $cost_{3.1}(s')$. Then, we compute its cumulative minimum function, $\min_{s' \le t} \left(cost_{3.1}(s') - g(s') \right)$. Finally, we add $g(t)$. In each step, we either add or subtract a quadratic, or take the cumulative minimum of a function. These steps are essentially the same as the ones for type (B) paths, with minor modifications. By applying essentially the same arguments as in Section 4.4.3, we obtain the following three lemmas.

Lemma 47. *Let $A_{k-1,\ell'}$ be the parent of $A_{k,\ell}$. Suppose all paths from $A_{k-1,\ell'}$ to $A_{k,\ell}$ are type $(C3)$ paths. Then the cost along $A_{k,\ell}$ is a continuous piecewise quadratic function. Moreover, the local minima along $A_{k,\ell}$ are either at its endpoints or where its derivative is well defined and equal to zero.*

Lemma 48. *Let $A_{k-1,\ell'}$ be the parent of $A_{k,\ell}$. Suppose all paths from $A_{k-1,\ell'}$ are type $(C3)$ paths. If we propagate a single quadratic piece of $A_{k-1,\ell'}$ to the next level A_k, it is a piecewise quadratic function with at most a constant number of pieces. Moreover, this propagation step takes only constant time.*

Lemma 49. *Let $A_{k-1,\ell'}$ be the parent of $A_{k,\ell}$. Suppose all paths from $A_{k-1,\ell'}$ to $A_{k,\ell}$ are type $(C3)$ paths. Then*

$$D(A_{k,\ell}) \le D(A_{k-1,\ell'}),$$

$$|A_{k,\ell}| \leq |A_{k-1,\ell'}|.$$

4.5 Conclusion

We presented the first exact algorithm for computing CDTW of one-dimensional curves, which runs in polynomial time. Our main technical contribution is bounding the total complexity of the functions which the algorithm propagates, to bound the total running time of the algorithm. One direction for future work is to improve the upper bound on the total complexity of the propagated functions. Our $O(n^5)$ upper bound is pessimistic, for example, we do not know of a worst case instance. Another direction is to compute CDTW in higher dimensions. In two dimensions, the Euclidean $\mathcal{L}_2$ norm is the most commonly used norm, however, this is likely to result in algebraic issues similar to that for the weighted region problem [48]. One way to avoid these algebraic issues is to use a polyhedral norm, such as the $\mathcal{L}_1$, $\mathcal{L}_\infty$, or an approximation of the $\mathcal{L}_2$ norm [72, 109]. This would result in an approximation algorithm similar to [124], but without a dependency on the spread.

Acknowledgements

The authors would like to thank Jan Erik Swiadek for helpful feedback on the manuscript. In particular, we included their improved proof of Lemma 42.

Chapter 5

Improving the dilation of a metric graph by adding edges

5.1 Introduction

Let $G = (V, E)$ be a graph embedded in a metric space M. For every pair of points $u, v \in V$, the weight of the edge (u, v) is equal to the distance $d_M(u, v)$ between points u and v in the metric space M. Let $d_G(u, v)$ be the weight of the shortest path between u and v in the graph G. For any real number $t > 1$, we call G a *t-spanner* if $d_G(u, v) \leq t \cdot d_M(u, v)$ for every pair of points $u, v \in V$. The *stretch*, or *dilation*, of G is the smallest t for which G is a t-spanner.

Spanners have been studied extensively in the literature, especially in the geometric setting. Given a fixed $t > 1$, a fixed dimension $d \geq 1$, and a set of n points V in d-dimensional Euclidean space, there is a t-spanner on the point set V with $O(n)$ edges. For a summary of the considerable research on geometric spanners, see the surveys [77, 98, 147] and the book by Narasimhan and Smid [133]. Spanners in doubling metrics [50, 95, 110] and in general graphs [19, 135, 154] have also received considerable attention.

Most of the literature on spanners focuses on building the graph from scratch. This chapter instead focuses on adding edges to improve an existing graph. Applications where graph networks tend to be better connected over time include road, rail, electric and communication networks. The overall quality of these networks depend on both the quality of the initial design and the quality of the additions. In this chapter, we focus on the latter. In particular, given an initial metric graph, and a budget of k edges, which k edges do we add to produce a minimum-dilation graph?

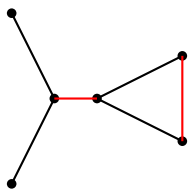


Figure 5.1: An example where $k = 2$ edges (red) are added to an initial graph G (black) to produce a minimum-dilation graph.

Problem 1. *Given a positive integer k and a metric graph $G = (V, E)$, compute a set $S \subseteq V \times V$ of k edges so that the dilation of the resulting graph $G' = (V, E \cup S)$ is minimised.*

The problem stated is a major open problem in the field [81, 122, 166]. It is also one of twelve open problems posed in the final chapter of Narasimhan and Smid's book [133]. As no major breakthroughs have been made, special cases have been studied.

The first special case is when $k = 1$. Let n and m be the number of vertices and edges of the graph G, respectively. Farshi et al. [81] provided an $O(n^4)$ time exact algorithm and an $O(mn + n^2 \log n)$ time 3-approximation. Wulff-Nilsen [166] improved the running time of the exact algorithm to $O(n^3 \log n)$, and in a follow-up paper Luo and Wulff-Nilsen [122] provided an $O((n^4 \log n)/\sqrt{m})$ time exact algorithm that uses linear space. Several of the papers that study the $k = 1$ case mention the $k > 1$ case as one of the main open problems in the field.

The second special case is if G is an empty graph. Giannopoulos et al. [91] and Gudmundsson and Smid [102] independently proved that it is NP-hard to produce the highest quality spanner by adding k edges to an empty graph. This implies that Problem 1 is NP-hard. If we restrict ourselves to polynomial time algorithms, it therefore makes sense to consider approximation algorithms. In Euclidean space, Aronov et al. [15] showed how to add $k = n - 1 + \ell$ edges to an empty graph to produce an $O(n/(\ell + 1))$-spanner in $O(n \log n)$ time. By setting $\ell = \varepsilon n$, this result implies an $O(1/\varepsilon)$-approximation to Problem 1 for all $k \geq (1 + \varepsilon)n$. However, the general case where G is a non-empty (Euclidean or metric) graph and $k \leq n - 1$ still remains open.

Farshi et al. [81] conjectured that generalising their algorithm to general k may provide a reasonable approximation algorithm. In Section 5.5, we show an $\Omega(2^k)$ lower bound for their algorithm.

In this chapter we obtain the first positive result for the general case. Our approximation algorithm runs in $O(n^3 \log n)$ time and guarantees an $O(k)$-approximation factor. Although our algorithm may not be optimal, we hope that we provide some insight for further research, or for related graph augmentation problems [8, 112, 113, 117].

We provide a tight analysis of our algorithm. We show that, for any $\varepsilon > 0$, our algorithm yields an approximation factor of $(1 + \varepsilon)(k + 1)$, but the same algorithm cannot yield an approximation factor better than $(1 - \varepsilon)(k + 1)$. We achieve our main result by reducing Problem 1 to the following approximate decision version:

Problem 2. *Given an integer k, a real number t, and a metric graph $G = (V, E)$, decide whether $t^* \leq t$ or $t^* > \frac{t}{k+1}$, where t^* is the minimum dilation of $G' = (V, E \cup S)$ over all sets S where $S \subseteq V \times V$ and $|S| = k$. In the case where $\frac{t}{k+1} < t^* \leq t$, either of the two options may be chosen arbitrarily.*

Our algorithm for Problem 2 is a slight modification of the standard greedy t-spanner algorithm. We provide details of our algorithm and argue its correctness in Section 5.2. In Section 5.3, we show how to use the approximate decision algorithm for Problem 2 to develop an approximation algorithm for Problem 1. We prove that only $O(\log n)$ calls to the greedy algorithm is required to obtain an $(1 + \varepsilon)(k + 1)$-approximation. Finally, in Section 5.4, we provide a construction to show that the same algorithm cannot yield an approximation factor better than $(1 - \varepsilon)(k + 1)$.

5.2 The Greedy Construction

As mentioned in the introduction, our approach to solving Problem 2 is a modified greedy t-spanner construction. We introduce some notation for the purposes of stating the algorithm. For an edge $e \in V \times V$, let $d_M(e)$ denote the length of the edge e in the metric space M. Given a graph G, let $\delta_G(e)$ denote the shortest path between the endpoints of e in the graph G. Let $d_G(e)$ be the total length of edges along the path $\delta_G(e)$.

In the original greedy spanner construction, the algorithm begins with an empty graph G, and a positive real value $t > 1$, and yields a t-spanner as follows: sort all the edges in $\{V \times V\}$ by increasing weight and then process them in order. Processing an edge e entails a shortest path query. If $d_G(e) > t \cdot d_M(e)$, then the edge e is added to G, otherwise it is discarded. The algorithm terminates when all edges have been processed. The resulting graph is a t-spanner.

In our setting we will start with a initial graph G, a positive real value $t > 1$ and a positive integer k. Our modified greedy algorithm sorts the edges in $\{V \times V\} \setminus E$ by increasing weight and then processes them in order. For each edge e, we perform a shortest path query. If $d_G(e) > t \cdot d_M(e)$, then the edge e is added to G, otherwise it is discarded. The algorithm terminates if all edges have been processed, or if $k+1$ edges have been added to G by the algorithm.

Formally, the greedy edges a_i and the augmented graphs G_i are defined inductively as follows:

Definition 3. *Let $G_0 = G$, and for $1 \leq i \leq k+1$, let $G_i = G_{i-1} \cup a_i$ where a_i is the shortest edge in $V \times V$ satisfying $d_{G_{i-1}}(a_i) > t \cdot d_M(a_i)$.*

If the algorithm terminates after all the edges have been processed, then at most k edges have been added to yield a t-spanner. Therefore $t^* \leq t$. Otherwise, if at least $k+1$ edges are added, we will prove in Section 5.2.1 that $t^* > \frac{t}{k+1}$.

5.2.1 Proof of correctness

Our approach is to use the edges added by the greedy algorithm to obtain an upper bound on t with respect to t^*. Our upper bound comes from the following relationship, which is a straightforward consequence of Definition 3:

Observation 4. *In the graph G_{i-1}, if there is a connected path between the endpoints of a_i with total length L, then $L > t \cdot d_M(a_i)$.*

Our goal is to construct a connected path in G_{i-1} between the endpoints of a_i and to bound its length by $(k+1)\, t^* \cdot d_M(a_i)$. If we are able to do this, then Observation 4 would immediately imply that $(k+1)\, t^* > t$, as required. Note that i is some fixed integer between 1 and $k+1$. As part of our construction, we will show how to select a suitable value for i.

To motivate how we construct a connected path in G_{i-1} between the endpoints of a_i, let us consider a special case where $k = 1$. Let G be the initial graph and let the first two greedy edges be a_1 and a_2. Suppose that an optimal edge to add is s_1, and let $G^* = G \cup \{s_1\}$. See Figure 5.2.

We select $i = 2$ in Observation 4, so that our goal is to construct a connected path in $G_1 = G \cup \{a_1\}$ between the endpoints of a_2 and upper bound its length by $2\, t^* \cdot d_M(a_2)$.

A naïve connected path between the endpoints of a_2 that has length upper bounded by $t^* \cdot d_M(a_2)$ is the path $\delta_{G^*}(a_2)$, which we recall is the shortest path between the endpoints of

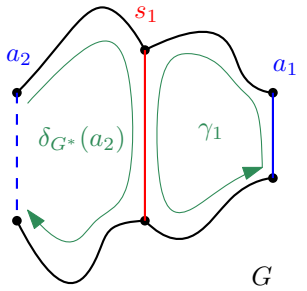


Figure 5.2: The graph G with optimal edge s_1 and greedy edges a_1 and a_2.

a_2 in the graph G^*. The path is shown in Figure 5.2. The reason that $d_{G^*}(a_2) \leq t^* \cdot d_M(a_2)$ is because the dilation of G^* is t^*. Unfortunately, the issue with this path is that it uses the edge s_1 and therefore is not a path in G_1, so Observation 4 does not apply.

We modify $\delta_{G^*}(a_2)$ into a longer path that does not use s_1. Our approach is to combine the path with a cycle by using the *symmetric difference* operation. Recall that the symmetric difference of a set of sets are all the elements that appear in an odd number of those sets.

To remove s_1 from the path $\delta_{G^*}(a_2)$, we take its symmetric difference with the cycle γ_1, which is formed by linking the path $\delta_{G^*}(a_1)$ and the edge a_1 end to end. Ideally, the symmetric difference of $\delta_{G^*}(a_2)$ and γ_1 would form a connected path between the endpoints of a_2. Moreover, if both the path $\delta_{G^*}(a_2)$ and the cycle γ_1 use the edge s_1 exactly once, then taking the symmetric difference cancels the two occurrences of s_1, leaving a path that is entirely in G_1.

In fact, we can show this approach works in general. We begin with the naïve path $\delta_{G^*}(a_i)$, where G^* is the optimal graph defined as follows:

Definition 5. *Let $S \subseteq V \times V$ be the set of k edges so that $G \cup S$ has dilation t^*. Then $G^* = G \cup S$.*

Similar to the $k = 1$ case, the path $\delta_{G^*}(a_i)$ is not in the graph G_{i-1}. We modify the path $\delta_{G^*}(a_i)$ by taking its symmetric difference with a set of cycles. We prove that for any set of cycles, the symmetric difference of $\delta_{G^*}(a_i)$ and the set of cycles always contains a connected path between the endpoints of a_i. Moreover, we show how to select the set of cycles in such a way that all edges in S are cancelled out by the symmetric difference. In this way, we have constructed a connected path in the graph G_{i-1} between the endpoints of a_i.

We first prove that taking the symmetric difference of $\delta_{G^*}(a_i)$ with any set of cycles maintains the invariant that there always exists a connected path between the endpoints of a_i.

Lemma 6. *The symmetric difference of a path P with any number of cycles contains a connected path between the endpoints of P. See Figure 5.3.*

Proof. Consider a subgraph formed by the symmetric difference of P and a set of cycles. We will look at the degree of all vertices in this subgraph.

Consider the parity of the degree of each vertex. Taking the symmetric difference maintains the parity of the sum of the degrees. The contribution of a cycle to the degree of all vertices is even, whereas the contribution of P to the degree of all vertices is even except

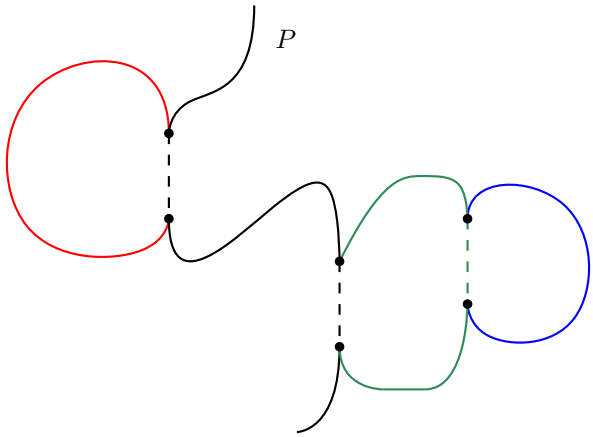


Figure 5.3: Given a path (black) and cycles (red, green, blue), the symmetric difference (solid) contains a connected path between the endpoints of the black path.

for the endpoints of P. Hence, the only two vertices with odd degree are the endpoints of P. Applying Euler's theorem to the connected component that contains the endpoints of P, we deduce that there is an Eulerian trail between the two vertices of odd degree. Hence, there is a connected path between the endpoints of P. □

Next, we construct the set of cycles $\Gamma = \{\gamma_j : 1 \leq j \leq k+1\}$. We will apply Lemma 6 to our naïve connected path $\delta_{G^*}(a_i)$ and a subset of Γ. Each cycle γ_j is simply a generalisation of γ_1 from the $k = 1$ case, which we recall is formed by linking the path $d_{G^*}(a_1)$ and the edge a_1 end to end.

Definition 7. *Let γ_j be a cycle formed by linking the path $\delta_{G^*}(a_j)$ and the edge a_j end to end.*

A subset of Γ is chosen in such a way so that the symmetric difference with $\delta_{G^*}(a_i)$ consists only of edges in G_{i-1}. To ensure this, we must choose the path $\delta_{G^*}(a_i)$ and the cycles γ_j so that all edges in S cancel out in the symmetric difference. We use elementary linear algebra to provide a non-constructive proof that there exists a path $\delta_{G^*}(a_i)$ and subset of Γ where this property holds.

Lemma 8. *Let $\{a_1, a_2, \ldots, a_{k+1}\}$ be the first $k+1$ edges given in Definition 3. Then there exists a non-empty subset $I \subseteq \{1, 2, \ldots, k+1\}$ so that the symmetric difference of $\{\delta_{G^*}(a_j) : j \in I\}$ does not contain any edges of S.*

Proof. Recall from Definition 5 that S is the set of k edges so that $G \cup S$ has dilation t^*. Consider $\delta_{G^*}(a_j) \cap S$, which is a subset of S. We can represent any subset of S as an element of the vector space $\{0,1\}^S$, as each binary digit simply represents whether an element is in that subset. Take the basis $\{1_j : j \in S\}$ for the vector space $\{0,1\}^S$. The basis element 1_j simply represents whether the j^{th} element of S is in that subset. Hence, we can expand $\delta_{G^*}(a_j) \cap S$ into a sum of basis elements by writing $\delta_{G^*}(a_j) \cap S = \sum \lambda_{ij} 1_j$.

As there are $k+1$ subsets $\delta_{G^*}(a_j) \cap S$, their vector space expansions $\sum \lambda_{ij} 1_j$ must be linearly dependent. The linear dependence equation, when taken in modulo 2, can be rearranged into the form $\sum_{j \in I} \delta_{G^*}(a_j) \cap S = 0$ for some $I \subseteq \{1, 2, \ldots, k+1\}$ and $I \neq \emptyset$.

The modulo 2 equation $\sum_{j \in I} \delta_{G^*}(a_j) \cap S = 0$ directly implies that the symmetric difference of $\{\delta_{G^*}(a_j) \cap S : j \in I\}$ is empty. □

For the remainder of this section, let $I \subseteq \{1, 2, \ldots, k+1\}$ be the subset that satisfies the conditions of Lemma 8, in other words, the symmetric difference of $\{\delta_{G^*}(a_j) : j \in I\}$ does not contain any edges of S. We select the path $\delta_{G^*}(a_i)$ where $i = \max I$. Let $J = I \setminus \{i\}$ and select the subset $\Gamma' = \{\gamma_j : j \in J\}$. We construct the set of edges that is the symmetric difference of $\delta_{G^*}(a_i)$ and Γ'. This completes the construction of the required path.

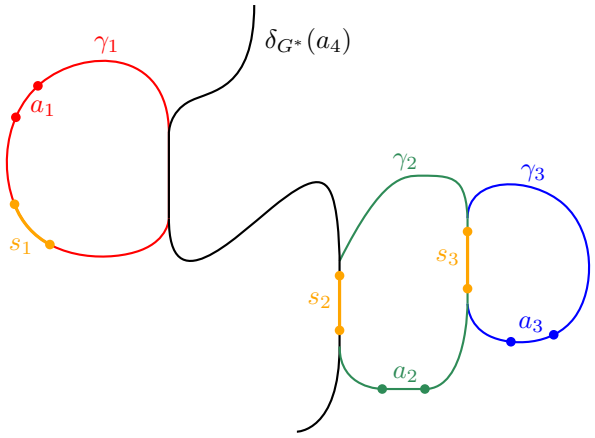


Figure 5.4: An example where we take the symmetric difference of $d_{G^*}(a_4)$, γ_2 and γ_3 to avoid all three of the edges s_1, s_2 and s_3 that are not in G_3.

For an illustrated example, see Figure 5.4. Let $k = 3$ and $S = \{s_1, s_2, s_3\}$, so that $s_1 \in \delta_{G^*}(a_1)$, $s_2 \in \delta_{G^*}(a_2), \delta_{G^*}(a_4)$ and $s_3 \in \delta_{G^*}(a_2), \delta_{G^*}(a_3)$. By Lemma 8, there must be a non-empty subset $I \subseteq \{1, 2, 3, 4\}$ so that the symmetric difference of $\{\delta_{G^*}(a_j) : j \in I\}$ does not contain any of the edges s_1, s_2 or s_3. In particular, the subset $I = \{2, 3, 4\}$ includes s_1 zero times, and s_2 and s_3 both twice. Hence, the symmetric difference of $\delta_{G^*}(a_4)$ with the cycles $\Gamma' = \{\gamma_2, \gamma_3\}$ avoids all three of the edges s_1, s_2, and s_3.

Now we show this symmetric difference indeed satisfies the conditions of Observation 4, so that it can be applied to yield an upper bound on t with respect to t^*. Recall that the requirements of Observation 4 are that the set of edges must contain a connected path between the endpoints of a_i that uses only edges in G_{i-1}. By Lemma 6, the symmetric difference contains a connected path between the endpoints of a_i. By Lemma 8, we have $\{\delta_{G^*}(a_j) \cap S : j \in I\} = \emptyset$, so therefore the symmetric difference of $\{\delta_{G^*}(a_j) : j \in I\}$ does not contain any edges of S. This implies that the symmetric difference of $\{\delta_{G^*}(a_i)\}$ and $\Gamma' = \{\gamma_j : j \in J\}$ also does not contain any edges of S. Hence, we have constructed a set of edges that contains a connected path in G_{i-1} between the endpoints of a_i, as required.

Observation 4 implies an upper bound on t in terms of the lengths of all the edges in the symmetric difference of $\{\delta_{G^*}(a_i)\}$ and $\Gamma' = \{\gamma_j : j \in J\}$. In Lemma 9 we formalise this upper bound. Then, in Lemma 10, we use the fact that the dilation of G^* is t^* to obtain an upper bound on the sum of the lengths in each cycle γ_j. In Lemma 11, we strengthen the inequality by giving a lower bound on the length of the edges that are both in γ_i and S, and therefore cannot be part of the final symmetric difference. In Theorem 13 we put this all together and prove the final bound $(k+1)\, t^* > t$.

Let c_j be the total length of edges in the cycle γ_j. Let c'_j be the total length of edges in the intersection $\delta_{G^*}(a_j) \cap S$. Then Observation 4 implies:

Lemma 9. $d_{G^*}(a_i) + \sum_{j \in J} c_j - \sum_{j \in I} c'_j > t \cdot d_M(a_i)$

Proof. The total length of all edges in $\delta_{G^*}(a_i)$ is $d_{G^*}(a_i)$. The total length of all edges in γ_j is c_j. Taking the sum $d_{G_{i-1}}(a_i) + \sum_{j \in J} c_j$ yields an upper bound on the total length of all edges in the symmetric difference $\{\delta_{G^*}(a_i)\}$ and $\{\gamma_j : j \in J\}$. However, this total length includes edges in S, in particular, it includes the total length of all edges in the intersections $\{\delta_{G^*}(a_j) \cap S : j \in I\}$. We know from Lemma 8 that no edge in S appears in the symmetric difference, so we do not need to include any of the edges in $\{\delta_{G^*}(a_j) \cap S : j \in I\}$ in the total length. Hence, $d_{G^*}(a_i) + \sum_{j \in J} c_j - \sum_{j \in I} c'_j$ is an upper bound on the total length of the edges in the symmetric difference. Since the symmetric difference contains a connected path in G_{i-1} between the endpoints of a_i, Observation 4 implies the stated inequality. □

Next, we use the relationship between γ_j and the graph G^* to obtain an upper bound on c_j.

Lemma 10. $c_j \leq (t^* + 1) \cdot d_M(a_j)$

Proof. Recall from Definition 7 that the cycle γ_j is the path $\delta_{G^*}(a_j)$ and the edge a_j linked end to end. Since the dilation of G^* is t^*, we have $d_{G^*}(a_j) \leq t^* \cdot d_M(a_j)$. Therefore, $c_j = d_{G^*}(a_j) + d_M(a_j) \leq (t^* + 1) \cdot d_M(a_j)$. □

We strengthen the inequality in Lemma 9 by providing a lower bound on the edges that are in γ_j but cannot be part of the final symmetric difference.

Lemma 11. *If $t \geq (k+1)\, t^*$, then $\frac{k}{k+1} \cdot d_M(a_j) \leq c'_j$ for all j.*

Proof. First, we prove the inequality

$$d_{G_{j-1}}(a_j) \leq d_{G^*}(a_j) + \sum_{s \in \delta_{G^*}(a_j) \cap S} d_{G_{j-1}}(s).$$

We do so in a similar manner to Lemma 9. We construct a path in G_{j-1} between the endpoints of a_j that has length $d_{G^*}(a_j) + \sum_{s \in \delta_{G^*}(a_j) \cap S} d_{G_{j-1}}(s)$. We start with the path $d_{G^*}(a_j)$. We modify it taking the symmetric difference of $d_{G^*}(a_j)$ with a set of cycles $\beta = \{\beta_s : s \in \delta_{G^*}(a_j) \cap S\}$. The cycle β_s is formed by linking the path $d_{G_{j-1}}(s)$ and the edge s end to end. The cycle β_s replaces every edge $s \in \delta_{G^*}(a_j) \cap S$ with the path $d_{G_{j-1}}(s) \in G_{j-1}$. Hence, the symmetric difference of $d_{G^*}(a_j)$ with the set β is a path in G_{j-1} between the endpoints of a_j. Therefore, we have $d_{G_{j-1}}(a_j) \leq d_{G^*}(a_j) + \sum_{s \in \delta_{G^*}(a_j) \cap S} d_{G_{j-1}}(s)$.

Suppose for sake of contradiction that $\frac{k}{k+1} \cdot d_M(a_j) > c'_j$. Consider any $s \in \delta_{G^*}(a_j) \cap S$. Then s is shorter than a_j, since $d_M(a_j) > \frac{k}{k+1} \cdot d_M(a_j) > c'_j \geq d_M(s)$. In the graph G_{j-1}, the edge a_j is a shortest edge satisfying $d_{G_{j-1}}(a_j) > t \cdot d_M(a_j)$. Since s is shorter than a_j, we must have that $d_{G_{j-1}}(s) \leq t \cdot d_M(s)$. Now,

$$\begin{aligned}
d_{G_{j-1}}(a_j) &\leq d_{G^*}(a_j) + \textstyle\sum_{s \in \delta_{G^*}(a_j) \cap S} d_{G_{j-1}}(s) \\
&\leq t^* \cdot d_M(a_j) + t \cdot \textstyle\sum_{s \in \delta_{G^*}(a_j) \cap S} d_M(s) \\
&= t^* \cdot d_M(a_j) + t \cdot c'_j \\
&< t^* \cdot d_M(a_j) + t \cdot \tfrac{k}{k+1} d_M(a_j) \\
&\leq t \cdot \tfrac{1}{k+1} d_M(a_j) + t \cdot \tfrac{k}{k+1} d_M(a_j) \\
&= t \cdot d_M(a_j)
\end{aligned}$$

where the second last line is given by $t \geq (k+1)\, t^*$. But we know from Definition 3 that $d_{G_{j-1}}(a_j) > t \cdot d_M(a_j)$, so we obtain a contradiction. Therefore, we must have $\frac{k}{k+1} \cdot d_M(a_j) \leq c'_j$. □

Using Lemmas 9-11 we are able to prove the main result of this section.

Theorem 12. *Suppose the greedy algorithm adds $k+1$ edges into the graph. Then $(k+1)\, t^* > t$.*

Proof. Combining Lemmas 9 and 10 yields:

$$\begin{array}{rcl} t \cdot d_M(a_i) & < & d_{G^*}(a_i) + \sum_{j \in J} c_j - \sum_{j \in I} c'_j \\ & \leq & t^* \cdot d_M(a_i) + \sum_{j \in J} (t^*+1) d_M(a_j) - \sum_{j \in I} c'_j \\ & = & t^* \cdot \sum_{j \in I} d_M(a_j) + \sum_{j \in J} d_M(a_j) - \sum_{j \in I} c'_j \end{array}$$

Suppose for sake of contradiction that $t \geq (k+1)\, t^*$. By Lemma 11 we have $\frac{k}{k+1} d_M(a_j) \leq c'_j$. Summing over all $j \in I$ yields:

$$\begin{array}{rcl} \sum_{j \in I} c'_j & \geq & \sum_{j \in I} \frac{k}{k+1} d_M(a_j) \\ & = & \frac{k}{k+1} d_M(a_i) + \sum_{j \in J} \frac{k}{k+1} d_M(a_j) \\ & \geq & \sum_{j \in J} (\frac{k}{k+1} d_M(a_j) + \frac{1}{k+1} d_M(a_i)) \\ & \geq & \sum_{j \in J} d_M(a_j) \end{array}$$

The final step is because $j < i$ so a_j is not longer than a_i. Therefore,

$$\begin{array}{rcl} t \cdot d_M(a_i) & < & t^* \sum_{j \in I} d_M(a_j) + \sum_{j \in J} d_M(a_j) - \sum_{j \in I} c'_j \\ & \leq & t^* \sum_{j \in I} d_M(a_j) \\ & \leq & t^* \cdot (k+1) \cdot d_M(a_i) \end{array}$$

which implies $(k+1)\, t^* > t$, as required. □

5.2.2 Running time analysis

We analyse the running time of the greedy algorithm. Recall that the greedy algorithm sorts the edges in $\{V \times V\} \setminus E$ by increasing length and then processes them in order. Processing an edge e entails a shortest path query. If $d_G(e) > t \cdot d_M(e)$, then the edge e is added to G, otherwise it is discarded.

Our algorithm performs efficient shortest path queries by building and maintaining an all pairs shortest paths (APSP) data structure for each of the graphs G_i. When an edge pq is added to the graph, the data structure updates the length of the shortest path between every pair of points $u, v \in V$. We compute the paths $u \to v$, $u \to p \to q \to v$, and $u \to q \to p \to v$, and choose the minimum length. For a fixed $u, v \in V$, this can be handled in constant time, since all pairwise distances are stored.

Hence, the overall running time of the algorithm is as follows. In preprocessing, we build the APSP data structure in $O(mn + n^2 \log n)$ time. Sorting the edges in $\{V \times V\} \setminus E$ takes $O(n^2 \log n)$ time. Querying whether $d_G(e) > t \cdot d_M(e)$ can be handled in constant time, and there are at most $O(n^2)$ such queries. Updating the APSP data structure takes $O(n^2)$ time, and there are at most $k+1$ updates. Putting this all together yields:

Theorem 13. *Given an integer k, a real number t and a graph G with n vertices and m edges, there is an $O((m + n \log n + kn) \cdot n)$ time algorithm that returns either $t^* \leq t$ or $t^* > \frac{t}{k+1}$.*

5.3 Minimising the Dilation

We return to Problem 1, which is to compute a $(1+\varepsilon)(k+1)$-approximation for the minimum dilation t^*. For any real value t, we can use Theorem 13 to decide whether $t^* \leq t$ or $t^* > \frac{t}{k+1}$. Hence, it remains only to provide some bounded interval that t^* is guaranteed to be in. Once we have such an interval, then we can binary search on an ε-grid of the interval to obtain a $(1+\varepsilon)(k+1)$-approximation.

We compute this interval in two steps. Our first step is to identify a set T of $O(n^4)$ real numbers so that at least one of these numbers is an $O(n)$-approximation of t^*. Our second step is to use the approximate decision algorithm in Theorem 13 to perform a binary search on the set T and yield an $O(nk^2)$-approximation for t^*. The $O(nk^2)$-approximation provides the required interval.

We begin by identifying the set T of $O(n^4)$ real numbers.

Lemma 14. *Define* $T = \{\frac{d_M(u,v)}{d_M(p,q)} : u, v, p, q \in V,\ u \neq v, p \neq q\}$. *Then there exists an element* $t \in T$ *such that* $t \leq t^* \leq n \cdot t$.

Proof. Consider the graph $G^* = (V, E \cup S)$. Let the dilation of t^* be attained by the pair of points $u, v \in V$. Let pq be a longest edge along the shortest path from u to v in G^*. See Figure 5.5.

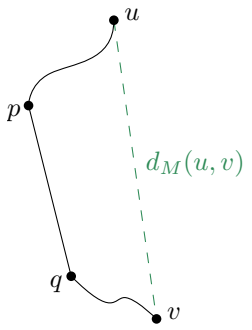


Figure 5.5: The edge pq is a longest edge on the shortest path from u to v.

Recall that $d_{G^*}(u, v)$ is the length of the shortest path from u to v in the graph G^*. The dilation of t^* is attained by the pair of points u, v, which implies $d_{G^*}(u, v) = t^* \cdot d_M(u, v)$. The shortest path from u to v has total length $d_{G^*}(u, v)$ and has at most n edges, where the length of each edge is at most $d_M(p, q)$. This implies $d_M(p, q) \leq d_{G^*}(u, v) \leq n \cdot d_M(p, q)$. But $d_{G^*}(u, v) = t^* \cdot d_M(u, v)$, so this inequality rearranges to give

$$\frac{d_M(p,q)}{d_M(u,v)} \leq t^* \leq n \cdot \frac{d_M(p,q)}{d_M(u,v)},$$

as required. □

Next, we use the approximate decision algorithm in Theorem 13 to binary search the set $T = \{\frac{d_M(u,v)}{d_M(p,q)} : u, v, p, q \in V,\ u \neq v, p \neq q\}$ in order to yield an $O(nk^2)$-approximation. A naïve implementation of the binary search would entail computing and sorting the elements in T, which would require $O(n^4 \log n)$ time. To speed up our algorithm, we avoid

the $O(n^4 \log n)$ preprocessing step, and we do so by using the result of Mirzaian and Arjomandi [127]. The result states that given two sorted lists X and Y each of size n, one can select the i^{th} smallest element of the set $X + Y = \{x + y : x \in X, y \in Y\}$ in $O(n)$ time.

Lemma 15. *There is an $O((m + n \log n + kn) \cdot n \log n)$ time algorithm that computes an $O(nk^2)$-approximation for t^*.*

Proof. In a preprocessing step, construct and sort the sets $X = \{\log(d_M(u, v)) : u, v \in V, u \neq v\}$ and $Y = \{-\log(d_M(p, q)) : p, q \in V, p \neq q\}$. To perform the binary search, select the i^{th} smallest element of $X + Y = \{\log(\frac{d_M(u,v)}{d_M(p,q)}) : u, v, p, q \in V, u \neq v, p \neq q\}$. Reverse the log transformation to obtain the i^{th} smallest element of T. Call this element $t_i \in T$. Apply Theorem 13 to the two dilation values $\frac{2}{3} \cdot t_i$ and $n(k + 1) \cdot t_i$. This returns one of three possibilities:

1. $t^* \leq \frac{2}{3} \cdot t_i$ and $t^* \leq n(k + 1) \cdot t_i$, or
2. $t^* > \frac{2}{3} \cdot \frac{t_i}{k+1}$ and $t^* \leq n(k + 1) \cdot t_i$, or
3. $t^* > \frac{2}{3} \cdot \frac{t_i}{k+1}$ and $t^* > n \cdot t_i$.

The fourth combination cannot occur as it yields a contradiction. Notice that in case one, we have $t^* < t_i$, so the element $t \in T$ satisfying $t \leq t^* \leq n \cdot t$ must be less than t_i. We can continue the binary search over the elements in T that are less than t_i. Similarly, in case three, we have $t^* > n \cdot t_i$, so the element $t \in T$ satisfying $t \leq t^* \leq n \cdot t$ must be greater than t_i. We can continue the binary search over the elements in T that are greater than t_i. In case two we halt, since we have an $O(nk^2)$-approximation for t^*.

We analyse the running time of this algorithm. Sorting the sets X and Y takes $O(n^2 \log n)$ time. For each of the $O(\log n)$ binary search step, selecting the i^{th} element of $X + Y$ takes $O(n^2)$ time [127]. For each of the $O(\log n)$ binary search steps, applying Theorem 13 takes $O((m + n \log n + kn) \cdot n)$. Putting this all together yields the stated running time. □

Finally, we apply a multiplicative $(1 + \varepsilon)$-grid to the $O(nk^2)$-approximation to yield an $(1 + \varepsilon)(k + 1)$-approximation.

Theorem 16. *For any fixed $\varepsilon > 0$, there is an $O((m+n \log n+kn) \cdot n \log n)$ time algorithm that computes a $(1 + \varepsilon)(k + 1)$-approximation for t^*.*

To simplify the running time, we note that if $k \geq n - 1$, then adding the minimum spanning tree to any graph makes it an n-spanner, which is a $(k+1)$-approximation for the minimum dilation. Plugging in $k < n - 1$ and $m \leq n^2$ into Theorem 16 yields:

Theorem 17. *For any fixed $\varepsilon > 0$, there is an $O(n^3 \log n)$ time algorithm that computes an $(1 + \varepsilon)(k + 1)$-approximation for t^*.*

5.4 Approximation factor no better than $(1 - \varepsilon)(k + 1)$

We provide a construction to show that the algorithms in Theorem 13 and Theorem 16 cannot yield an approximation factor better than $(1 - \varepsilon)(k + 1)$.

Theorem 18. *For any $k \geq 1$ and $\varepsilon > 0$, there exists a graph so that for any $t \leq (1-\varepsilon)(k+1) \cdot t^*$, the greedy algorithm in Definition 3 adds at least $k + 1$ edges to the graph.*

Proof. Fix h to be a small positive constant less than $\frac{1}{t}$, and fix a constant h' to be arbitrarily small relative to h. We construct the graph G shown in Figure 5.6.

Let the vertices of G be

$$\begin{array}{rcll} a_1 & = & (0, 2h) & \\ b_i & = & (1, 2ih) & \forall\, 1 \le i \le k \\ c_i & = & (2, 2ih) & \forall\, 1 \le i \le k \\ d_i & = & (k+3+i, 2ih) & \forall\, 1 \le i \le k \\ e_i & = & (k+3+i, (2i+1)h) & \forall\, 1 \le i \le k \\ f_i & = & (2, (2i+1)h - h') & \forall\, 1 \le i \le k \\ g_i & = & (1, (2i+1)h) & \forall\, 1 \le i \le k \\ y_1 & = & (0, (2k+1)h) & \\ z_1 & = & (0, 3h) & \end{array}$$

The graph G is a path between these vertices. The edges of G are between consecutive elements in the sequence $a_1, b_1, c_1, d_1, e_1, f_1, g_1, b_2, c_2, \ldots, f_k, g_k, y_1, z_1$. See Figure 5.6.

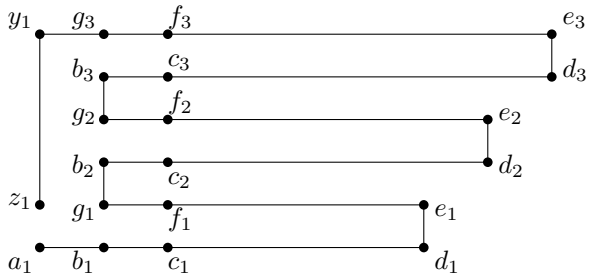


Figure 5.6: The construction for $k = 3$.

The pairs of points with the largest dilation are (a_1, z_1), (b_i, g_i) and (c_i, f_i). We can pick a small enough value of h so that the dilation of all other pairs are relatively insignificant. The optimal k edges to add are (b_i, g_i) for all $1 \le i \le k$. After adding these k edges, the pairs of points with the largest dilation are (a_1, z_1) and (c_i, f_i). Of these, the pair of points (a_1, z_1) realises the maximum dilation, which is $t^* = (2 + (4k+1)h)/h \approx 2/h$.

Now let us run the greedy spanner construction for some $t \le (1-\varepsilon)(k+1) \cdot t^*$. All pairs of points (a_1, z_1), (b_i, g_i) and (f_i, c_i) start off with dilation greater than $2(k+4)/h$. But $2(k+4)/h = (k+4) \cdot 2/h > (k+3) \cdot t^* > t$, where the second inequality is true for sufficiently small values of h. The pairs of points with highest dilation are (a_1, z_1), (b_i, g_i) and (f_i, c_i), and the edges connecting these pairs of points satisfies $d_{G_i}(e) > t \cdot d_M(e)$. The shortest of these edges will be added first by the greedy t-spanner construction. The pairs (c_i, f_i) have distance $h - h'$, making the edge between them the shortest and first to be considered by the greedy algorithm. Adding an edge between (c_i, f_i) does not reduce the dilation of the other pairs of points (c_j, f_j). Therefore, the greedy spanner construction first adds the edges (c_i, f_i) for all $1 \le i \le k$.

After adding (c_i, f_i) for all $1 \le i \le k$, the dilation between the pair of points a_1 and z_1 is now $(2k+2+(4k+1)h)/h$. But $(2k+2+(4k+1)h)/h = (2k+2+(4k+1)h)/(2+(4k+1)h) \cdot t^* > (1-\varepsilon)(k+1) \cdot t^*$ for sufficiently small values of h relative to ε. Therefore, the greedy t-spanner construction must add the edges (c_i, f_i) for all $1 \le i \le k$ plus at least one additional edge, so it adds at least $k+1$ edges in total. □

Our construction shows that in Theorem 13 we cannot hope to obtain a bound that is much better than $t^* > \frac{t}{k+1}$. Similarly, in Theorem 16, our construction implies that the algorithm may continue searching for higher dilation values up until $(1-\varepsilon)(k+1)\cdot t^*$. Therefore, we cannot hope to obtain a much better approximation ratio than $(1+\varepsilon)(k+1)$ with our algorithm.

5.5 The Bottleneck Algorithm

Farshi et al. [81] studied the special case where $k = 1$. They achieved a 3-approximation by adding the *bottleneck edge*, which is an edge between a pair of points that achieves the maximum dilation. They also provided a generalisation of their algorithm for $k > 1$. The generalisation consists of k stages. In each stage, the dilation of the graph is computed, and a pair of points that achieves the maximum dilation is identified. Then an edge is added between those pair of points. Formally, given an initial metric graph G, and an integer k:

Definition 19. *Let $G_0 = G$, and for $1 \leq i \leq k$, let $G_i = G_{i-1} \cup b_i$ where b_i is an edge between the pair of points that achieves the maximum dilation of G_{i-1}.*

Farshi et al. [81] conjectured that the dilation of the augmented graph G_k may be reasonable approximation for the dilation of the optimal graph G^*. We provide a negative result that states that their algorithm cannot yield an approximation factor better than 2^k.

Theorem 20. *For any $k \geq 1$, there exists a initial graph G where bottleneck algorithm in Definition 19 yields a graph G_k with dilation 2^k times that of the dilation of the optimal graph G^*.*

Proof. Fix h to be a small constant. Let the vertices of G be

$$\begin{array}{rcll} x_0 & = & (-1, h) & \\ y_i & = & (0, 2^i h) & \forall \quad 1 \leq i \leq k+1 \\ z_i & = & (2^{i-1}, 3 \cdot 2^{i-1} h) & \forall \quad 1 \leq i \leq k \\ x_1 & = & (-1, 2^{k+1} h + h) & \end{array}$$

Join the vertices together to form a path $x_0, y_1, z_1, y_2, z_2, \ldots, y_k, z_k, y_{k+1}, x_1$. See Figure 5.7.

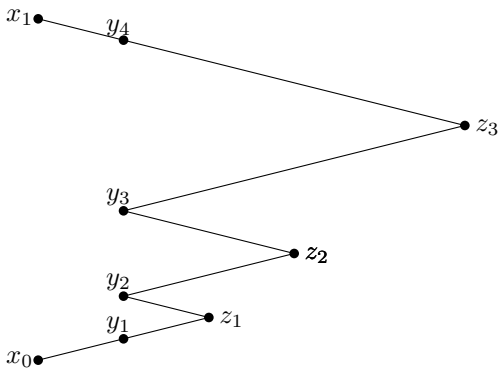


Figure 5.7: The construction for $k = 3$.

It is straightforward to check that all edges in G have gradient $\pm h$. Since h is a small constant, all edges are almost horizontal. Therefore, the pairs of vertices with maximum dilation are those that are vertically above one another, in other words, the pairs (x_0, x_1), or (y_i, y_{i+1}) for $1 \leq i \leq k$. In particular, all the pairs listed have a dilation value of $\sqrt{1+h^2}/h$.

Since (x_0, x_1) is one of the pairs of vertices with maximum dilation, we can choose the first bottleneck edge b_1 to connect these two points. It is easy to check that since the distance in the graph between (x_0, x_1) is twice the distance of any other pair (y_i, y_{i+1}), adding the first bottleneck edge does not reduce the dilation of any of the pairs (y_i, y_{i+1}). Inductively, we can show that for $i \geq 2$, $b_i = (y_{k-i+2}, y_{k-i+3})$ is the i^{th} bottleneck edge added. This is because it initially had the maximum dilation of $\sqrt{1+h^2}/h$, and adding the bottleneck edges $b_1, b_2, \ldots b_{i-1}$ did not reduce its dilation factor. Finally, after adding $b_1, \ldots b_k$, the dilation of the augmented graph G_k is still $\sqrt{1+h^2}/h$ and is attained by (y_1, y_2).

The optimal placements of k edges would be the edges $(y_1, y_2), \ldots (y_k, y_{k+1})$. Under this placement of k edges, the maximum dilation value is attained by (x_0, x_1), and is $2\sqrt{1+h^2}/(2^{k+1} \cdot h) = \sqrt{1+h^2}/(2^k \cdot h)$. Hence, the augmented graph G_k has a dilation of 2^k times the dilation of the optimal graph G^*.

Note that in our construction, ties are broken adversarially when choosing the bottleneck edge to add. If we would like to lift the requirement on the adversarial choice of which bottleneck edge to add, we can perturb x_0 and x_1 vertically towards each other, which guarantees that (x_0, x_1) is the first bottleneck edge to be added. We can do so similarly for the other bottleneck edges. □

5.6 Conclusion

In Farshi et al. [81] it was conjectured that generalising their algorithm to any positive integer k may provide a reasonable approximation algorithm. In Section 5.5, we showed an $\Omega(2^k)$ lower bound for the approximation factor. We obtained the first positive result for the general case. Our approximation algorithm runs in $O(n^3 \log n)$ time and guarantees an $O(k)$-approximation factor.

Two obvious open problems are to develop an algorithm with a better approximation factor, or to show an inapproximability bound.

Chapter 6

Bicriteria approximation for minimum dilation graph augmentation

6.1 Introduction

Let G be a graph embedded in a metric space M. Let $V(G)$, $E(G)$ be the vertices and edges of G. For vertices $u, v \in V(G)$, define $d_M(u, v)$ to be the metric distance between points $u, v \in M$, and define $d_G(u, v)$ to be the shortest path distance between vertices $u, v \in G$. The *dilation* or *stretch* of G is the minimum $t \in \mathbb{R}$ so that for all $u, v \in V(G)$, we have $d_G(u, v) \leq t \cdot d_M(u, v)$.

Dilation measures the quality of a network in applications such as transportation and communication networks. For now, we restrict our attention to the special case of low dilation trees.

Problem 1. *Given a set of n points V embedded in a metric space M, compute a spanning tree of V with minimum dilation.*

Problem 1 is known across the theory community, as either the minimum dilation spanning tree problem [15, 26, 56], the tree spanner problem [47, 82, 87] or the minimum maximum-stretch spanning tree problem [76, 121, 136]. The problem is NP-hard even if M is an unweighted graph metric [47] or the Euclidean plane [56]. Problem 1 is closely related to tree embeddings of general metrics [17], and has applications to communication networks and distributed systems [136].

The approximability of Problem 1 is an open problem stated in several surveys and papers [56, 77, 136], and is a major obstacle towards constructing low dilation graphs with few edges [15, 108]. The minimum spanning tree is an $O(n)$-approximation [77] for Problem 1, but no better result is known. Only in the special case where M is an unweighted graph is there an $O(\log n)$-approximation [76].

Obstacle 2. *Is there an $O(n^{1-\varepsilon})$-approximation algorithm for Problem 1, for any $\varepsilon > 0$?*

If we no longer restrict ourselves to trees, we can shift our attention to spanners, which are low dilation sparse graphs. An advantage of spanners over minimum dilation trees is

that spanners are not affected by Obstacle 2. Spanners obtain significantly better dilation guarantees, at the cost of adding slightly more edges. The trade-off between sparsity and dilation in spanners has been studied extensively [13, 60, 85, 119]. For an overview of the rich history and multitude of applications of spanners, see the survey on graph spanners [7] and the textbook on geometric spanners [134].

Spanner constructions focus on the initial design of the network. However, networks tend to improve over time. In this chapter, we focus on the improvement step. Given a graph and a budget k, which k edges do we add to the graph to minimise its dilation?

Problem 3. *Given a positive integer k and a metric graph G, compute a set S of k edges so that the dilation of the graph $G' = (V(G), E(G) \cup S)$ is minimised. Note that $S \subseteq V(G) \times V(G)$.*

Narasimham and Smid [134] stated Problem 3 as one of twelve major open problems in their reference textbook. Despite this, there were no breakthroughs for over a decade. Gudmundsson and Wong [108] provided the first positive result for Problem 3, an $O(k)$-approximation algorithm that runs in $O(n^3 \log n)$ time. One downside is that their approximation factor is linear in k. However, since Problem 1 is a special case of Problem 3, Obstacle 2 applies to Problem 3 as well.

Obstacle 4. *One cannot obtain an $O(k^{1-\varepsilon})$-approximation algorithm for Problem 3 for any $\varepsilon > 0$, without first resolving Obstacle 2.*

One way to circumvent Obstacle 4 is to consider a bicriteria approximation. An advantage of bicriteria approximations is that we can show significantly better dilation guarantees, at the cost of adding slightly more edges.

The goal of our bicriteria problem is to investigate the trade-off between sparsity and dilation. We define the sparsity parameter f to be the number of edges added by our algorithm divided by k. We define the dilation parameter g to be the dilation of our algorithm (which adds fk edges) divided by the dilation of the optimal solution (which adds k edges).

Problem 5. *Given a positive integer k, a metric graph G, sparsity $f \in \mathbb{R}$ and dilation $g \in \mathbb{R}$, construct a set S of fk edges so that the dilation of the graph $G' = (V(G), E(G) \cup S)$ is at most gt^*, where t^* is the minimum dilation in Problem 3. Note that $S \subseteq V(G) \times V(G)$.*

We define an (f, g)-bicriteria approximation to be an algorithm for Problem 5 that achieves sparsity f and dilation g.

6.1.1 Contributions

Our main result is a $(2\sqrt[r]{2}\, k^{1/r}, 2r)$-bicriteria approximation for Problem 5 that runs in $O(n^3 \log n)$ time, for all $r \geq 1$. In other words, if t^* is the minimum dilation after adding any k edges to a graph, then our algorithm adds $O(k^{1+1/r})$ edges to the graph to obtain a dilation of $2rt^*$. Our dilation guarantees are significantly better than the previous best result [108], at the cost of adding slightly more edges. For example, if $r = \log(2k)$ we obtain a $(4, 2\log(2k))$-bicriteria approximation. See Table 6.1.

Our approach is to use the greedy spanner construction. The greedy spanner is among the most extensively studied spanner constructions [13, 60, 85, 119]. Therefore, it is perhaps unsurprising that greedy spanner can be used for Problem 5. Nonetheless, we believe that our result shows the utility and versatility of the greedy spanner.

Our main technical contribution is our analysis of the greedy spanner. Our main insight is to construct an auxilliary graph, which we call the girth graph, and to argue that the approximation ratio is bounded by the length of the shortest cycle in the girth graph. Moreover, our analysis of the greedy spanner is tight, up to constant factors. In particular, assuming the Erdős girth conjecture, there is a graph class for which our algorithm is an $(\Omega(k^{1/r}), 2r+1)$-bicriteria approximation.

The restriction $r \geq 1$ is necessary in our main result. We prove that it is NP-hard to obtain a $(\text{poly}(k), 2-\varepsilon)$-bicriteria approximation, for any $\varepsilon > 0$. Finally, we use ideas from our proof of NP-hardness to provide a $(2k \log n, 1)$-bicriteria approximation.

Our results are summarised in Table 6.1. For a technical overview of our results, see Section 6.2.

Sparsity (f)	Dilation (g)	Complexity	Reference
1	$k+1$	$O(n^3 \log n)$	Gudmundsson and Wong [108]
$2+\varepsilon$	$O_\varepsilon(\log(k))$	$O(n^3 \log n)$	$r = O_\varepsilon(\log(k))$ in Theorem 6
4	$2\log(2k)$	$O(n^3 \log n)$	$r = \log(2k)$ in Theorem 6
$2^{1+\varepsilon}\ k^\varepsilon$	$2/\varepsilon$	$O(n^3 \log n)$	$r = 1/\varepsilon$ in Theorem 6
$2\sqrt{2}\ \sqrt{k}$	4	$O(n^3 \log n)$	$r = 2$ in Theorem 6
$4k$	2	$O(n^3 \log n)$	$r = 1$ in Theorem 6
Theorem 6 is tight under the Erdős girth conjecture			Theorem 7
$\text{poly}(k)$	$2-\varepsilon$	NP-hard	Theorem 8
$2k \log n$	1	$O(n^6)$	Theorem 9

Table 6.1: The trade-off between sparsity f and dilation g in bicriteria approximations for Problem 5. Note that $O_\varepsilon(\cdot)$ hides dependence on ε.

6.1.2 Related work

Most of the work on Problem 3 focuses on the special case where one edge is added. Farshi, Giannopoulos and Gudmundsson [81] provide an $O(n^4)$ time algorithm as well as an $O(n^3)$ time 3-approximation. Wulff-Nilsen [166] present an $O(n^3 \log n)$ time algorithm. Luo and Wulff-Nilsen [122] improves the space requirement to linear. Aronov et al. [14] provide a nearly-linear time algorithm in the special case where the graph is a simple polygon and an interior point.

A variant of Problem 3 is to add k edges to a graph to minimise the diameter instead of the dilation. Frati, Gaspers, Gudmundsson and Mathieson [88] provide a fixed parameter tractable 4-approximation for the problem. Several special cases have been studied. Demaine and Zadimoghaddam [64] consider adding k edges of length δ, where δ is small relative to the diameter. Große et al. [96] present nearly-linear time algorithms for adding one edge to either a path or a tree in order to minimise its diameter. Follow up papers improve the running time of the algorithm for paths [157] and for trees [23, 159]. Another variant is to add k edges to a graph to minimise the radius. Gudmundsson, Sha and Yao [101] provide a 3-approximation for adding k edges to a graph to minimise its radius. The problem of

adding one edge to minimise the radius of paths [115, 158] and trees [99] has also been studied.

A problem closely related to Problem 1 is to compute minimum dilation graphs. In his Master's thesis, Mulzer [130] studies minimum dilation triangulations for the regular n-gon. Eppstein and Wortman [78] provide a nearly-linear time algorithm to compute a minimum dilation star. Giannopoulos, Knauer and Marx [92] prove that, given a set of points, it is NP-hard to compute a minimum dilation tour or a minimum dilation path. Aronov et al. [15] show that one can construct a graph with $n-1+k$ edges and dilation $O(n/(k+1))$.

Our algorithm for Problem 5 uses the greedy spanner, which is among the most extensively studied spanner constructions. In general metrics, Althöfer et al. [13] show that the greedy $(2k-1)$-spanner has $O(n^{1+1/k})$ edges. In d-dimensional Euclidean space, Das, Heffernan and Narasimhan [60] show that the greedy $(1+\varepsilon)$-spanner has $O(n\varepsilon^{-2d})$ edges, which Le and Solomon [119] improves to $O(n\varepsilon^{-d+1})$ edges. In both cases, the sparsity-dilation trade-off is optimal [85, 119].

6.2 Technical overview

We divide our technical overview into six subsections. In Section 6.2.1, we summarise the previous algorithm of Gudmundsson and Wong [108]. In Section 6.2.2, we give an overview of our main result, that is, our $(2\sqrt[r]{2}\ k^{1/r}, 2r)$-bicriteria approximation for all $r \geq 1$. In Section 6.2.3, we present the main ideas for proving our analysis is tight, assuming the Erdős girth conjecture. In Section 6.2.4, we summarise our proof that it is NP-hard to obtain a $(\text{poly}(k), 2-\varepsilon)$-bicriteria approximation, for any $\varepsilon > 0$. In Section 6.2.5, we present a $(2k\log n, 1)$-bicriteria approximation. In Section 6.2.6, we summarise the structure of the remainder of the chapter.

6.2.1 Previous algorithm of [108]

Gudmundsson and Wong's [108] algorithm constructs the greedy spanner, which is among the most extensively studied spanner constructions. The only modification to the algorithm is: the traditional greedy t-spanner takes as input a set of vertices, but the modified greedy t-spanner takes as input a graph.

The greedy t-spanner construction has two steps. First, all edges that are not in the initial graph are sorted by their length. Second, the edges are processed from shortest to longest. A processed edge uv is added if $d_G(u,v) > d_M(u,v)$, otherwise the edge uv is not added.

For Problem 3, Gudmundsson and Wong's [108] show, in their main lemma, that if the greedy t-spanner adds at least $k+1$ edges, then $t \leq (k+1)\,t^*$. Here, t^* is the minimum dilation if k edges are added to our graph. Using this lemma, they then perform a binary search over a multiplicative $(1+\delta)$-grid for a $t \in \mathbb{R}$ such that the greedy $(1+\delta)\,t$-spanner adds at most k edges, but the greedy t-spanner adds at least $k+1$ edges. Then the greedy $(1+\delta)\,t$-spanner adds at most k edges and $(1+\delta)\,t \leq (1+\delta)(k+1)\,t^*$, so $(1+\delta)\,t$ is a $(1+\delta)(k+1)$-approximation of t^*.

Next, we briefly summarise the proof that if $k+1$ edges are added by the greedy algorithm, then $t \leq (k+1)\ t^*$. In Lemma 2 of [108], the authors use the $k+1$ greedy edges to construct a set of $k+1$ vectors in a k-dimensional vector space. They define I to be a linearly dependent subset of the $k+1$ vectors. In Theorem 5 of [108], the authors use

the linear dependence property of I to prove that $t \leq |I| \cdot t^*$. Since $|I| \leq k+1$, they obtain $t \leq (k+1)\ t^*$. Unfortunately, the vector space approach of [108] fails extend to sublinear dilation factors g in Problem 5, even if we allow polynomial sparsity values.

6.2.2 Greedy bicriteria approximation

Our algorithm is the same as the one in [108]. Our main difference is in the analysis of the greedy t-spanner, in particular, in our main lemma.

For Problem 5, we show, in our main lemma, that if the greedy t-spanner adds at least $fk+1$ edges, then $t \leq gt^*$. Then, we apply the same binary search procedure to find a $t \in \mathbb{R}$ where the greedy $(1+\delta)\,t$-spanner adds at most fk edges, but the greedy t-spanner adds at least $fk+1$ edges. Then $(1+\delta)\,t$ is an $(f,(1+\delta)g)$-bicriteria approximation of t^*.

In Table 6.1, we omit the factor of $(1+\delta)$ from the second column for three reasons — first, for clarity and ease of comparison, second, to have the dilation factor reflect their respective main lemmas instead of the less interesting multiplicative $(1+\delta)$-grid, and third, because δ has no impact on the sparsity parameter, and has minimal impact on the running time (see Theorem 6). We make similar omissions in Section 6.1.1 and the abstract.

Next, we briefly summarise our new proof that if $fk+1$ edges are added, then $t \leq gt^*$. In order to extend our analysis to sublinear dilation factors g, we abandon the vector space approach of [108]. Our main idea is to construct an auxiliary graph, which we call the girth graph. The girth graph is an unweighted graph with $2k$ vertices and $fk+1$ edges. Instead of defining I to be a linearly dependent subset, we define I to be the shortest cycle in the girth graph. We use a classical result in graph theory to choose the values $f = 2\sqrt[r]{2}\ k^{1/r}$ and $g = 2r$, so that $|I| \leq g$. Our final step is to carefully prove $t \leq |I| \cdot t^*$, using the cycle property of I. Therefore, $t \leq gt^*$.

Putting this all together, we obtain Theorem 6. For a full proof, see Section 6.3.

Theorem 6. *For all $r \geq 1$, there is an $(f,(1+\delta)g)$-bicriteria approximation for Problem 5 that runs in $O(n^3(\log n + \log \frac{1}{\delta}))$ time, where*

$$f = 2\sqrt[r]{2}\ k^{1/r} \quad \textit{and} \quad g = 2r.$$

6.2.3 Greedy analysis is tight

Our analysis in Theorem 6 is tight. This means one cannot obtain better bounds (up to constant factors) using the greedy spanner. Our proof assumes the Erdős girth conjecture [79].

The girth of an unweighted graph is defined as the number of edges in its shortest cycle. In the proof of Theorem 6, we cite a classical result stating that a graph with n vertices and at least $n^{1+1/r}$ edges has girth at most $2r$. The Erdős girth conjecture states that there are graphs with n vertices, at least $\Omega(n^{1+1/r})$ edges and girth $2r+2$. Several conditional lower bounds have been shown under the Erdős girth conjecture, namely, the sparsity-dilation trade-off of the greedy spanner [13], and the space requirement of approximate distance oracles [154].

We summarise our construction that proves that our analysis is tight. Assuming the Erdős girth conjecture, there exists a graph H with $n = k+1$ vertices, $m = \Omega(n^{1+1/r})$ edges, and girth $2r+2$. We construct a graph G so that if we run the algorithm in Theorem 6, the girth graph of G would be H. We use the properties of H to show that, if there are k edges that can be added to G so that the resulting dilation is t^*, then if we add $m-1$ edges to G using the greedy t-spanner construction, the resulting dilation is at least $(2r+1)\ t^*$.

Putting this all together, we obtain Theorem 7. For a full proof, see Section 6.4.

Theorem 7. *For all $r \geq 1$, assuming the Erdős girth conjecture, there is a graph class for which the algorithm in Theorem 6 returns an (f, g)-bicriteria approximation, where*

$$f = \Omega(k^{1/r}) \quad \textit{and} \quad g = 2r + 1.$$

6.2.4 Set cover reduction

Next, we show that the restriction $r \geq 1$ is necessary in Theorem 6. Recall that Theorem 6 states that there is a $(2\sqrt[r]{2}\ k^{1/r}, (1+\delta)\,2r)$-bicriteria approximation algorithm for all $r \geq 1$. We prove that it is NP-hard to obtain a $(\text{poly}(k), 2-\varepsilon)$-bicriteria approximation for any $\varepsilon > 0$. Our proof is a reduction from set cover.

We summarise our construction of the Problem 5 instance. We show that every set cover instance can be reduced to a Problem 5 instance. We represent an element with a pair of points, and we represent a set with a triple of points. In our Problem 5 instance, we add edges to connect either the pairs or the triples. We show via an exchange argument that it is always better to connect the triples. Connecting a triple corresponds to choosing a set, which lowers the dilation of all elements in that set to below the threshold value. Finally, we show that a $(\text{poly}(k), 2-\varepsilon)$-bicriteria approximation for our Problem 5 would solve set cover within an approximation factor of $\text{poly}(k)$. However, it is NP-hard to approximate set cover to within a factor of $(1-\alpha)\log n$, where $\alpha > 0$ is a constant and n is the size of the set cover instance [66]. Moreover, k is a constant in the NP-hard instance of [66], so $\text{poly}(k) \ll (1-\alpha)\log n$.

Putting this all together, we obtain Theorem 8. For a full proof, see Section 6.5.

Theorem 8. *For all $\varepsilon > 0$, it is NP-hard to obtain an (f, g)-bicriteria approximation for Problem 5, where*

$$f = \text{poly}(k) \quad \textit{and} \quad g = 2 - \varepsilon.$$

6.2.5 Set cover algorithm

Finally, we provide a $(2k \log n, 1)$-bicriteria approximation that runs in $O(n^6)$ time. Our main idea is to formulate the problem into a set cover instance, and then to apply an $O(\log n)$-approximation algorithm for set cover [57].

We state our algorithm. Let $t \in \mathbb{R}$. Define G_e to be the graph if an edge e is added to G. Define $S_e = \{(u, v) : d_{G_e}(u, v) \leq t \cdot d_M(u, v)\}$. Apply the algorithm of [57] on $\{S_e : e \in V(G) \times V(G)\}$ to obtain a set cover S. We claim that if $|S| > 2k^2 \log n$, then $t \leq t^*$. Finally, we perform binary search in the same way as [108] to obtain a $(2k \log n, 1)$-bicriteria approximation.

To prove correctness, it remains to show our claim. We show the contrapositive. If $t > t^*$, from the definition of t^* there exists k edges that can be added to G to make it a t-spanner. Consider a clique with vertices that are the endpoints of the k edges. Adding these $2k^2$ edges to the graph would also make it a t-spanner. Moreover, each t-path, that is, a (u, v)-path with length at most $t \cdot d_M(u, v)$, uses at most one edge in the clique. Therefore, the union of the sets S_e, where e is an edge in the clique, forms a set cover over all pairs of vertices (u, v). The optimal solution of the set cover instance is at most $2k^2$, so the algorithm of [57] returns a set S such that $|S| \leq 2k^2 \log n$.

Putting this all together, we obtain Theorem 9.

Theorem 9. *There is an (f, g)-bicriteria approximation for Problem 5, where*

$$f = 2k \log n \quad \textit{and} \quad g = 1.$$

This completes the overview of the main results of this chapter.

6.2.6 Structure of chapter

The structure of the remainder of our chapter is summarised in Table 6.2.

	Reference	Proof
$(2\sqrt[r]{2}\ k^{1/r}, (1+\delta)\,2r)$-bicriteria approximation	Theorem 6	Section 6.3
Theorem 6 analysis is tight	Theorem 7	Section 6.4
$(\text{poly}(k), 2-\varepsilon)$-bicriteria approximation is NP-hard	Theorem 8	Section 6.5

Table 6.2: Proofs of Theorems 6, 7, 8 can be found in Sections 6.3, 6.4, 6.5 respectively.

6.3 Greedy bicriteria approximation

In this section, we will prove Theorem 6. We restate the theorem for convenience.

Theorem 6. *For all $r \geq 1$, there is an $(f, (1+\delta)g)$-bicriteria approximation for Problem 5 that runs in $O(n^3(\log n + \log \frac{1}{\delta}))$ time, where*

$$f = 2\sqrt[r]{2}\ k^{1/r} \quad \textit{and} \quad g = 2r.$$

Recall from Section 6.1 that the vertices and edges of G are $V(G)$ and $E(G)$ respectively. Let $e \in V(G) \times V(G)$ be an edge not necessarily in $E(G)$. Let $d_M(e)$ denote the length of the edge e in the metric space M and let $d_G(e)$ denote the shortest path distance between the endpoints of e in the graph G. Consider a minimum dilation graph G^* after adding an optimal set S^* of k edges to G. Let t^* be the dilation of G^*.

Recall from Section 6.2 that our approach is to use the greedy t-spanner construction. We formalise the construction in the definition below.

Definition 10. *Define $G_0 = G$, and for $i \geq 1$, define $G_i = G_{i-1} \cup \{a_i\}$, where a_i is the shortest edge in $V(G) \times V(G)$ satisfying $d_{G_{i-1}}(a_i) > t \cdot d_M(a_i)$. The process halts if no edge a_i exists.*

We have two cases: either the process halts after adding at most fk edges, or after adding more than fk edges. If more than fk edges are added, we show a dilation bound on t. In particular, Lemma 15 states that if there is an edge a_i satisfying $d_{G_{i-1}}(a_i) > t \cdot d_M(a_i)$ for all $1 \leq i \leq fk+1$, then we have the dilation bound $t \leq gt^*$. We will specify the parameters $f, g \geq 1$ at a later point in this section.

Our approach is to construct an auxilliary graph H, which we will also refer to as the girth graph. Define the vertices of H to be $V(H) = \{v_1, \ldots, v_{2k}\}$. Each vertex in $V(H)$ corresponds to an endpoint of an edge in the optimal set of k edges S^*. In particular, let $S^* = \{s_1, \ldots, s_k\}$, and let $v_{2i-1}, v_{2i} \in V(H)$ correspond to the endpoints of s_i. Define the

edges of H to be $E(H) = \{e_1, \ldots, e_{fk+1}\}$. We will describe the procedure for constructing each edge e_i.

Consider the greedy edge a_i, see Figure 6.1. Define $\delta_{G^*}(a_i)$ to be the shortest path between the endpoints of a_i in G^*, shown in grey in Figure 6.1. Note that $\delta_{G^*}(a_i)$ denotes a path, whereas $d_{G^*}(a_i)$ denotes a length. Suppose that there are no edges in S^* along the path $\delta_{G^*}(a_i)$, for some $1 \leq i \leq fk + 1$. Then,

$$t^* \cdot d_M(a_i) \geq d_{G^*}(a_i) = d_G(a_i) \geq d_{G_i}(a_i) > t \cdot d_M(a_i),$$

so $t < t^* \leq gt^*$, which would already imply Lemma 15.

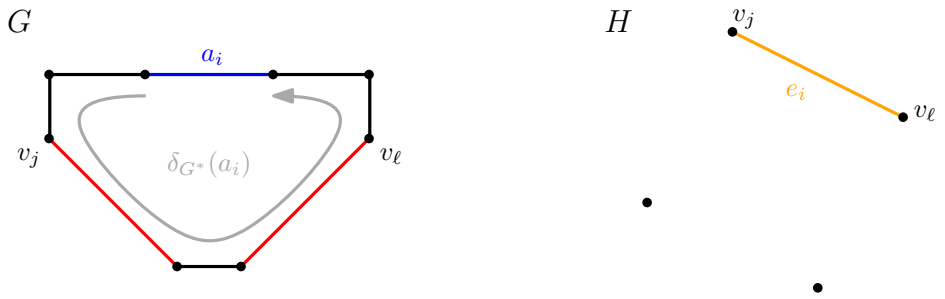


Figure 6.1: Left: The graph G (black), the greedy edge a_i (blue), the path $\delta_{G^*}(a_i)$ (grey), and the edges $\delta_{G^*}(a_i) \cap S^*$ (red). Right: The girth graph H and the edge e_i (orange).

Therefore, we can assume that $\delta_{G^*}(a_i)$ contains at least one edge in S^*, for every $i = 1, \ldots, fk + 1$. Consider the edges $\delta_{G^*}(a_i) \cap S^*$, shown in red in Figure 6.1. Choose a direction for the path $\delta_{G^*}(a_i)$, sort the list of endpoints of $\delta_{G^*}(a_i) \cap S^*$ with respect to this direction, and let the first and last endpoints in the sorted list be v_j and v_ℓ. Another way to characterise v_j (respectively v_ℓ) is that v_j is an endpoint of an edge in $\delta_{G^*}(a_i) \cap S^*$ so that the shortest path between v_j and one of the endpoints of a_i contains no edges in S^* (respectively the other endpoint of a_i). Finally, we define e_i to be the edge in H connecting v_j to v_ℓ. Note that e_i is an undirected, unweighted edge, shown in orange in Figure 6.1. This completes the construction of H.

In Figure 6.2, we provide a more complete example of a graph G and its girth graph H. The optimal set of $k = 4$ edges is $S^* = \{s_1, s_2, s_3, s_4\}$, which is shown in red. The five greedy edges $\{a_1, a_2, a_3, a_4, a_5\}$ are shown in blue. The first and last endpoints of $\delta_{G^*}(a_1) \cap S^*$ are v_1 and v_3, so $e_1 = v_1v_3$. Similarly, $e_2 = v_3v_5$, $e_3 = v_5v_7$, $e_4 = v_1v_7$ and $e_5 = v_5v_6$.

Next, define J to be the shortest cycle in H, and define $I = \{j : e_j \in J\}$. Recall that the girth of a graph is the length of its shortest cycle. Therefore, the girth of H is $|J| = |I|$.

We use a classical result in graph theory to set the parameters f and g.

Lemma 11. *A graph with n vertices and at least $n^{1+1/r}$ edges has girth at most $2r$.*

Proof. The lemma is a classical result [24]. The survey on graph spanners by Ahmed et al. provides a self-contained proof, see Proposition 2.3 in [7]. □

With Lemma 11 in mind, we set $f = 2\sqrt[r]{2}\ k^{1/r}$ and $g = 2r$, where $r \geq 1$. Then, the graph H has $2k$ vertices, $(2k)^{1+1/r} + 1$ edges, and therefore H has girth $|I| \leq g = 2r$.

Having defined the girth graph H, the indices I, and the parameters f and g, the next step is to prove Lemmas 12, 13 and 14. The three lemmas will be combined to prove our

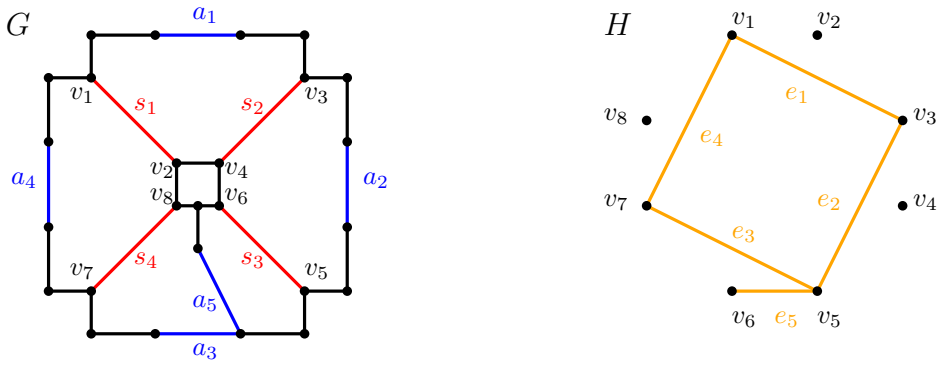


Figure 6.2: Left: The graph G (black), the optimal edges $s_1, \ldots, s_4$ (red), and the greedy edges $a_1, \ldots, a_5$ (blue). Right: The girth graph H has edges $e_1, \ldots, e_5$ (orange) and a girth of 4.

main lemma, Lemma 15, which states that $t \leq gt^*$. We start with Lemma 12, in which we construct a path.

Lemma 12. *Let $i = \max I$. There is a path in H between the endpoints of a_i using only edges in*

$$\{G \cap \delta_{G^*}(a_j) : j \in I\} \cup \{a_j : j \in I \setminus \{i\}\}.$$

Proof. Recall that $J = \{e_j : j \in I\}$ is a cycle in H. After removing the edge e_i, there is still a path in $J \subseteq H$ between the endpoints of e_i. Let the vertices along this path be $w_1, \ldots, w_m$, where $e_i = w_1 w_m$, and $w_\ell w_{\ell+1} \in J$ for all $\ell = 1, \ldots, k-1$. In Figure 6.3, the path $w_1, \ldots, w_m$ is shown in orange. Let the endpoints of a_i be $a_i(0)$ and $a_i(1)$. Our approach is to use the path $w_1, \ldots, w_m \subset H$ to construct a path between $a_i(0)$ and $a_i(1)$ that only uses edges in $\{G \cap \delta_{G^*}(a_j) : j \in I\} \cup \{a_j : j \in I \setminus \{i\}\}$.

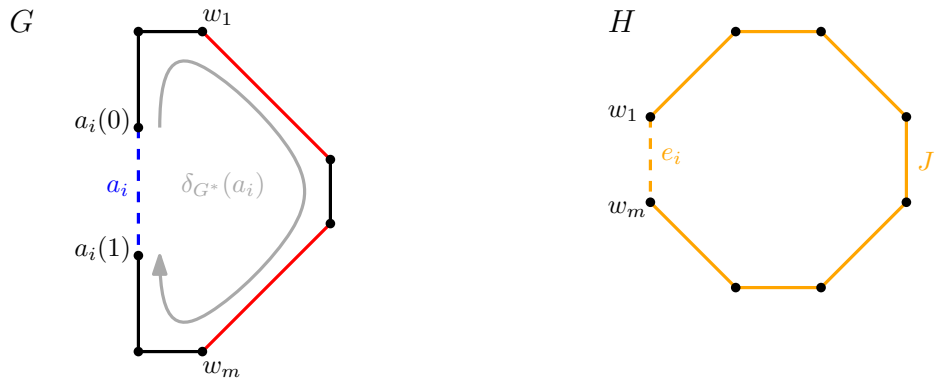


Figure 6.3: Left: The graph G (black), the greedy edge a_i (blue), the path $\delta_{G^*}(a_i)$ (grey), and the edges $\delta_{G^*}(a_i) \cap S^*$ (red). Right: The girth graph H, the cycle J (orange), and edge e_i (dashed).

First, we consider the edge $e_i = w_1 w_m$. Recall from the definition of $V(H)$ that w_1 and w_m are endpoints of edges in the optimal set S^*. Moreover, from the definition of

$e_i \in E(H)$, we know that w_1 and w_m are the first and last endpoints of S^* along the path $\delta_{G^*}(a_i)$. The path $\delta_{G^*}(a_i)$ is shown in grey in Figure 6.3. The endpoints of $\delta_{G^*}(a_i)$ are $a_i(0)$ and $a_i(1)$. Therefore, the subpath of $\delta_{G^*}(a_i)$ between $a_i(0)$ and w_1 only uses edges in G and no edges in $S^* = G^* \setminus G$. Therefore, the subpath only uses edges in $G \cap \delta_{G^*}(a_i)$. The subpath from $a_i(0)$ to w_1 is shown in black in Figure 6.3. Similarly, there is a path between w_m and $a_i(1)$ using only edges in $G \cap \delta_{G^*}(a_i)$.

Next, we consider the edge $e_j = w_\ell w_{\ell+1}$, where $1 \leq \ell \leq k-1$, $j \in I$ and $j < i$. Let the endpoints of a_j be $a_j(0)$ and $a_j(1)$. From the definition of $e_j = w_\ell w_{\ell+1}$, there is a subpath of $\delta_{G^*}(a_j)$ between w_ℓ and $a_j(0)$ that only uses edges in $G \cap \delta_{G^*}(a_j)$. Similarly, there is subpath of $\delta_{G^*}(a_j)$ between $a_j(1)$ and $w_{\ell+1}$ that only uses edges in $G \cap \delta_{G^*}(a_j)$. Therefore, there is a path between w_ℓ and $w_{\ell+1}$ that uses only edges in $\{G \cap \delta_{G^*}(a_j)\} \cup a_j$.

See Figure 6.4 for an example. Consider the edge $e_1 = w_1 w_2$. There is a path between w_1 and w_2 that only uses edges in $\{G \cap \delta_{G^*}(a_j)\}$, which are black edges, and the blue edge a_1. Similarly arguments apply for $e_2 = w_2 w_3$ and $e_3 = w_3 w_4$.

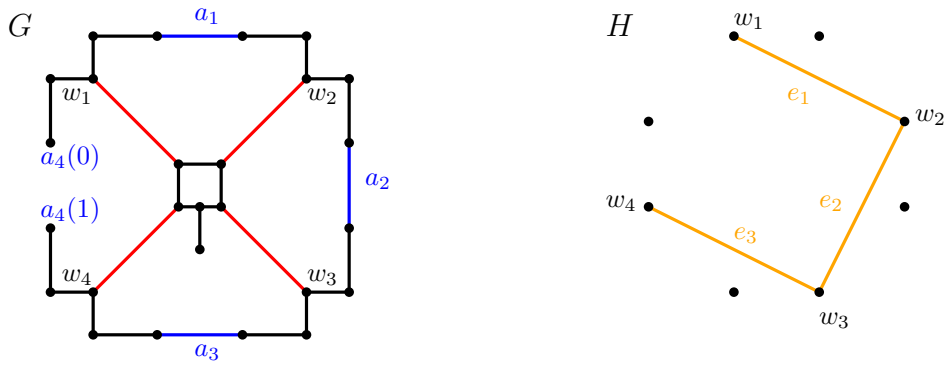


Figure 6.4: The path w_1, w_2, w_3, w_4 is shown on the right. There is a path between $a_4(0)$ and $a_4(1)$ only using the blue edges a_1, a_2, a_3 and black edges in $\delta_{G^*}(a_1), \delta_{G^*}(a_2), \delta_{G^*}(a_3)$ or $\delta_{G^*}(a_4)$.

The final step is to put it all together. There is a path between $a_i(0)$ and w_1 that only uses edges in $G \cap \delta_{G^*}(a_i)$. For $\ell = 1, \ldots, k-1$, there is a path between w_ℓ and $w_{\ell+1}$ that only uses edges in $\{G \cap \delta_{G^*}(a_j)\} \cup a_j$, where $j \in I \setminus \{i\}$. There is a path between w_m and $a_i(1)$ that only uses edges in $G \cap \delta_{G^*}(a_i)$. Therefore, there is a path between $a_i(0)$ and $a_i(1)$ that only uses edges in $\{G \cap \delta_{G^*}(a_j) : j \in I\} \cup \{a_j : j \in I \setminus \{i\}\}$, as required. □

In Lemma 13, we show a lower bound on the length of the path in Lemma 12.

Lemma 13. *The length of the path in Lemma 12 is at least $t \cdot d_M(a_i)$.*

Proof. From Definition 10, we have $d_{G_{i-1}}(a_i) > t \cdot d_M(a_i)$. Therefore, any path in G_{i-1} between the endpoints of a_i has length at least $t \cdot d_M(a_i)$. It suffices to show that the path is in G_{i-1}. By Lemma 12, all of the edges in the path are in $\{G \cap \delta_{G^*}(a_j) : j \in I\}$ or $\{a_j : j \in I \setminus \{i\}\}$. But $\{G \cap \delta_{G^*}(a_j) : j \in I\} \subseteq G \subseteq G_{i-1}$ and $\{a_j : j \in I \setminus \{i\}\} \subseteq G_{i-1}$. So the path is in G_{i-1} and its length is at least $t \cdot d_M(a_i)$. □

In Lemma 14, we upper bound the length of the path in Lemma 12.

Lemma 14. *The length of the path in Lemma 12 is at most $|I| \cdot t^* \cdot d_M(a_i)$.*

Proof. Given a set of edges E, let total(E) denote the total sum of edge lengths in E. Recall that the path in Lemma 12 only uses edges in $\{G \cap \delta_{G^*}(a_j) : j \in I\} \cup \{a_j : j \in I \setminus \{i\}\}$. A naïve approach to prove the lemma is to bound total($\{\delta_{G^*}(a_j) : j \in I\} \cup \{a_j : j \in I \setminus \{i\}\}$). Note that G is removed from the first set of braces. We have

$$\begin{aligned} \text{total}(\{\delta_{G^*}(a_j) : j \in I\}) &\leq \textstyle\sum_{j \in I} t^* \cdot d_M(a_j) \leq |I| \cdot t^* \cdot d_M(a_i), \\ \text{total}(\{a_j : j \in I \setminus \{i\}\}) &= \textstyle\sum_{j \in I \setminus \{i\}} d_M(a_j) \leq (|I| - 1) \cdot d_M(a_i). \end{aligned}$$

Therefore, the total length of the path is at most $(|I| \cdot t^* + |I| - 1) \cdot d_M(a_i) < |I| \cdot (t^* + 1) \cdot d_M(a_i)$. Since $(t^* + 1) \leq 2t^*$, we have proven Lemma 14 up to a factor of 2. This analysis would already yield an $(f, 2g)$-bicriteira approximation. However, to shave off the factor of 2 and obtain a tight analysis, we need a more sophisticated argument.

We strengthen our upper bound by re-introducing G back into the first set of braces, in other words, by bounding total($\{G \cap \delta_{G^*}(a_j)\}$). Since $G = G^* \setminus S^*$, we write

$$\text{total}(\{G \cap \delta_{G^*}(a_j)\}) = \text{total}(\{\delta_{G^*}(a_j)\}) - \text{total}(\{S^* \cap \delta_{G^*}(a_j)\}).$$

We have two cases, depending on the size of total($\{S^* \cap \delta_{G^*}(a_j)\}$).

Case 1. total($\{S^* \cap \delta_{G^*}(a_j)\}$) $\geq (1 - \frac{1}{|I|}) \cdot d_M(a_j)$ for all $j \in I$. Then,

$$\begin{aligned} t \cdot d_M(a_i) &\leq \text{total}(\{G \cap \delta_{G^*}(a_j) : j \in I\}) + \text{total}(\{a_j : j \in I \setminus \{i\}\}) \\ &= \text{total}(\{\delta_{G^*}(a_j) : j \in I\}) - \text{total}(\{S^* \cap \delta_{G^*}(a_j) : j \in I\}) + \textstyle\sum_{j \in I \setminus \{i\}} d_M(a_j) \\ &\leq \textstyle\sum_{j \in I} t^* \cdot d_M(a_j) - \sum_{j \in I} (1 - \frac{1}{|I|}) \cdot d_M(a_j) + \sum_{j \in I \setminus \{i\}} d_M(a_j) \\ &= \textstyle\sum_{j \in I} t^* \cdot d_M(a_j) - (1 - \frac{1}{|I|}) \cdot d_M(a_i) + \sum_{j \in I \setminus \{i\}} \frac{1}{|I|} \cdot d_M(a_j) \\ &\leq |I| \cdot t^* \cdot d_M(a_i) - (1 - \tfrac{1}{|I|}) \cdot d_M(a_i) + (\tfrac{|I|-1}{|I|}) \cdot d_M(a_i) \\ &= |I| \cdot t^* \cdot d_M(a_i), \end{aligned}$$

where the first line uses Lemma 12 and 13, the second line uses $G = G^* \setminus S^*$, the third line uses the assumption from the case distinction, and the final three lines simplify the expression. Therefore, $t \leq L \leq |I| \cdot t^* = gt^*$, where L is the length of the path in Lemma 12.

Case 2. total($\{S^* \cap \delta_{G^*}(a_j)\}$) $< (1 - \frac{1}{|I|}) \cdot d_M(a_j)$ for some $j \in I$. Then for every $s \in \{S^* \cap \delta_{G^*}(a_j)\}$, it holds that $d_M(s) < d_M(a_i)$. Therefore, $d_{G_{j-1}}(s) \leq t \cdot d_M(s)$, since a_j is the shortest edge in G_{j-1} satisfying $d_{G_{j-1}}(a_j) > t \cdot d_M(a_j)$. Let the endpoints of a_j be $a_j(0)$ and $a_j(1)$. Let the edges of $\delta_{G^*}(a_j)) \cap S^*$ be $s_1, \ldots, s_m$, and let the endpoints of s_i be w_{2i-1} and w_{2i}. Assume without loss of generality that the endpoints $w_1, \ldots, w_{2m}$ are in sorted

order along the path $\delta_{G^*}(a_j)$. Then,

$$\begin{aligned}
d_{G_{j-1}}(a_j) &\leq d_{G_{j-1}}(a_j(0), w_1) + \sum_{i=1}^{m} d_{G_{j-1}}(w_{2i-1}, w_{2i}) + \sum_{i=1}^{m-1} d_{G_{j-1}}(w_{2i}, w_{2i+1}) \\
&\quad + d_{G_{j-1}}(w_{2m}, a_j(1)) \\
&= d_{G_{j-1}}(a_j(0), w_1) + \sum_{i=1}^{m} d_{G_{j-1}}(s_i) + \sum_{i=1}^{m-1} d_{G_{j-1}}(w_{2i}, w_{2i+1}) \\
&\quad + d_{G_{j-1}}(w_{2m}, a_j(1)) \\
&\leq d_{G_{j-1}}(a_j(0), w_1) + \sum_{i=1}^{m-1} d_{G_{j-1}}(w_{2i}, w_{2i+1}) + d_{G_{j-1}}(w_{2m}, a_j(1)) \\
&\quad + \sum_{i=1}^{m} t \cdot d_M(s_i) \\
&\leq d_G(a_j(0), w_1) + \sum_{i=1}^{m-1} d_G(w_{2i}, w_{2i+1}) + d_G(w_{2m}, a_j(1)) \\
&\quad + \sum_{i=1}^{m} t \cdot d_M(s_i) \\
&= d_{G^*}(a_j(0), w_1) + \sum_{i=1}^{m-1} d_{G^*}(w_{2i}, w_{2i+1}) + d_{G^*}(w_{2m}, a_j(1)) \\
&\quad + \sum_{i=1}^{m} t \cdot d_M(s_i) \\
&= d_{G^*}(a_j) + \sum_{i=1}^{m} t \cdot d_M(s_i),
\end{aligned}$$

where the first line uses the triangle inequality, the second line uses $s_i = w_{2i-1}w_{2i}$, the third line uses $d_{G_{j-1}}(s) \leq t \cdot d_M(s)$, the fourth line uses $G \subset G_{i-1}$, the fifth line uses that all the subpaths no longer use edges in S^*, and the sixth line uses that all edges are a subset of the edges in $\delta^*_G(a_j)$. Therefore,

$$\begin{aligned}
t \cdot d_M(a_j) < d_{G_{j-1}}(a_j) &\leq d_{G^*}(a_j) + \sum_{i=1}^{m} t \cdot d_M(s_i) \\
&= t^* \cdot d_M(a_j) + t \cdot \text{total}(\{S^* \cap \delta_{G^*}(a_j)\}) \\
&= t^* \cdot d_M(a_j) + t \cdot (1 - \tfrac{1}{|I|}) \cdot d_M(a_j).
\end{aligned}$$

Simplifying, we get $t < t^* + t - \frac{t}{|I|}$, which implies $t < |I| \cdot t^* = gt^*$. □

Lemma 15 summarises the previous three lemmas.

Lemma 15. *If a_j exists for all $j = 1, \ldots, fk+1$, then $t \leq gt^*$.*

Finally, we use Lemma 15 to prove Theorem 6. The idea is to combine the sparsity bound in the case where the greedy construction halts after adding at most fk edges, with the dilation bound $t \leq gt^*$ in the case where the greedy construction halts after adding at least $fk+1$ edges.

Theorem 6. *For all $r \geq 1$, there is an $(f, (1+\delta)g)$-bicriteria approximation for Problem 5 that runs in $O(n^3(\log n + \log \frac{1}{\delta}))$ time, where*

$$f = 2\sqrt[r]{2}\ k^{1/r} \quad \text{and} \quad g = 2r.$$

Proof. First, we describe the decision algorithm. Given any $t \in \mathbb{R}$, the decision algorithm is to construct the greedy t-spanner as described in Definition 10. If at most fk edges are added, then we continue searching over dilation values that are less than t. If at least $fk+1$ edges are added, then we continue searching over dilation values that are greater than t.

Second, we perform a binary search to obtain an $(f, (1+\delta)g)$-bicriteria approximation for Problem 5. Given a set of vertices, Gudmundsson and Wong [108] show how to (implicitly) binary search a set of $O(n^4)$ critical values, so that the dilation of any graph with those

vertices will be within a factor of $O(n)$ of one of the critical values. We refine the search to a multiplicative $(1+\delta)$-grid. As a result, we obtain a $t \in \mathbb{R}$ where a greedy t-spanner adds at least $fk+1$ edges, but a greedy $(1+\delta)t$-spanner adds at most fk edges. By Lemma 15, we have $t \leq gt^*$, so $(1+\delta)t \leq (1+\delta)gt^*$. The greedy $(1+\delta)t$-spanner adds at most fk edges to the graph and its dilation is at most $(1+\delta)gt^*$, so we have an $(f,(1+\delta)g)$-bicriteria-approximation.

Third, we analyse the running time. The running time of the decision algorithm is $O(n^3)$ [108]. We perform the binary search by first calling the decider $O(\log n)$ times on the critical values, and an additional $O(\log \frac{1}{\delta})$ times on the multiplicative $(1+\delta)$-grid. □

6.4 Greedy analysis is tight

In this section, we will prove Theorem 7. We restate the theorem for convenience.

Theorem 7. *For all $r \geq 1$, assuming the Erdős girth conjecture, there is a graph class for which the algorithm in Theorem 6 returns an (f,g)-bicriteria approximation, where*

$$f = \Omega(k^{1/r}) \quad \textit{and} \quad g = 2r+1.$$

Recall that the Erdős girth conjecture [79] states that, for all positive integers n and r, there exists a graph H with n vertices, $m = \Omega(n^{1+1/r})$ edges, and girth $2r+2$. Recall that the girth of a graph is the length of its shortest cycle. Note that Theorem 7 does not contradict Theorem 6, as the constant in $\Omega(k^{1/r})$ is less than $2\sqrt[r]{2}$. One would need to resolve the Erdős girth conjecture to determine the precise constant.

We summarise our approach. We construct a graph G so that its girth graph is H. Using the properties of the girth graph, we show that the greedy spanner gives an (f,g)-bicriteria approximation where $f = \Omega(k^{1/r})$ and $g = (1+\delta)(2r+1)$. Our result shows that constructing the girth graph is essentially the "correct" way to analyse the greedy spanner, up to constant factors, since lower bounds on the girth of H directly translates to lower bounds on the dilation factor of the greedy algorithm for Problem 5.

We divide our proof of Theorem 7 into six parts. First, we define the underlying metric space M. Second, we define the graph G. Third, we define our Problem 5 instance. Fourth, we upper bound t^* in our Problem 5 instance. Fifth, we show that if $t \leq gt^*$ in our Problem 5 instance, then the greedy t-spanner adds at least $fk+1$ edges to G. Sixth, we analyse the algorithm in Theorem 6.

Part 1. We define the underlying metric space M. Assume that the vertices of the girth graph H are $V(H) = \{w_1, \ldots, w_n\}$ and its edges are $E(H) = \{e_1, \ldots, e_m\}$, where $m = \Omega(n^{1+1/r})$. The vertices of M are $V(M) = \{u_{i,j} : 1 \leq i \leq n, 0 \leq j \leq m\}$. For an example of the vertices $u_{i,j}$ where $1 \leq i \leq 4$, and $0 \leq j \leq 5$, see Figure 6.5.

The metric is a graph metric, where distances are shortest path distances in the graph $M = (V(M), E(M))$. It remains to construct the edges $E(M)$. We divide the edges $E(M)$ into three subsets: M_1, M_2 and M_3. Choose $\varepsilon = 1/4rn$. Refer to Figure 6.5.

- (Black) Define $M_1 = \{u_{i,0}u_{i,j} : 1 \leq i \leq n,\ 1 \leq j \leq m\}$. Each edge in M_1 has length 1.
- (Red) Define $M_2 = \{u_{i,0}u_{i+1,0} : 1 \leq i \leq n-1\}$. Each edge in M_2 has length 2ε.
- (Blue) Define $M_3 = \{u_{a,j}u_{b,j} : e_j = (w_a, w_b),\ 1 \leq j \leq m\}$. Each edge in M_3 has length ε.

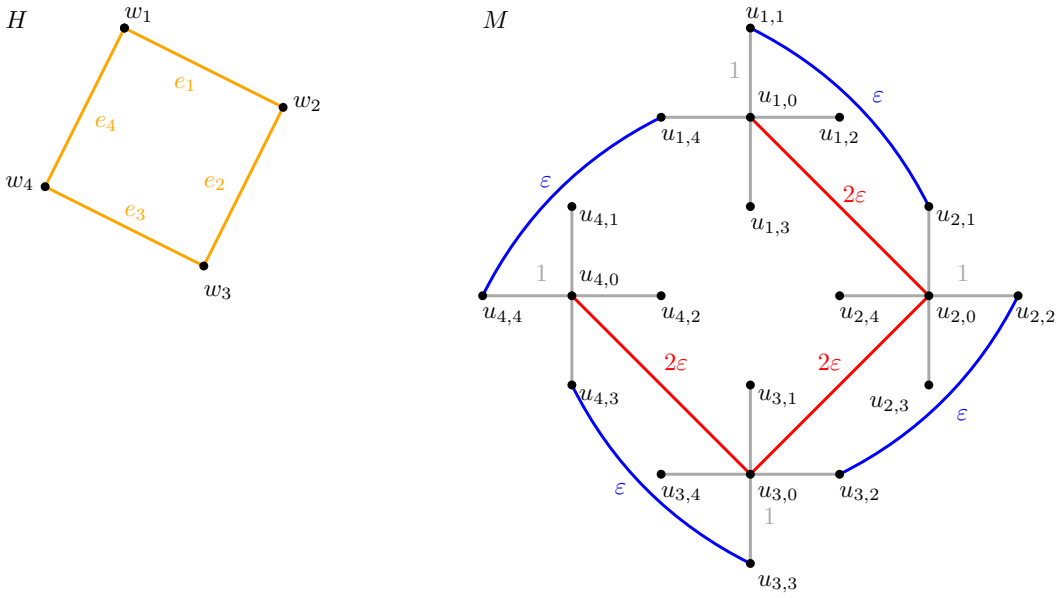


Figure 6.5: The girth graph H (orange), the vertices $u_{i,j} \in V(M)$ (black), the edges M_1 with length 1 (grey), the edges M_2 with length 2ε (red), and the edges M_3 with length ε (blue).

This completes the definition of the metric M.

Part 2. We define the graph G. Let $V(G) = V(M)$, and set $E(G) = M_1$. In Figure 6.5, the graph G only uses the black edges.

Part 3. We define our Problem 5 instance. Let M be the metric in part 1, G be the graph in part 2, $k = n - 1$ and $f = (m - 1)/(n - 1)$. We will prove that the dilation parameter is at least $g = 2r + 1$.

Part 4. We upper bound t^* in our Problem 5 instance. Define $S^* = M_2$ and define $G^* = G \cup S^*$. Since $|M_2| = n - 1$, the dilation of G^* is at least t^*, so $d_{G^*}(u_{i,0}, u_{j,0}) = 2\varepsilon \cdot |i - j|$. If $(i, a) \neq (j, b)$, then $d_{G^*}(u_{i,a}, u_{j,b}) = 2\varepsilon \cdot |i - j| + 2$. Thus, the diameter of G^* is at most $2\varepsilon n + 2$. The metric distance between any pair of points in G^* is at least ε. Therefore, the dilation of G' is at most $(2n\varepsilon + 2)/\varepsilon$, so $t^* \leq 2n + 2/\varepsilon$.

Part 5. We show that if $t \leq gt^*$ in our Problem 5 instance, then the greedy t-spanner adds at least $fk + 1$ edges to G. We prove by induction that the first $fk + 1$ edges added by the greedy t-spanner are all edges from M_3. The base case of $i = 0$ is trivially true. Assume the inductive hypothesis that the first i edges are from M_3, where $0 \leq i \leq fk$. Consider the graph G_i, and an edge $e \in M_3$ not currently in G_i. Then $d_M(e) = \varepsilon$. Let $e = u_{a,j}u_{b,j}$, where $a \neq b$. Next, we compute $d_{G_i}(e)$ by considering the shortest path between $u_{a,j}$ and $u_{b,j}$ in G_i. If there is no shortest path, then $d_{G_i}(e) = \infty$, and by Definition 10, the greedy t-spanner would add either e or another edge with length ε to G_i to obtain G_{i+1}. Otherwise, the shortest path must use edges in M_3, since the edges in M_1 alone cannot connect $u_{a,j}$ to $u_{b,j}$, since $a \neq b$. Let the edges of M_3 along the shortest path from $u_{a,j}$ to $u_{b,j}$, in order, be $\{(u_{a_1,j_1}, u_{b_1,j_1}), \ldots, (u_{a_d,j_d}, u_{b_d,j_d})\}$ for some $1 \leq d \leq m-1$. Note for all $1 \leq c \leq d$, each of the vertices $u_{a_c,j_c}, u_{b_c,j_c} \in V(G)$ are distinct, since no edges in M_3 share an endpoint. Consider the path from $u_{a,j}$ to u_{a_1,j_1}. All edges along this path are in M_1,

and there are at least two edges, since $j \neq 0$ and $j_1 \neq 0$. Therefore, $d_{G_i}(u_{a,j}, u_{a_1,j_1}) \geq 2$, and $a = a_1$. Applying the same argument between u_{b_d,j_d} and $u_{b,j}$, we get $d_{G_i}(u_{b_d,j_d}, u_{b,j}) \geq 2$. Applying the same argument between u_{b_c,j_c} and $u_{a_{c+1},j_{c+1}}$ for all $1 \leq c \leq d-1$, we get $d_{G_i}(u_{b_c,j_c}, u_{a_{c+1},j_{c+1}}) \geq 2$. Summing this all together, the length of the shortest path from $u_{a,j}$ to $u_{b,j}$ is at least $2(d+1) + 2d\varepsilon$. Next, we show $d \geq 2r+1$. Since there is an edge $u_{a,j}u_{b,j}$ in M_3, there is an edge between w_a and w_b in H. Since there is an edge $u_{a_c,j_c}u_{b_c,j_c}$ in M_3, there is an edge between w_{a_c} and w_{b_c} in H. But there is a path from $u_{a,j}$ to u_{a_1,j_1} using edges only in M_1, so $a = a_1$. Similarly, $b_1 = a_2, \ldots, b_{d-1} = a_d$, $b_d = b$. So there is a cycle $w_{a_1}, w_{a_2}, \ldots, w_{a_d}, w_{b_d}$ of length $d+1$ in the graph H. But the girth of H is $2r+2$, so $d \geq 2r+1$, as claimed. The length of the shortest path from $u_{a,j}$ to $u_{b,j}$ is at least $2(d+1) + 2d\varepsilon$, which is at least $2(2r+2) + 2(2r+1)\varepsilon$. But now,

$$\begin{aligned} d_{G_i}(u_{a,j}, u_{b,j}) &\geq 2(2r+2) + 2(2r+1)\varepsilon \\ &\geq (2r+1) \cdot \varepsilon \cdot (2/\varepsilon + \tfrac{1}{2r+1} \cdot (2/\varepsilon) + 2) \\ &> g \cdot d_M(u_{a,j}, u_{b,j}) \cdot (2/\varepsilon + 2n) \\ &= gt^* \cdot d_M(u_{a,j}, u_{b,j}) \\ &\geq t \cdot d_M(u_{a,j}, u_{b,j}). \end{aligned}$$

Therefore, $d_{G_i}(u_{a,j}, u_{b,j}) > t \cdot d_M(u_{a,j}, u_{b,j})$, so by Definition 10, the greedy t-spanner would add either $u_{a,j}u_{b,j}$ or another edge in M_3 to G_i to obtain G_{i+1}. This completes the induction, so the first $fk+1$ edges added by the greedy t-spanner are all edges from M_3. As a consequence, $t \leq gt^*$ implies that the greedy t-spanner adds at least $fk+1$ edges to G, completing the proof.

Part 6. We analyse the algorithm in Theorem 6. If at most fk edges are added, then we continue searching for dilation values less than t, whereas if at least $fk+1$ edges are added, then we continue searching for dilation values greater than t. Therefore, for all $t \leq gt^*$, the algorithm in Theorem 6 would continue searching for values greater than t. The dilation value returned by the algorithm is at least gt^*. So Theorem 6 returns an $(f, (1+\delta)g)$-bicriteria approximation, where $f = m - 1/n - 1 = \Omega(k^{1/r})$, and $g = 2r+1$, as required.

Finally, putting all six parts together, we obtain Theorem 7.

6.5 Set cover reduction

In this section, we will prove Theorem 8. We restate the theorem for convenience.

Theorem 8. *For all $\varepsilon > 0$, it is NP-hard to obtain an (f, g)-bicriteria approximation for Problem 5, where*

$$f = \text{poly}(k) \quad \textit{and} \quad g = 2 - \varepsilon.$$

We reduce from set cover. Our set cover instance consists of m elements $E = \{e_1, \ldots, e_m\}$ and L sets $S = \{S_1, \ldots, S_L\}$, where $S_\ell \subseteq E$ for all $1 \leq \ell \leq L$. Dinur and Steurer [66] state that, for every constant $\alpha > 0$, there exists a constant k for which it is NP-hard to decide whether (i) there exists a cover consisting of k sets, or (ii) any cover consists of more than $k(1-\alpha)\log n$ sets. Here, n denotes the complexity of the set cover input. Note that $n = O(mL)$ since $\sum_{\ell=1}^{L} |S_\ell| \leq mL$.

We summarise our approach. We show that every set cover instance can be reduced to a Problem 5 instance. If there is an (f, g)-bicriteria approximation for Problem 5 that can be

computed in polynomial time, where $f = (1-\alpha)\log n$ and $g = 2-\varepsilon$, then one would be able to approximate set cover to within a factor of $(1-\alpha)\log n$ in polynomial time, contradicting the result of [66].

We divide our proof of Theorem 8 into six parts. First, we define the underlying metric space M. Second, we define the graph G. Third, we define our Problem 5 instance. Fourth, we upper bound the dilation, assuming the set cover instance is a YES-instance. Fifth, we lower bound the dilation, assuming the set cover instance is a NO-instance. Sixth, we show that it is NP-hard to obtain an (f, g)-bicriteria approximation.

Part 1. We define the underlying metric space M. For each element e_i where $1 \leq i \leq m$, construct $2k+2$ vertices $U_i = \{u_{i,1}, u'_{i,1}, \ldots, u_{i,k+1}, u'_{i,k+1}\}$. For each set S_ℓ where $1 \leq \ell \leq L$, construct three vertices $V_\ell = \{v_\ell, v'_\ell, w_\ell\}$. For an example of the vertices $u_{i,j}, u'_{i,j}, v_\ell, v'_\ell, w_\ell$ where $1 \leq i \leq 3$, $1 \leq j \leq 2$, and $1 \leq \ell \leq 2$, see Figure 6.6. Define the vertices $V(M) = U_1 \cup \ldots U_m \cup V_1 \cup \ldots V_L$.

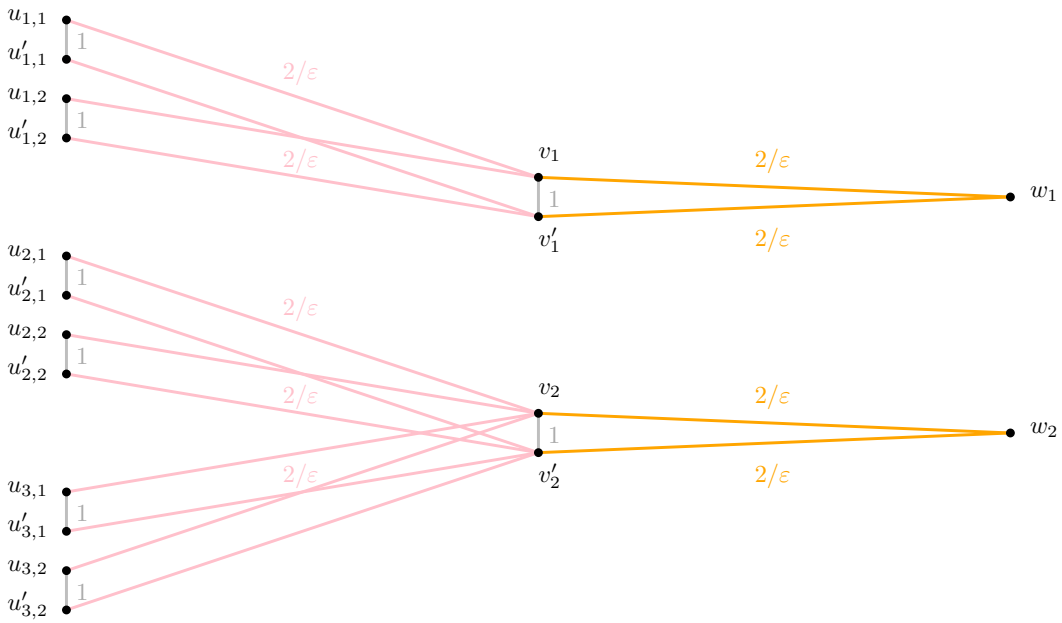


Figure 6.6: The graph metric M for a set cover instance where $k = 1$, $S_1 = \{e_1\}$ and $S_2 = \{e_2, e_3\}$. The edges $M_1 \cup M_2$ are shown in grey, and have length 1. The edges $M_3 \cup M_4$ are shown in pink, and have length $2/\varepsilon$. The edges $M_5 \cup M_6$ are shown in orange, and have length $2/\varepsilon$.

The metric distances between any pair of vertices in $V(M)$ is the shortest path between them in the weighted graph $M = (V(M), E(M))$. We divide the edges $E(M)$ into six subsets: $M_1, \ldots, M_6$. Refer to Figure 6.6.

- (Grey) Define $M_1 = \{u_{i,j}u'_{i,j} : 1 \leq i \leq m,\ 1 \leq j \leq k+1\}$. Each edge in M_1 has length 1.
- (Grey) Define $M_2 = \{v_\ell v'_\ell : 1 \leq \ell \leq L\}$. Each edge in M_2 has length 1.
- (Pink) Define $M_3 = \{u_{i,j}v_\ell : e_i \in S_\ell,\ 1 \leq j \leq k+1\}$. Each edge in M_3 has length $2/\varepsilon$.

- (Pink) Define $M_4 = \{u'_{i,j}v'_\ell : e_i \in S_\ell,\ 1 \le j \le k+1\}$. Each edge in M_4 has length $2/\varepsilon$.
- (Orange) Define $M_5 = \{v_\ell w_\ell : 1 \le \ell \le L\}$. Each edge in M_5 has length $2/\varepsilon$.
- (Orange) Define $M_6 = \{v'_\ell w_\ell : 1 \le \ell \le L\}$. Each edge in M_6 has length $2/\varepsilon$.

This completes the definition of the metric M.

Part 2. We define the graph G. Define the vertices of G to be $V(G) = V(M)$. Define the edges of G to be $E(G) = M_3 \cup M_4 \cup M_5 \cup M_6$. This completes the definition of G. In Figure 6.6, the graph G uses only the pink and orange edges.

Part 3. We define our Problem 5 instance. The metric space M, graph G, and parameter k are defined as above. We will show that if the set cover instance is a YES-instance, then there are k edges that one can add to the graph so the resulting dilation is $t^* \le 4/\varepsilon+1$. We will show that if the set cover instance is a NO-instance, then there are no fk edges that one can add to the graph so that the resulting dilation is at most gt^*, where $f = (1-\alpha)\log n$ and $g = 2-\varepsilon$. This completes the definition of the Problem 5 instance.

Part 4. We show $t^* \le 4/\varepsilon+1$, assuming the set cover instance is a YES-instance. In particular, there exists k sets $\{S_{\ell_1}, \ldots, S_{\ell_k}\}$ in S that covers E. Define $S^* = \{v_{\ell_1}v'_{\ell_1}, \ldots, v_{\ell_k}v'_{\ell_k}\}$, and define $G^* = G \cup S^*$. For all graph metrics M, the maximum dilation is obtained between a pair of points where the shortest path in M between them only uses a single edge in $E(M)$. This is because, if the shortest path has multiple edges in $E(M)$, then the dilation between the endpoints of one of those edges would be at least as large. Therefore, it suffices to consider the dilation between the endpoints of the edges in $M_1, \ldots, M_6$. However, the dilation between the endpoints of edges in $M_3, \ldots, M_6$ is 1. Therefore, it suffices to consider endpoints of edges in M_1 and M_2. For M_2,

$$d_{G^*}(v_\ell, v'_\ell) \le d_{G^*}(v_\ell, w_\ell) + d_{G^*}(w_\ell, v'_\ell) \le 4/\varepsilon.$$

For M_1, recall that $\{S_{\ell_1}, \ldots, S_{\ell_k}\}$ covers E. So, for each $u_{i,j}u'_{i,j} \in M_1$ there exists an ℓ satisfying $u_{i,j}v_\ell \in M_3$, $v_\ell v'_\ell \in S^*$, and $v'_\ell u'_{i,j} \in M_4$. Therefore,

$$d_{G^*}(u_{i,j}, u'_{i,j}) \le d_{G^*}(u_{i,j}, v_\ell) + d_{G^*}(v_\ell, v'_\ell) + d_{G^*}(v'_\ell, u'_{i,j}) \le 4/\varepsilon + 1.$$

Hence, the dilation of G^* is at most $4/\varepsilon + 1$, completing the proof that $t^* \le 4/\varepsilon + 1$.

Part 5. We show that there are no fk edges that one can add to G to obtain a dilation of at most gt^*, assuming the set cover instance is a NO-instance. Recall that $f = (1-\alpha)\log n$ and $g = 2-\varepsilon$. Suppose for the sake of contradiction that there exists fk edges so that the final dilation is at most $gt^* = (2-\varepsilon)(4/\varepsilon+1) < 8/\varepsilon$. Let the set of fk edges be F. Suppose that one of the edges in F is from the set M_1. Let the edge be $u_{i,j}u'_{i,j} \in F \cap M_1$. For all $e_i \in E$ there exists an S_ℓ so that $e_i \in S_\ell$. We exchange the edge $u_{i,j}u'_{i,j}$ with $v_\ell v'_\ell$. We show that this exchange does not affect the property that the final dilation is at most $gt^* < 8/\varepsilon$. Similarly to in part 4, it suffices to consider pairs of points that are endpoints of $M_1 \cup M_2$. For M_2, any path between v_p and v'_p that uses the edge $u_{i,j}u'_{i,j}$ must pass through v_ℓ and v'_ℓ, so exchanging the edge $u_{i,j}u'_{i,j}$ with $v_\ell v'_\ell$ decreases the shortest path distance between all $v_p v'_p \in M_2$. For M_1, any path from $u_{p,q}$ to $u'_{p,q}$ that uses the edge $u_{i,j}u'_{i,j}$ has length at least $8/\varepsilon + 1$, so it cannot be the shortest path between $u_{p,q}u'_{p,q} \in M_1$. Therefore, after performing exchanges for all $u_{i,j}u'_{i,j} \in F \cap M_1$, we can assume that $F \subseteq M_2$. Since $|F| = k(1-\alpha)\log n$ and the set cover instance is a NO-instance, the set F cannot be a set cover for the elements $e_i \in E$. Therefore, there exists an i so that $v_\ell v'_\ell \notin F$ for all S_ℓ where $e_i \in S_\ell$. For all $v_\ell v'_\ell \notin F$, we have $d_{G\cup F}(v_\ell, v'_\ell) \ge d_{G\cup F}(v_\ell, w_\ell) + d_{G\cup F}(w_\ell, v'_\ell) = 4/\varepsilon$. Therefore,

$$d_{G\cup F}(u_{i,j}, u'_{i,j}) \ge \max_{\ell : e_i \in S_\ell} \left(d_{G\cup F}(u_{i,j}, v_\ell) + d_{G\cup F}(v_\ell, v'_\ell) + d_{G\cup F}(v'_\ell, u'_{i,j})\right) \ge 8/\varepsilon,$$

contradicting the fact that the final dilation is at most gt^*. This completes the proof that there are no fk edges that one can add to G to obtain a dilation of at most gt^*, assuming the set cover instance is a NO-instance.

Part 6. We show that it is NP-hard to obtain an (f, g)-bicriteria approximation. If there is an (f, g)-bicriteria approximation for Problem 5, one would be able to decide whether (i) there exists a set of k edges to add to G to obtain a dilation of t^*, or (ii) there are no fk edges that one can add to G to obtain a dilation of at most gt^*. Therefore, one would be able to decide whether the set cover instance is a YES-instance or a NO-instance. Since set cover is NP-hard, it follows that an (f, g)-bicriteria approximation is NP-hard, where $f = (1 - \alpha) \log n$ and $g = 2 - \varepsilon$. In our reduction, the size of the graph is proportional to the size of the set cover instance, so $|G| = \Omega(n)$.

Finally, putting all six parts together, we obtain Lemma 16.

Lemma 16. *For all $\alpha > 0$, $\varepsilon > 0$, it is NP-hard to obtain an (f, g)-bicriteria approximation for Problem 5, where*

$$f = (1 - \alpha) \log |G| \quad \textit{and} \quad g = 2 - \varepsilon.$$

In the set cover instance of [66], k is a constant, so $h(k) = o(\log n)$ for all functions h.

Corollary 17. *For all functions h, it is NP-hard to obtain an (f, g)-bicriteria approximation for Problem 5, where*

$$f = h(k) \quad \textit{and} \quad g = 2 - \varepsilon.$$

Setting $h(k) = \text{poly}(k)$ in Corollary 17 gives us Theorem 8.

6.6 Conclusion

We provide bicriteria approximation algorithms for the problem of adding k edges to a graph to minimise its dilation. Our main result is a $(2\sqrt[r]{2}\ k^{1/r}, 2r)$-bicriteria approximation for all $r \geq 1$, that runs in $O(n^3 \log n)$ time. Our analysis is tight and it is NP-hard to obtain a $(\text{poly}(k), 2 - \varepsilon)$-bicriteria approximation for any $\varepsilon > 0$. We provide a simple $(2k^2 \log n, 1)$-bicriteria approximation.

We conclude with directions for future work. Problem 1 remains open. In particular, Obstacle 2 asks: is there an $\varepsilon > 0$ for which there is an $O(n^{1-\varepsilon})$-approximation algorithm for the minimum dilation spanning tree problem? The linear approximation factor of Problem 3 cannot be improved unless Obstacle 4 is resolved. Another way to circumvent Obstacle 4 is to consider Problem 3 in the special case of an unweighted graph metric. Finally, Problem 5 offers several directions for future work. Can one obtain a trade-off between sparsity and dilation that is better than the greedy t-spanner construction? What is the sparsity-dilation trade-off when $1 < f < 2$? Can the lightness of the greedy t-spanner be bounded in Problem 5? Can the approximation factor or the running time of Theorem 9 be improved?

Bibliography

[1] Mohammad Ali Abam, Mark de Berg, Mohammad Farshi, and Joachim Gudmundsson. Region-fault tolerant geometric spanners. *Discret. Comput. Geom.*, 41(4):556–582, 2009.

[2] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015*, pages 59–78. IEEE Computer Society, 2015.

[3] Peyman Afshani, Lars Arge, and Kasper Dalgaard Larsen. Orthogonal range reporting: query lower bounds, optimal structures in 3-d, and higher-dimensional improvements. In David G. Kirkpatrick and Joseph S. B. Mitchell, editors, *Proceedings of the 26th Symposium on Computational Geometry, SoCG 2010*, pages 240–246. ACM, 2010.

[4] Peyman Afshani and Anne Driemel. On the complexity of range searching among curves. In *Proceedings of the 29th Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018*, pages 898–917. SIAM, 2018.

[5] Pankaj K. Agarwal, Kyle Fox, Kamesh Munagala, Abhinandan Nath, Jiangwei Pan, and Erin Taylor. Subtrajectory clustering: Models and algorithms. In *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, 2018*, pages 75–87. ACM, 2018.

[6] Pankaj K. Agarwal, Kyle Fox, Jiangwei Pan, and Rex Ying. Approximating dynamic time warping and edit distance for a pair of point sequences. In Sándor P. Fekete and Anna Lubiw, editors, *32nd International Symposium on Computational Geometry, SoCG 2016*, volume 51 of *LIPIcs*, pages 6:1–6:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

[7] Abu Reyan Ahmed, Greg Bodwin, Faryad Darabi Sahneh, Keaton Hamm, Mohammad Javad Latifi Jebelli, Stephen G. Kobourov, and Richard Spence. Graph spanners: A tutorial review. *Computer Science Review*, 37:100253, 2020.

[8] Hee-Kap Ahn, Mohammad Farshi, Christian Knauer, Michiel H. M. Smid, and Yajun Wang. Dilation-optimal edge deletion in polygonal cycles. *International Journal of Computational Geometry & Applications*, 20(1):69–87, 2010.

[9] Hugo A. Akitaya, Frederik Brüning, Erin W. Chambers, and Anne Driemel. Covering a curve with subtrajectories. *CoRR*, abs/2103.06040, 2021.

[10] Mohamed H. Ali, John Krumm, Travis Rautman, and Ankur Teredesai. ACM SIGSPATIAL GIS cup 2012. In *Proceedings of the 20th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL 2012*, pages 597–600. ACM, 2012.

[11] Helmut Alt, Alon Efrat, Günter Rote, and Carola Wenk. Matching planar maps. *J. Algorithms*, 49(2):262–283, 2003.

[12] Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *Int. J. Comput. Geom. Appl.*, 5:75–91, 1995.

[13] Ingo Althöfer, Gautam Das, David P. Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9:81–100, 1993.

[14] Boris Aronov, Kevin Buchin, Maike Buchin, Bart M. P. Jansen, Tom de Jong, Marc J. van Kreveld, Maarten Löffler, Jun Luo, Rodrigo I. Silveira, and Bettina Speckmann. Connect the dot: Computing feed-links for network extension. *Journal of Spatial Information Science*, 3(1):3–31, 2011.

[15] Boris Aronov, Mark de Berg, Otfried Cheong, Joachim Gudmundsson, Herman J. Haverkort, Michiel H. M. Smid, and Antoine Vigneron. Sparse geometric graphs with small dilation. *Computational Geometry*, 40(3):207–219, 2008.

[16] Gowtham Atluri, Anuj Karpatne, and Vipin Kumar. Spatio-temporal data mining: A survey of problems and methods. *ACM Comput. Surv.*, 51(4):83:1–83:41, 2018.

[17] Mihai Badoiu, Piotr Indyk, and Anastasios Sidiropoulos. Approximation algorithms for embedding general metrics into trees. In *Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA 2007.

[18] Julian Baldus and Karl Bringmann. A fast implementation of near neighbors queries for Fréchet distance (GIS cup). In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL 2017*, pages 99:1–99:4. ACM, 2017.

[19] Surender Baswana and Sandeep Sen. A simple and linear time randomized algorithm for computing sparse spanners in weighted graphs. *Random Structures & Algorithms*, 30(4):532–563, 2007.

[20] Selcan Kaplan Berkaya, Alper Kursat Uysal, Efnan Sora Gunal, Semih Ergin, Serkan Gunal, and M Bilginer Gulmezoglu. A survey on ECG analysis. *Biomedical Signal Processing and Control*, 43:216–235, 2018.

[21] Donald J. Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In Usama M. Fayyad and Ramasamy Uthurusamy, editors, *Knowledge Discovery in Databases: Papers from the 1994 AAAI Workshop, Seattle, Washington, USA, July 1994. Technical Report WS-94-03*, pages 359–370. AAAI Press, 1994.

[22] Krishnan Bhaskaran, Antonio Gasparrini, Shakoor Hajat, Liam Smeeth, and Ben Armstrong. Time series regression studies in environmental epidemiology. *International Journal of Epidemiology*, 42(4):1187–1195, 2013.

[23] Davide Bilò. Almost optimal algorithms for diameter-optimally augmenting trees. *Theoretical Computer Science*, 931:31–48, 2022.

[24] Béla Bollobás. *Extremal graph theory*. Courier Corporation, 2004.

[25] Sotiris Brakatsoulas, Dieter Pfoser, Randall Salas, and Carola Wenk. On map-matching vehicle tracking data. In *Proceedings of the 31st International Conference on Very Large Data Bases, VLDB 2005*, pages 853–864. ACM, 2005.

[26] Aléx F. Brandt, Miguel F. A. de Gaiowski, Pedro J. de Rezende, and Cid C. de Souza. Computing minimum dilation spanning trees in geometric graphs. In *Proceedings of the 21st Annual International Computing and Combinatorics Conference*, COCOON 2015.

[27] Milutin Brankovic, Kevin Buchin, Koen Klaren, André Nusser, Aleksandr Popov, and Sampson Wong. (k, ℓ)-medians clustering of trajectories using continuous dynamic time warping. In *SIGSPATIAL '20: 28th International Conference on Advances in Geographic Information Systems*, pages 99–110. ACM, 2020.

[28] Karl Bringmann. Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014*, pages 661–670. IEEE Computer Society, 2014.

[29] Karl Bringmann, Anne Driemel, André Nusser, and Ioannis Psarros. Tight bounds for approximate near neighbor searching for time series under the Fréchet distance. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022*, pages 517–550. SIAM, 2022.

[30] Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015*, pages 79–97. IEEE Computer Society, 2015.

[31] Karl Bringmann and Marvin Künnemann. Improved approximation for Fréchet distance on c-packed curves matching conditional lower bounds. *Int. J. Comput. Geom. Appl.*, 27(1-2):85–120, 2017.

[32] Karl Bringmann, Marvin Künnemann, and André Nusser. Fréchet distance under translation: Conditional hardness and an algorithm via offline dynamic grid reachability. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 2902–2921. SIAM, 2019.

[33] Karl Bringmann and Wolfgang Mulzer. Approximability of the discrete Fréchet distance. *J. Comput. Geom.*, 7(2):46–76, 2016.

[34] Kevin Buchin, Maike Buchin, David Duran, Brittany Terese Fasy, Roel Jacobs, Vera Sacristán, Rodrigo I. Silveira, Frank Staals, and Carola Wenk. Clustering trajectories for map construction. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, GIS 2017*, pages 14:1–14:10. ACM, 2017.

[35] Kevin Buchin, Maike Buchin, Joachim Gudmundsson, Jorren Hendriks, Erfan Hosseini Sereshgi, Vera Sacristán, Rodrigo I. Silveira, Jorrick Sleijster, Frank Staals, and Carola Wenk. Improved map construction using subtrajectory clustering. In *LocalRec'20: Proceedings of the 4th ACM SIGSPATIAL Workshop on Location-Based Recommendations, Geosocial Networks, and Geoadvertising, LocalRec@SIGSPATIAL 2020*, pages 5:1–5:4. ACM, 2020.

[36] Kevin Buchin, Maike Buchin, Joachim Gudmundsson, Maarten Löffler, and Jun Luo. Detecting commuting patterns by clustering subtrajectories. *Int. J. Comput. Geom. Appl.*, 21(3):253–282, 2011.

[37] Kevin Buchin, Maike Buchin, Wouter Meulemans, and Wolfgang Mulzer. Four Soviets walk the dog: Improved bounds for computing the Fréchet distance. *Discret. Comput. Geom.*, 58(1):180–216, 2017.

[38] Kevin Buchin, Maike Buchin, and Yusu Wang. Exact algorithms for partial curve matching via the fréchet distance. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009*, pages 645–654. SIAM, 2009.

[39] Kevin Buchin, Yago Diez, Tom van Diggelen, and Wouter Meulemans. Efficient trajectory queries under the Fréchet distance (GIS cup). In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL 2017*, pages 101:1–101:4. ACM, 2017.

[40] Kevin Buchin, Anne Driemel, Joachim Gudmundsson, Michael Horton, Irina Kostitsyna, Maarten Löffler, and Martijn Struijs. Approximating (k, ℓ)-center clustering for curves. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 2922–2938. SIAM, 2019.

[41] Kevin Buchin, Chenglin Fan, Maarten Löffler, Aleksandr Popov, Benjamin Raichel, and Marcel Roeloffzen. Fréchet distance for uncertain curves. In Artur Czumaj, Anuj Dawar, and Emanuela Merelli, editors, *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020, July 8-11, 2020, Saarbrücken, Germany (Virtual Conference)*, volume 168 of *LIPIcs*, pages 20:1–20:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[42] Kevin Buchin, Tim Ophelders, and Bettina Speckmann. SETH says: Weak Fréchet distance is faster, but only if it is continuous and in one dimension. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019*, pages 2887–2901. SIAM, 2019.

[43] Maike Buchin. *On the computability of the Fréchet distance between triangulated surfaces*. PhD thesis, Freie Universität Berlin, 2007.

[44] Maike Buchin, Anne Driemel, and Dennis Rohde. Approximating (k, ℓ)-median clustering for polygonal curves. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*, pages 2697–2717. SIAM, 2021.

[45] Maike Buchin and Bernhard Kilgus. Fréchet distance between two point sets. In J. Mark Keil and Debajyoti Mondal, editors, *Proceedings of the 32nd Canadian Conference on Computational Geometry, CCCG 2020, August 5-7, 2020, University of Saskatchewan, Saskatoon, Saskatchewan, Canada*, pages 249–257, 2020.

[46] Maike Buchin, Ivor van der Hoog, Tim Ophelders, Lena Schlipf, Rodrigo I. Silveira, and Frank Staals. Efficient Fréchet distance queries for segments. *CoRR*, abs/2203.01794, 2022.

[47] Leizhen Cai and Derek G. Corneil. Tree spanners. *SIAM Journal on Discrete Mathematics*, 8(3):359–387, 1995.

[48] Jean-Lou De Carufel, Carsten Grimm, Anil Maheshwari, Megan Owen, and Michiel H. M. Smid. A note on the unsolvability of the weighted region shortest path problem. *Comput. Geom.*, 47(7):724–727, 2014.

[49] Erin W. Chambers, Brittany Terese Fasy, Yusu Wang, and Carola Wenk. Map-matching using shortest paths. *ACM Trans. Spatial Algorithms Syst.*, 6(1):6:1–6:17, 2020.

[50] Hubert T.-H. Chan, Anupam Gupta, Bruce M. Maggs, and Shuheng Zhou. On hierarchical routing in doubling metrics. In *Proceedings of the 16th Annual Symposium on Discrete Algorithms, SODA*, pages 762–771. SIAM, 2005.

[51] Cheng Chang and Baoyao Zhou. Multi-granularity visualization of trajectory clusters using sub-trajectory clustering. In *ICDM Workshops 2009, IEEE International Conference on Data Mining Workshops, 2009*, pages 577–582. IEEE Computer Society, 2009.

[52] Pingfu Chao, Yehong Xu, Wen Hua, and Xiaofang Zhou. A survey on map-matching algorithms. In *Proceedings of the 31st Australasian Database Conference, ADC 2020*, volume 12008 of *Lecture Notes in Computer Science*, pages 121–133. Springer, 2020.

[53] Daniel Chen, Anne Driemel, Leonidas J. Guibas, Andy Nguyen, and Carola Wenk. Approximate map matching with respect to the Fréchet distance. In *Proceedings of the 13th Workshop on Algorithm Engineering and Experiments, ALENEX 2011*, pages 75–83. SIAM, 2011.

[54] Daniel Chen, Leonidas J Guibas, Qixing Huang, and Jian Sun. A faster algorithm for matching planar maps under the weak Fréchet distance. *Unpublished, December*, 2008.

[55] Daniel Chen, Christian Sommer, and Daniel Wolleb. Fast map matching with vertex-monotone Fréchet distance. In *Proceedings of the 21st Symposium on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems, ATMOS 2021*, volume 96 of *OASIcs*, pages 10:1–10:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[56] Otfried Cheong, Herman J. Haverkort, and Mira Lee. Computing a minimum-dilation spanning tree is NP-hard. *Computational Geometry*, 41(3):188–205, 2008.

[57] Vasek Chvátal. A greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, 4(3):233–235, 1979.

[58] Ian R Cleasby, Ewan D Wakefield, Barbara J Morrissey, Thomas W Bodey, Steven C Votier, Stuart Bearhop, and Keith C Hamer. Using time-series similarity measures to compare animal movement trajectories in ecology. *Behavioral Ecology and Sociobiology*, 73(11):1–19, 2019.

[59] Richard Cole. Slowing down sorting networks to obtain faster sorting algorithms. *J. ACM*, 34(1):200–208, 1987.

[60] Gautam Das, Paul J. Heffernan, and Giri Narasimhan. Optimally sparse spanners in 3-dimensional euclidean space. In *Proceedings of the 9th Annual Symposium on Computational Geometry*, SoCG 1993.

[61] Mark de Berg, Otfried Cheong, Marc J. van Kreveld, and Mark H. Overmars. *Computational Geometry: Algorithms and Applications, 3rd Edition*. Springer, 2008.

[62] Mark de Berg, Joachim Gudmundsson, and Ali D. Mehrabi. A dynamic data structure for approximate proximity queries in trajectory data. In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL 2017*, pages 48:1–48:4. ACM, 2017.

[63] Mark de Berg, Ali D. Mehrabi, and Tim Ophelders. Data structures for Fréchet queries in trajectory data. In *Proceedings of the 29th Canadian Conference on Computational Geometry, CCCG 2017*, pages 214–219, 2017.

[64] Erik D. Demaine and Morteza Zadimoghaddam. Minimizing the diameter of a network using shortcut edges. In *Proceedings of the 12th Scandinavian Workshop on Algorithm Theory*, SWAT 2010.

[65] Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[66] Irit Dinur and David Steurer. Analytical approach to parallel repetition. In *Proceedings of the 46th ACM Symposium on Theory of Computing*, STOC 2014.

[67] Anne Driemel and Sariel Har-Peled. Jaywalking your dog: Computing the Fréchet distance with shortcuts. *SIAM J. Comput.*, 42(5):1830–1866, 2013.

[68] Anne Driemel, Sariel Har-Peled, and Carola Wenk. Approximating the Fréchet distance for realistic curves in near linear time. *Discret. Comput. Geom.*, 48(1):94–127, 2012.

[69] Anne Driemel, Amer Krivosija, and Christian Sohler. Clustering time series under the Fréchet distance. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016*, pages 766–785. SIAM, 2016.

[70] Anne Driemel and Ioannis Psarros. ANN for time series under the Fréchet distance. In *Proceedings of the 17th International Symposium on Algorithms and Data Structures, WADS 2021*, volume 12808 of *Lecture Notes in Computer Science*, pages 315–328. Springer, 2021.

[71] Anne Driemel, Ioannis Psarros, and Melanie Schmidt. Sublinear data structures for short Fréchet queries. *CoRR*, abs/1907.04420, 2019.

[72] Richard M Dudley. Metric entropy of some classes of sets with differentiable boundaries. *Journal of Approximation Theory*, 10(3):227–236, 1974.

[73] Christian A. Duncan, Michael T. Goodrich, and Stephen G. Kobourov. Balanced aspect ratio trees: Combining the advantages of k-d trees and octrees. *J. Algorithms*, 38(1):303–333, 2001.

[74] Fabian Dütsch and Jan Vahrenhold. A filter-and-refinement-algorithm for range queries based on the Fréchet distance (GIS cup). In *Proceedings of the 25th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL 2017*, pages 100:1–100:4. ACM, 2017.

[75] Alon Efrat, Quanfu Fan, and Suresh Venkatasubramanian. Curve matching, time warping, and light fields: New algorithms for computing similarity between curves. *J. Math. Imaging Vis.*, 27(3):203–216, 2007.

[76] Yuval Emek and David Peleg. Approximating minimum max-stretch spanning trees on unweighted graphs. *SIAM Journal of Computing*, 38(5):1761–1781, 2008.

[77] David Eppstein. Spanning trees and spanners. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, pages 425–461. Elsevier, 2000.

[78] David Eppstein and Kevin A. Wortman. Minimum dilation stars. In *Proceedings of the 21st Annual Symposium on Computational Geometry*, SoCG 2005.

[79] Paul Erdős. On some extremal problems in graph theory. *Israel Journal of Mathematics*, 3:113–116, 1965.

[80] Philippe Esling and Carlos Agón. Time-series data mining. *ACM Comput. Surv.*, 45(1):12:1–12:34, 2012.

[81] Mohammad Farshi, Panos Giannopoulos, and Joachim Gudmundsson. Improving the stretch factor of a geometric network by edge augmentation. *SIAM Journal of Computing*, 38(1):226–240, 2008.

[82] Sándor P. Fekete and Jana Kremer. Tree spanners in planar graphs. *Discrete Applied Mathematics*, 108(1-2):85–103, 2001.

[83] Arnold Filtser and Omrit Filtser. Static and streaming data structures for Fréchet distance queries. In *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021*, pages 1150–1170. SIAM, 2021.

[84] Arnold Filtser, Omrit Filtser, and Matthew J. Katz. Approximate nearest neighbor for curves - simple, efficient, and deterministic. In *47th International Colloquium on Automata, Languages, and Programming, ICALP 2020*, volume 168 of *LIPIcs*, pages 48:1–48:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[85] Arnold Filtser and Shay Solomon. The greedy spanner is existentially optimal. *SIAM Journal on Computing*, 49(2):429–447, 2020.

[86] Omrit Filtser. Universal approximate simplification under the discrete Fréchet distance. *Inf. Process. Lett.*, 132:22–27, 2018.

[87] Fedor V. Fomin, Petr A. Golovach, and Erik Jan van Leeuwen. Spanners of bounded degree graphs. *Information Processing Letters*, 111(3):142–144, 2011.

[88] Fabrizio Frati, Serge Gaspers, Joachim Gudmundsson, and Luke Mathieson. Augmenting graphs to minimize the diameter. *Algorithmica*, 72(4):995–1010, 2015.

[89] Bin Fu, Robert T. Schweller, and Tim Wylie. Discrete planar map matching. In *Proceedings of the 31st Canadian Conference on Computational Geometry, CCCG 2019*, pages 218–224, 2019.

[90] Alexander Gavruskin, Bakhadyr Khoussainov, Mikhail Kokho, and Jiamou Liu. Dynamic algorithms for monotonic interval scheduling problem. *Theor. Comput. Sci.*, 562:227–242, 2015.

[91] Panos Giannopoulos, Rolf Klein, Christian Knauer, Martin Kutz, and Dániel Marx. Computing geometric minimum-dilation graphs is NP-hard. *International Journal of Computational Geometry & Applications*, 20(2):147–173, 2010.

[92] Panos Giannopoulos, Christian Knauer, and Dániel Marx. Minimum-dilation tour (and path) is NP-hard. In *Proceedings of the 23rd European Workshop on Computational Geometry*, EuroCG 2007.

[93] Omer Gold and Micha Sharir. Dynamic time warping and geometric edit distance: Breaking the quadratic barrier. *ACM Trans. Algorithms*, 14(4):50:1–50:17, 2018.

[94] Teofilo F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theor. Comput. Sci.*, 38:293–306, 1985.

[95] Lee-Ad Gottlieb. A light metric spanner. In *Proceedings of the 56th Symposium on Foundations of Computer Science, FOCS*, pages 759–772, 2015.

[96] Ulrike Große, Christian Knauer, Fabian Stehn, Joachim Gudmundsson, and Michiel H. M. Smid. Fast algorithms for diameter-optimally augmenting paths and trees. *International Journal of Foundations of Computer Science*, 30(2):293–313, 2019.

[97] Joachim Gudmundsson, Michael Horton, John Pfeifer, and Martin P. Seybold. A practical index structure supporting Fréchet proximity queries among trajectories. *ACM Trans. Spatial Algorithms Syst.*, 7(3):15:1–15:33, 2021.

[98] Joachim Gudmundsson and Christian Knauer. Dilation and detours in geometric networks. In Teofilo F. Gonzalez, editor, *Handbook of Approximation Algorithms and Metaheuristics*. Chapman and Hall/CRC, 2007.

[99] Joachim Gudmundsson and Yuan Sha. Algorithms for radius-optimally augmenting trees in a metric space. In *Proceedings of the 17th Algorithms and Data Structures Symposium*, WADS 2021.

[100] Joachim Gudmundsson, Yuan Sha, and Sampson Wong. Approximating the packedness of polygonal curves. In Yixin Cao, Siu-Wing Cheng, and Minming Li, editors, *31st International Symposium on Algorithms and Computation, ISAAC 2020*, volume 181 of *LIPIcs*, pages 9:1–9:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[101] Joachim Gudmundsson, Yuan Sha, and Fan Yao. Augmenting graphs to minimize the radius. In *Proceedings of the 32nd International Symposium on Algorithms and Computation*, ISAAC 2021.

[102] Joachim Gudmundsson and Michiel H. M. Smid. On spanners of geometric graphs. *International Journal of Foundations of Compututer Science*, 20(1):135–149, 2009.

[103] Joachim Gudmundsson and Michiel H. M. Smid. Fast algorithms for approximate Fréchet matching queries in geometric trees. *Comput. Geom.*, 48(6):479–494, 2015.

[104] Joachim Gudmundsson, Andreas Thom, and Jan Vahrenhold. Of motifs and goals: mining trajectory data. In *SIGSPATIAL 2012 International Conference on Advances in Geographic Information Systems, SIGSPATIAL 2012*, pages 129–138. ACM, 2012.

[105] Joachim Gudmundsson and Nacho Valladares. A GPU approach to subtrajectory clustering using the Fréchet distance. *IEEE Trans. Parallel Distributed Syst.*, 26(4):924–937, 2015.

[106] Joachim Gudmundsson, André van Renssen, Zeinab Saeidi, and Sampson Wong. Translation invariant Fréchet distance queries. *Algorithmica*, 83(11):3514–3533, 2021.

[107] Joachim Gudmundsson and Thomas Wolle. Football analysis using spatio-temporal tools. *Comput. Environ. Urban Syst.*, 47:16–27, 2014.

[108] Joachim Gudmundsson and Sampson Wong. Improving the dilation of a metric graph by adding edges. *ACM Transactions on Algorithms*, 18(3):20:1–20:20, 2022.

[109] Sariel Har-Peled and Mitchell Jones. Proof of Dudley's convex approximation. *arXiv preprint arXiv:1912.01977*, 2019.

[110] Sariel Har-Peled and Manor Mendel. Fast construction of nets in low-dimensional metrics and their applications. *SIAM Journal of Computing*, 35(5):1148–1184, 2006.

[111] Mahdi Hashemi and Hassan A. Karimi. A critical review of real-time map-matching algorithms: Current issues and future directions. *Comput. Environ. Urban Syst.*, 48:153–165, 2014.

[112] Jan-Henrik Haunert and Wouter Meulemans. Partitioning polygons via graph augmentation. In Jennifer A. Miller, David O'Sullivan, and Nancy Wiegand, editors, *Proceedings of the 9th Geographic Information Science, GIScience*, volume 9927, pages 18–33, 2016.

[113] Ferran Hurtado and Csaba D Tóth. Plane geometric graph augmentation: a generic perspective. In *Thirty Essays on Geometric Graph Theory*, pages 327–354. Springer, 2013.

[114] Piotr Indyk. Approximate nearest neighbor algorithms for frechet distance via product metrics. In *Proceedings of the 18th Symposium on Computational Geometry, SoCG 2002*, pages 102–106. ACM, 2002.

[115] Christopher Johnson and Haitao Wang. A linear-time algorithm for radius-optimally augmenting paths in a metric space. *Computational Geometry*, 96:101759, 2021.

[116] Koen Klaren. Continuous dynamic time warping for clustering curves. Master's thesis, Eindhoven University of Technology, 2020.

[117] Rolf Klein, Christian Knauer, Giri Narasimhan, and Michiel H. M. Smid. On the dilation spectrum of paths, cycles, and trees. *Computational Geometry*, 42(9):923–933, 2009.

[118] Matej Kubicka, Arben Çela, Hugues Mounier, and Silviu-Iulian Niculescu. Comparative study and application-oriented classification of vehicular map-matching methods. *IEEE Intell. Transp. Syst. Mag.*, 10(2):150–166, 2018.

[119] Hung Le and Shay Solomon. Truly optimal Euclidean spanners. In *Proceedings of the 60th Annual Symposium on Foundations of Computer Science*, FOCS 2019.

[120] Jae-Gil Lee, Jiawei Han, and Kyu-Young Whang. Trajectory clustering: a partition-and-group framework. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Beijing, China, June 12-14, 2007*, pages 593–604. ACM, 2007.

[121] Lan Lin and Yixun Lin. Optimality computation of the minimum stretch spanning tree problem. *Applied Mathematics and Computation*, 386:125502, 2020.

[122] Jun Luo and Christian Wulff-Nilsen. Computing best and worst shortcuts of graphs embedded in metric spaces. In *Proceedings of the 19th International Symposium on Algorithms and Computation*, ISAAC 2008.

[123] Ting Luo, Xinwei Zheng, Guangluan Xu, Kun Fu, and Wenjuan Ren. An improved DBSCAN algorithm to detect stops in individual trajectories. *ISPRS Int. J. Geo Inf.*, 6(3):63, 2017.

[124] Anil Maheshwari, Jörg-Rüdiger Sack, and Christian Scheffer. Approximating the integral Fréchet distance. *Comput. Geom.*, 70-71:13–30, 2018.

[125] Jessica Meade, Dora Biro, and Tim Guilford. Homing pigeons develop local route stereotypy. *Proceedings of the Royal Society B: Biological Sciences*, 272(1558):17–23, 2005.

[126] Nimrod Megiddo. Applying parallel computation algorithms in the design of serial algorithms. *J. ACM*, 30(4):852–865, 1983.

[127] Andranik Mirzaian and Eshrat Arjomandi. Selection in X+Y and matrices with sorted rows and columns. *Information Processing Letters*, 20(1):13–17, 1985.

[128] Joseph S. B. Mitchell and Christos H. Papadimitriou. The weighted region problem: Finding shortest paths through a weighted planar subdivision. *J. ACM*, 38(1):18–73, 1991.

[129] Meinard Müller. Dynamic time warping. In *Information Retrieval for Music and Motion*, pages 69–84. Springer, 2007.

[130] Wolfgang Mulzer. Minimum dilation triangulations for the regular n-gon. *Master's thesis Freie Universität Berlin, Germany*, 2004.

[131] Mario E. Munich and Pietro Perona. Continuous dynamic time warping for translation-invariant curve alignment with applications to signature verification. In *Proceedings of the International Conference on Computer Vision, 1999*, pages 108–115. IEEE Computer Society, 1999.

[132] Cory Myers, Lawrence Rabiner, and Aaron Rosenberg. Performance tradeoffs in dynamic time warping algorithms for isolated word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(6):623–635, 1980.

[133] Giri Narasimhan and Michiel H. M. Smid. *Geometric spanner networks*. Cambridge University Press, 2007.

[134] Giri Narasimhan and Michiel H. M. Smid. *Geometric spanner networks*. Cambridge University Press, 2007.

[135] David Peleg. *Distributed Computing: a Locality-Sensitive Approach*. SIAM, 2000.

[136] David Peleg. Low stretch spanning trees. In *Proceedings of the 27th International Symposium of Mathematical Foundations of Computer Science*, MFCS 2002.

[137] Mohammed A Quddus, Washington Y Ochieng, and Robert B Noland. Current map-matching algorithms for transport applications: State-of-the art and future research directions. *Transportation research part c: Emerging technologies*, 15(5):312–328, 2007.

[138] Peter Ranacher and Katerina Tzavella. How to compare movement? A review of physical movement similarity measures in geographic information science and beyond. *Cartography and Geographic Information Science*, 41(3):286–307, 2014.

[139] Aviad Rubinstein. Hardness of approximate nearest neighbor search. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018*, pages 1260–1268. ACM, 2018.

[140] Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):43–49, 1978.

[141] Otfried Schwarzkopf and Jules Vleugels. Range searching in low-density environments. *Inf. Process. Lett.*, 60(3):121–127, 1996.

[142] Pavel Senin. Dynamic time warping algorithm review. *Information and Computer Science Department University of Hawaii at Manoa Honolulu, USA*, 855(1-23):40, 2008.

[143] Bruno Serra and Marc Berthod. Subpixel contour matching using continuous dynamic programming. In *Conference on Computer Vision and Pattern Recognition, CVPR 1994*, pages 202–207. IEEE, 1994.

[144] Martin P. Seybold. Robust map matching for heterogeneous data via dominance decompositions. In *Proceedings of the 2017 SIAM International Conference on Data Mining, SDM 2017*, pages 813–821. SIAM, 2017.

[145] Junichi Shigezumi, Tatsuya Asai, Hiroaki Morikawa, and Hiroya Inakoshi. A fast algorithm for matching planar maps with minimum Fréchet distances. In *Proceedings of the 4th International ACM SIGSPATIAL Workshop on Analytics for Big Geospatial Data, BigSpatial@SIGSPATIAL 2015*, pages 25–34. ACM, 2015.

[146] Daniel Dominic Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983.

[147] Michiel H. M. Smid. Closest-point problems in computational geometry. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, pages 877–935. Elsevier, 2000.

[148] E. Sriraghavendra, K. Karthik, and Chiranjib Bhattacharyya. Fréchet distance based approach for searching online handwritten documents. In *9th International Conference on Document Analysis and Recognition, ICDAR 2007*, pages 461–465. IEEE Computer Society, 2007.

[149] Panagiotis Tampakis, Nikos Pelekis, Christos Doulkeridis, and Yannis Theodoridis. Scalable distributed subtrajectory clustering. In *2019 IEEE International Conference on Big Data (Big Data)*, pages 950–959. IEEE, 2019.

[150] Bo Tang, Man Lung Yiu, Kyriakos Mouratidis, and Kai Wang. Efficient motif discovery in spatial trajectories using discrete Fréchet distance. In *Proceedings of the 20th International Conference on Extending Database Technology, EDBT 2017, Venice, Italy, March 21-24, 2017*, pages 378–389. OpenProceedings.org, 2017.

[151] Yaguang Tao, Alan Both, Rodrigo I Silveira, Kevin Buchin, Stef Sijben, Ross S Purves, Patrick Laube, Dongliang Peng, Kevin Toohey, and Matt Duckham. A comparative analysis of trajectory similarity measures. *GIScience & Remote Sensing*, pages 1–27, 2021.

[152] Charles C. Tappert, Ching Y. Suen, and Toru Wakahara. The state of the art in online handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(8):787–808, 1990.

[153] Stephen J Taylor. *Modelling financial time series*. World Scientific, 2008.

[154] Mikkel Thorup and Uri Zwick. Approximate distance oracles. *Journal of the ACM*, 52(1):1–24, 2005.

[155] Kevin Toohey and Matt Duckham. Trajectory similarity measures. *ACM SIGSPATIAL Special*, 7(1):43–50, 2015.

[156] Taras K Vintsyuk. Speech discrimination by dynamic programming. *Cybernetics*, 4(1):52–57, 1968.

[157] Haitao Wang. An improved algorithm for diameter-optimally augmenting paths in a metric space. *Computational Geometry*, 75:11–21, 2018.

[158] Haitao Wang and Yiming Zhao. A linear-time algorithm for discrete radius optimally augmenting paths in a metric space. *International Journal of Computational Geometry and Applications*, 30(3&4):167–182, 2020.

[159] Haitao Wang and Yiming Zhao. Algorithms for diameters of unicycle graphs and diameter-optimally augmenting trees. *Theoretical Computer Science*, 890:192–209, 2021.

[160] Xiaoyue Wang, Abdullah Mueen, Hui Ding, Goce Trajcevski, Peter Scheuermann, and Eamonn J. Keogh. Experimental comparison of representation methods and distance measures for time series data. *Data Min. Knowl. Discov.*, 26(2):275–309, 2013.

[161] Hong Wei, Yin Wang, George Forman, and Yanmin Zhu. Map matching by Fréchet distance and global weight optimization. *Technical Paper, Departement of Computer Science and Engineering*, page 19, 2013.

[162] Hong Wei, Yin Wang, George Forman, and Yanmin Zhu. Map matching: comparison of approaches using sparse and noisy data. In *Proceedings of the 21st SIGSPATIAL International Conference on Advances in Geographic Information Systems, SIGSPATIAL 2013*, pages 434–437. ACM, 2013.

[163] Carola Wenk, Randall Salas, and Dieter Pfoser. Addressing the need for map-matching speed: Localizing Globalb curve-matching algorithms. In *Proceedings of the 18th International Conference on Scientific and Statistical Database Management, SSDBM 2006*, pages 379–388. IEEE Computer Society, 2006.

[164] Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theor. Comput. Sci.*, 348(2-3):357–365, 2005.

[165] Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the International Congress of Mathematicians: Rio de Janeiro 2018*, pages 3447–3487. World Scientific, 2018.

[166] Christian Wulff-Nilsen. Computing the dilation of edge-augmented graphs in metric spaces. *Computational Geometry*, 43(2):68–72, 2010.

[167] Tim Wylie and Binhai Zhu. Intermittent map matching with the discrete Fréchet distance. *CoRR*, abs/1409.2456, 2014.

[168] Öz Yilmaz. *Seismic data analysis: Processing, inversion, and interpretation of seismic data*. Society of exploration geophysicists, 2001.

[169] Yu Zheng and Xiaofang Zhou, editors. *Computing with Spatial Trajectories*. Springer, 2011.